

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Smart Evaluation

propusă de

Ionuț Lungeanu

Sesiunea: iulie, 2019

Coordonator științific

Prof. Colab. Florin Olariu

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ

Smart Evaluation

Ionuț Lungeanu

Sesiunea: iulie, 2019

Coordonator științific

Prof. Colab. Florin Olariu

Avizat,
Îndrumător lucrare de licență,
Prof. Colab. Florin Olariu.

Data: Semnătura:

Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Lungeanu Ionuț** domiciliat în **România, Sat Braniștea, Jud. Galați, str. Rândunelelor, nr.3**, născut la data de **08 mai 1996**, identificat prin CNP **1960508170055**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2018, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Smart Evaluation** elaborată sub îndrumarea domnului **Prof. Colab. Florin Olariu**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data:

Semnătura:

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Smart Evaluation**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Ionuț Lungeanu**

Data:

Semnătura:

Cuprins

Introducere	2
Motivația alegerii temei	2
Gradul de noutate a temei	2
Obiectivele generale ale lucrării	3
Descrierea sumară a soluției	3
Metodologia folosită	3
Structura lucrării	4
1 Contribuții	5
2 Descrierea problemei	6
3 Abordări anterioare	8
4 Descrierea soluției	9
4.1 Structura proiectului	9
4.2 Scenarii posibile	12
4.3 Proiectarea și implementarea bazei de date	13
4.4 Detalii de implementare	15
4.5 Funcționalitățile aplicației	24
Concluzii	28
Bibliografie	29
Anexa 1	30
Anexa 2	31
Anexa 3	32

Introducere

Motivația alegerii temei

Ideea implementării acestei aplicații a pornit de la faptul că, printr-un exercițiu de imaginație, am considerat că procesul clasic de evaluare, pe hârtie, poate fi îmbunătățit considerabil, reducând efortul profesorilor în crearea examenului propriu-zis, dar și efortul studenților în rezolvarea lui. Așadar, aplicația în cauză oferă profesorilor posibilitatea de a crea un examen rapid, printr-un procedeu simplu, iar studenților de a-l rezolva cu mai multă comoditate.

Gradul de noutate a temei

Deși evaluarea pe computer este mai benefică, nu multe universități din lume au încercat să adopte acest mod de a testa cunoștințele studenților. Începând din anul universitar 2016-2017, Universitatea Cambridge a introdus un proiect pilot, care să facă trecerea de la evaluarea scrisă la cea pe un computer. În iulie 2018, studenții din cadrul facultăților de istorie și arte clasice au testat această modalitate de evaluare, dar reprezentanții universității au extins testarea ei pentru încă trei ani, dorind să stabilească dacă situații precum zgomotul tastaturilor, repartizarea studenților pe săli sau furnizarea energiei electrice vor fi un impediment real pentru studenți, profesori sau supraveghetori. În cadrul Facultății de Informatică Iași, au fost susținute teste pe computer, în cadrul materiilor de Practica Python, Baze de date sau Introducere în Programare, rezultatele fiind afișate după rezolvarea testului.

Obiectivele generale ale lucrării

Prin implementarea acestei aplicații am încercat să ofer o alternativă mai bună și plăcută examinării studenților în procesul de evaluare de la sfârșitul fiecărui semestru. Pe lângă acest lucru sistemul de examinare propus reprezintă și o variantă mai tehnologizată a procesului clasic.

Descrierea sumară a soluției

Soluția propusă poate fi privită din două perspective: perspectiva profesorului și cea a studentului. Profesorul poate adăuga întrebări pe parcursul întregului anului universitar, iar în timpul sesiunii crează un examen nou. Acesta este creat folosind un algoritm de selecție a întrebărilor, în funcție de timpul, dificultatea și numărul întrebărilor dorite de profesor. Grupa de studenți cărora le-a fost asignat examenul se poate apuca de rezolvarea acestuia. Fiecare student propune o variantă de rezolvare a subiectelor, urmând ca nota să-i fie acordată după corectarea examenului de către profesor.

Metodologia folosită

Prima parte a lucrării a constituit-o cercetarea și căutarea universităților cu învățământ la zi care au un sistem de evaluare online și au renunțat, măcar parțial, la evaluarea clasică pe hârtie. Observând că nu multe universități au făcut acest lucru, am decis să implementez această aplicație, împărțind funcționalitățile de bază de cele care aduc o îmbunătățire a sistemului, dar nu constituie baza acestuia. Din punctul meu de vedere partea cea mai importantă oricărei aplicații este baza de date și lucrul cu acestea, de aceea am încercat să realizez o bază de date relațională cât mai eficientă posibil. După ce am pus bazele aplicației, am început implementarea funcționalităților de bază pentru rolul profesorului, iar apoi pentru cele ale studentului. În cele din urmă, după ce am realizat și template-ul interfeței grafice, următorul pas a fost stilizarea acesteia și implementarea funcționalităților din categoria "nice to have".

Structura lucrării

Lucrarea scrisă este structurată în 3 capitole. Primul capitol se numește Descrierea problemei, în care sunt prezentate avantajele și dezavantajele evaluării online. Al doilea capitol, Abordări anterioare și contribuții, conține atât aplicații similare altor aplicații deja implementate, dar și funcționalități noi, diferite de ceea ce este pe piață. În capitolul Descrierea soluției sunt prezentate detalii ce țin de arhitectura aplicației, de implementare și de utilizare.

Capitolul 1

Contribuții

Folosind tehnologiile de Front-End și Back-End am propus o rezolvare a problemei descrisă în capitolul *Descrierea soluției*. Față de aplicațiile deja prezente pe piață, aplicația Smart Evaluation are în plus pe lângă managementul examenelor și un sistem integrat de generare ale lor.

Pentru crearea aplicației Smart Evaluation am folosit web framework-ul ASP.NET CORE (vezi Anexa 1), împreună cu cele patru componente ale acestuia: Entity Framework(EF), Identity, MVC și Razor. Am ales acest framework datorită modularității oferite și posibilității de a dezvolta aplicații și a le rula pe Windows, macOS și Linux. De asemenea realizarea acestei aplicații m-a ajutat să-mi consolidez cunoștințele pentru crearea de aplicații web, și să lucrez cu unele dintre cele mai noi tehnologii de pe piață în acest domeniu.

Capitolul 2

Descrierea problemei

Dezavantajele procesului de evaluare pe hârtie și compararea sa cu evaluarea online au fost des dezbătute și abordate în mai multe studii de specialitate. Printre dezavantajele evaluării pe hârtie se numără imposibilitatea susținerii examenului de către studenții care nu pot ajunge fizic în sala de examen, dar și faptul că procesul este unul neprietenos cu mediul înconjurător.

În primul rând, un sistem de evaluare online asigură flexibilitatea și securitatea procesului de examinare. După ce subiectele examenului sunt adăugate în sistem, acesta poate amesteca și genera examene cu întrebări diferite pentru orice student, reducând posibilitatea împărțirii răspunsurilor între studenți. Totodată, folosind varianta clasică de examinare, timpul de creare a unor examene unice pentru fiecare student în parte este mai lung, nefiind avantajos în gestionarea timpului profesorului. De asemenea, se evită scurgerile de informații, iar cazurile în care întrebările sau răspunsurile unui examen riscă să ajungă în posesia unui examinat ar fi minime, nemaifiind necesară printarea, scanarea sau distribuirea manuală a subiectelor. Acest fapt reduce totodată și costurile, scăzând consumul de hârtie și tuș pentru imprimantă.

În al doilea rând, pe lângă procesul de creare a examenului, acest sistem aduce îmbunătățiri notării și gestionării rezultatelor studenților. Rezultatele sunt obținute ușor și instant, lucru benefic pentru studenți, având mai mult timp disponibil pentru pregătirea reexaminării. Din punctul de vedere al cadrelor didactice, stocarea rezultatelor într-un tabel completat automat este de asemenea un beneficiu, profesorii fiind scutiți de participarea la procesul de centralizare al rezultatelor, și de procesul corectare, proces ce necesită un efort și o atenție mărită, fiind și predispus la erori.

În al treilea rând, posibilitatea de a susține examenul în altă locație decât sala de

curs este un beneficiu pentru studenții cu probleme locomotorii, cei care studiază în străinătate și nu numai. Acest lucru vine, în schimb, și cu un dezavantaj: în timpul examenului este greu de depistat o posibilă fraudă. În acest scop, unele universități din Marea Britanie și SUA folosesc sistemul de supraveghere ProctorU (vezi Anexa 2).

Un argument mai solid pentru folosirea platformei online de evaluare este reducerea consumului hârtiei, deci implicit a salvării resurselor (pentru o tonă de hârtie sunt necesari 17 copaci, 26.500 litri de apă și 2000 kWh energie electrica), dar și reducerea cheltuielilor pentru achiziționarea acesteia. Totodată, consider că profesorilor le poate fi mai ușor să gestioneze răspunsurile examenelor și notele acestora într-o bază de date bine organizată, în defavoarea unui teanc de hârtii pe birou. Aplicația are și rolul de a reduce timpului alocat pentru testare, elimină necesitatea formulării propriu-zise a lucrării, ea fiind generată folosind un algoritm implementat, dar și distribuirea fizică a acesteia către studenți, la momentul susținerii testului.

Concluzionez primul capitol, *Descrierea produsului*, prin a afirma faptul că, implementarea acestui sistem de evaluare poate aduce, cu siguranță, schimbări benefice în sistemul universitar, iar micile dezavantaje ale acestuia, poate fi înlăturate în timp.

Capitolul 3

Abordări anterioare

O aplicație similară aplicației Smart Evaluation este **tutoreal**¹. Creată de către compania indiană White Speal Services, aplicația tutoreal este disponibilă pe sistemul de operare Android. Printre serviciile sale se numără un sistem de logare și înregistrare, o bază de date cu întrebări din cursuri, un sistem de feedback și sugestii pentru examenele primite. Aplicația tutoreal vine cu un sistem de informații video pentru fiecare examen, în care profesorii pot adăuga precizări privind acesta sau subiectele sale. Facultățile Ambition LAW Institute și KD Campus din India folosesc aplicația tutoreal pentru a examina studenții.

Sistemul de evaluare online este prezent cu preponderență în universitățile cu programe de învățământ fără frecvență sau în susținerea diverselor cursuri online de specializare pe anumite domenii.

¹Pagină originală <http://www.tutoreal.in>

Capitolul 4

Descrierea soluției

4.1 Structura proiectului

În acest capitol voi prezenta structura proiectului, cerințele sistemului, cu ajutorul diagramei Use Case, dar și proiectarea și implementarea bazei de date.

Pentru a implementa Smart Evaluation am folosit modelul arhitectural Model-View-Controller (MVC). Acest design pattern separă aplicația în trei componente principale: Models, Views și Controller. Această conturare permite reutilizarea codului, reduce complexitatea aplicației, fiind mai ușor de a scrie cod și de a-l testa, atunci când fiecare componentă are rolul său și este separată de celelalte componente. Spre exemplu, în interfața grafică se produc schimbări mult mai frecvente decât în celelalte componente. Dacă interfața grafică și regulile business ar fi combinate într-un singur obiect, atunci și regulile business ar trebui modificate de fiecare dată când interfața grafică suferă modificări.

Modelul conține clase ce reprezintă domeniul aplicației. Aceste obiecte gestionează operațiunile logice și de utilizare a informației.

View-ul este responsabil cu afișarea conținutului interfeței grafice care interacționează cu utilizatorul final.

Controller-ul este componenta care se ocupă de interacțiunea cu utilizatorul. În controller se citesc datele introduse de utilizator, se trimit către model, se execută operațiile, după care răspunsul este trimis către componenta View.

Fluxul de lucru cu MVC începe prin interacțiunea utilizatorului cu interfața grafică, datele adăugate de utilizator urmând să fie preluate de către controller. Acesta crează sau actualizează modelul, urmând să fie trimis înapoi către view, care îl utili-

zează pentru a genera o nouă interfață. View-ul primește datele din model, model care nu are cunoștințe directe despre cel dintâi menționat.

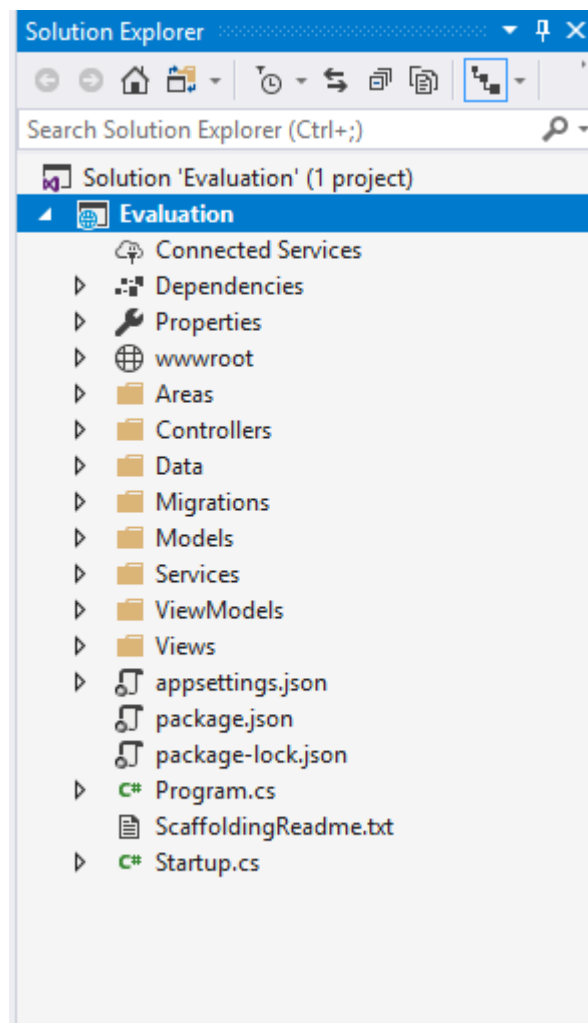


Figura 4.1: Structura proiectului

Proiectul Smart Evaluation, pe lângă Model, View și Controller, mai are în structura sa următoarele componente:

- **Areas** conține sistemul ASP.NET CORE Identity (vezi cap.2.2) care adaugă funcționalitățile de autentificare (înregistrare și logare) și autorizare a utilizatorilor.
- **Data** cuprinde scripturile folosite pentru reactualizarea bazei de date și clasa *DbContext* care este responsabilă cu interacțiunea cu datele.
- **Migrations** este alcătuit din clasele rezultate după aplicarea migrațiilor.
- **Services** conține metode folosite în componenta Controller cu scopul de a oferi codului o modularizare mai ridicată.

- **ViewModels** conține modelele folosite pentru a afișa informațiile în anumite view-uri. Diferența dintre **Model** și **ViewModel** este că cel din urmă conține date din mai multe modele grupate într-un singur obiect, necesare unui anumit **View**.

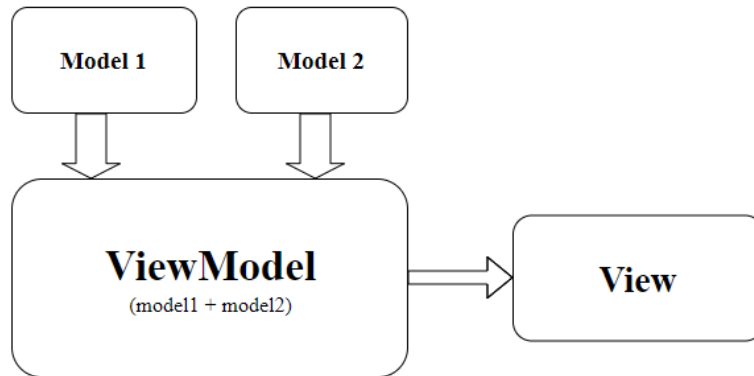


Figura 4.2: View Models

- Directorul **wwwroot** este directorul rădăcină pentru orice conținut static servit peste web. Acesta este împărțit în mai multe subdirectoare, fiecare subdirector reprezentând resursele grafice respective precum *css*, *javascript*, *Bootstrap*, *Font Awesome* etc.

4.2 Scenarii posibile

După cum se poate observa în diagrama Use Case de la Figura 1.3, aplicația Smart Evaluation are doi actori. Primul actor, profesorul, poate efectua două operații principale: administrarea întrebărilor propuse pentru examen, cu operațiile de a crea, a edita și de a șterge, și administrarea examenelor. Examenul este generat în funcție de valorile introduse de profesor pentru numărul de întrebări dorite, dificultatea și timpul examenului. Următorul pas este de a trimite examenul grupei de studenți alese de profesor.

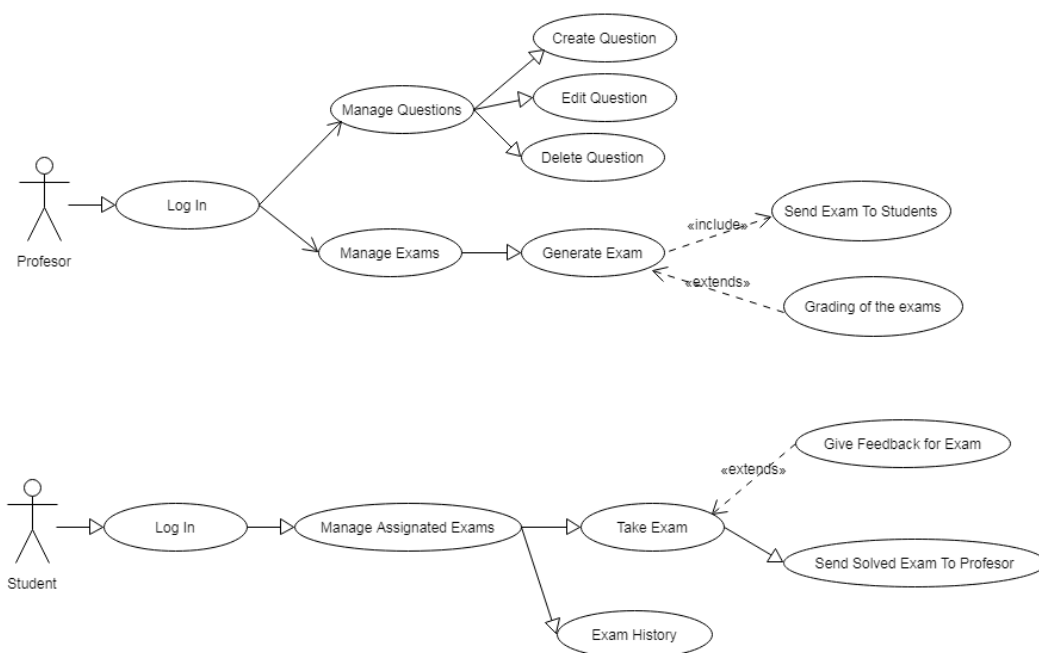


Figura 4.3: Diagrama Use Case

Cel de-al doilea actor, studentul, după logare își poate vedea situația examenelor rezolvate și nerezolvate. Pentru examenele rezolvate, el poate vedea rezolvarea pe care a propus-o, iar pentru cele nerezolvate poate propune o rezolvare, mai apoi aceasta fiind trimisă împreună cu un feedback opțional vizând examenul, profesorului de curs. Profesorul va corecta lucrarea și va acorda o notă. Acesta este fluxul de lucru al aplicației.

4.3 Proiectarea și implementarea bazei de date

Pentru a implementa baza de date am folosit sistemul de gestionare al bazelor de date relațional denumit Microsoft SQL Server. Acest sistem stochează datele pe modelul relațional, sub forma unei colecții de tabele cu rânduri și coloane. Tipurile coloanelor suportate de către SQL Server sunt atât tipuri primare, precum caracter, zecimal, dată calendaristică, dar și tipuri mai complexe ca și datele binare, datele geometrice, XML, etc. Microsoft SQL Server folosește T-SQL(Transact-SQL), care extinde SQL(Structured Query Language), adaugându-i noi funcționalități precum tratarea excepțiilor, declararea variabilelor sau programarea procedurală.

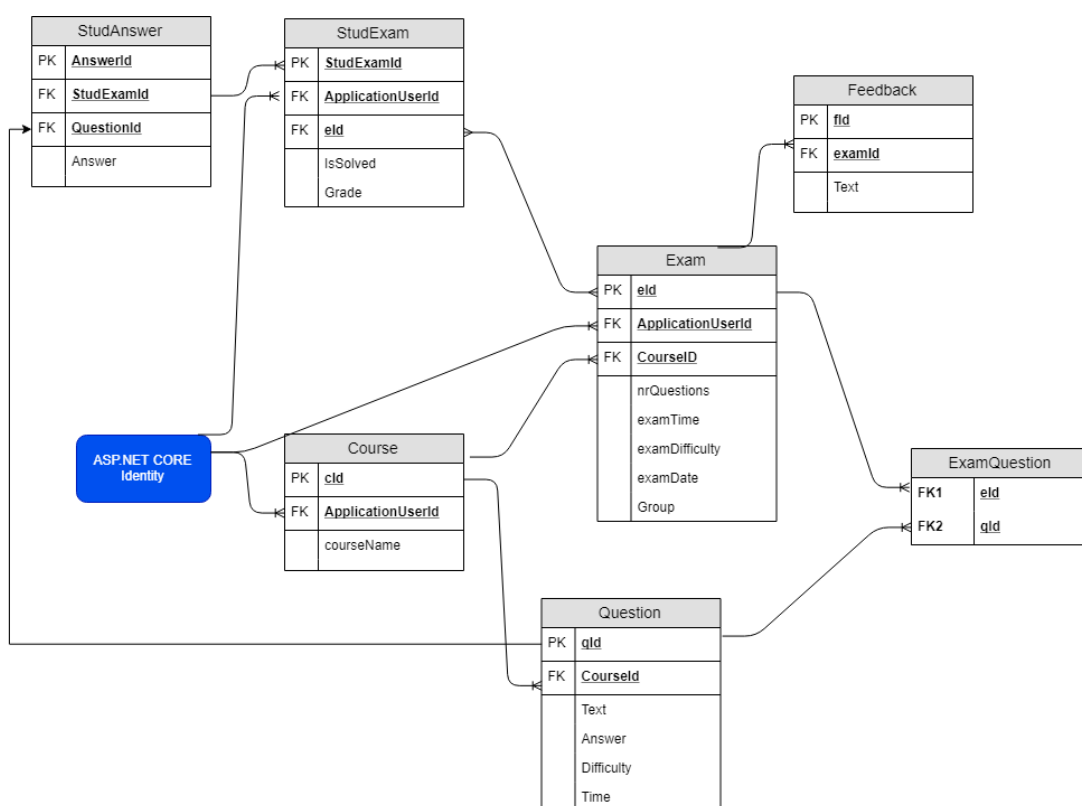


Figura 4.4: Structura bazei de date

După cum se poate observa și în schema de mai sus (Fig 3.4), exceptând sistemul de logare implementat folosind ASP.NET Core Identity (vezi Anexa 3), am creat alte 7 tabele care relaționează între ele folosind relația one-to-many.

Tabela *Course* conține toate cursurile predate de către un profesor. Aceasta relaționează cu tabela ASP.NET CORE Identity (vezi Anexa 3) folosind relația one-to-many deoarece un profesor poate preda unul sau mai multe cursuri, iar cu tabelele Exam și Question folosind aceeași relație, deoarece pentru un curs pot exista mai multe examene și întrebări create.

Tabela *Exam* conține toate examenele create de către un profesor. Câmpurile tabelii reprezintă caracteristicile examenului, precum numărul de întrebări, durata, dificultatea, data creării, și grupa, care vor fi completate de către profesor atunci când crează un nou examen.

În tabela *Question* sunt înregistrate toate întrebările adăugate de către un profesor. Aceasta conține câmpurile *Text* și *Answer*, reprezentând textul și răspunsul întrebării, și câmpurile *Difficulty* și *Time* reprezentând dificultatea și timpul necesar rezolvării acesteia.

Tabela *ExamQuestion* este o tabelă trivială, care ajută la implementarea relației many-to-many dintre tabelele *Exam* și *Question*. Această relație nu poate fi implementată direct în bazele de date relaționale, de aceea am împărțit-o în două relații de tipul one-to-many. Această tabelă se populează cu toate întrebările ce compun un examen.

Tabelele *StudExam* și *StudAnswer* conțin toate examenele asigurate de către un profesor unui student, respectiv toate rezolvările propuse de către un student subiectelor unui examen primit. Pe lângă cheile primare și străine, *StudExam* are în componența sa și câmpurile *Grade* reprezentând nota studentului la examen și *IsSolved* folosit pentru a oferi informația dacă examenul a fost rezolvat de student sau nu.

În tabela *Feedback* este formată din feedback-ul trimis de către după încheierea unui examen. Această tabelă are relația one-to-many cu tabela *Exam*, deoarece un examen poate avea unul sau mai multe feedback-uri.

Proiectarea și implementarea acestei scheme a bazei de date a făcut posibilă interogarea acesteia, precum și efectuarea operațiilor CRUD (Create, Read, Update, Delete) peste entitățile relaționale.

4.4 Detalii de implementare

Folosind ASP.NET CORE Identity am realizat un sistem de logare pentru aplicația Smart Evaluation cu redirectionare bazată pe rolurile utilizatorilor. Primul pas a fost să creez rolurile dorite (*Listing 1*), apoi să redirectionez utilizatorii în funcție de acestea (*Listing 2*).

```
public static void CreateRole(IServiceProvider serviceProvider,
                             string role)
{
    var roleManager = serviceProvider
        .GetRequiredService<RoleManager<IdentityRole>>();

    Task<bool> roleExists = roleManager.RoleExistsAsync(role);
    roleExists.Wait();

    if (!roleExists.Result)
    {
        Task<IdentityResult> roleResult = roleManager
            .CreateAsync(new IdentityRole(role));

        roleResult.Wait();
    }
}
```

Listing 1: Crearea rolurilor

Parametrul *role*, reprezintă numele rolului care este adăugat, funcția fiind apelată în fișierul *Startup.cs*. Funcția *RoleExistsAsync(role)* verifică dacă rolul este deja creat, dacă rezultatul este fals se apelează funcția *CreateAsync(newIdentityRole(role))* și se crează un nou rol.

```

var result = await _signInManager.PasswordSignInAsync
    (Input.Email,
     Input.Password,
     Input.RememberMe,
     lockoutOnFailure: true);

if (result.Succeeded)
{
    ApplicationUser user = await _signInManager
        .userManager.FindByEmailAsync(Input.Email);
    if(await _signInManager.UserManager.IsInRoleAsync(user, "Professor"))
    {
        _logger.LogInformation("User logged in.");
        return Redirect("~/Professor/Index");
    }
    else if(await _signInManager.UserManager.IsInRoleAsync(user, "Student"))
    {
        _logger.LogInformation("User logged in.");
        return Redirect("~/Students/LogIn");
    }
}

```

Listing 2: Funcția pentru logare bazată pe roluri

Funcția *PasswordSignInAsync* încearcă realizarea autentificării printr-o operație asincronă folosind emailul și parola preluate din formularul afișat prin componenta View. Ultimul parametru al funcției, *lockoutOnFailure*, este un flag care setează suspendarea contului pe o anumită perioadă de timp, în cazul în care se execută anumite autentificări eșuate.

```

<div class="form-group">
  <input asp-for="Input.Email" class="form-control" />
  <span asp-validation-for="Input.Email" class="text-danger"> </span>
</div>
<div class="form-group">
  <input asp-for="Input.Password" class="form-control" />
  <span asp-validation-for="Input.Password" class="text-danger"> </span>
</div>

```

Listing 3: Formularul pentru logare

Alte servicii implementate folosind ASP.NET CORE Identity pentru tratarea autentificării sunt confirmarea emailului după înregistrare, deconectarea, resetarea parolei etc. Aceste metode sunt implementate în clasele din figura următoare (Fig. 3.5).

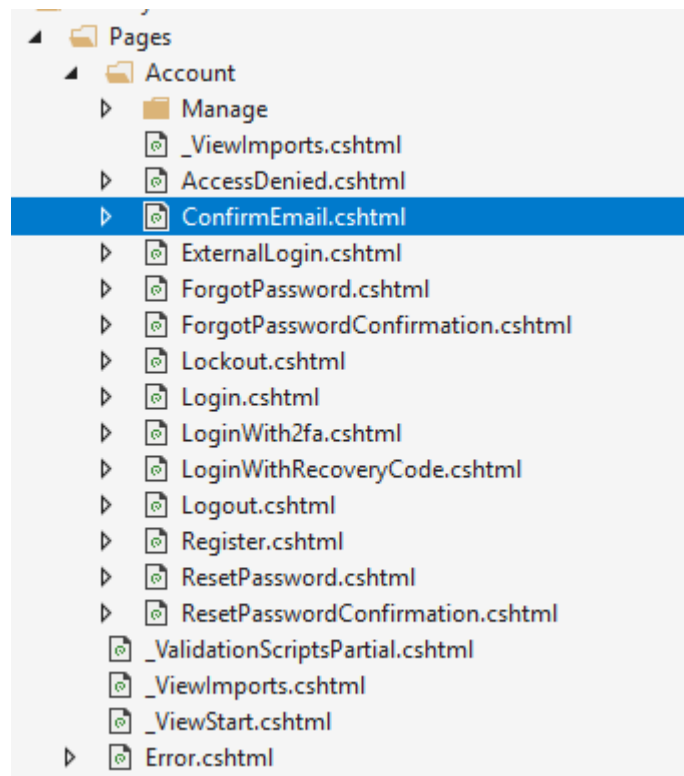


Figura 4.5: ASP.NET Core Identity

Funcția Index returnează o listă cu toate examenele create de către profesorul autentificat. Atributul de metodă *[Authorize]* face posibilă interzicerea utilizatorilor cu roluri diferite decât cel menționat să apeleze funcția de mai jos. Variabila *userId* stochează id-ul profesorului logat, iar în *examList* se salvează lista examenelor sale.

```

[Authorize(Roles = "Profesor")]
public IActionResult Index(ExamCourse examCourse)
{
    string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var examList = _context.Exam.Where(m=> m.ApplicationUserId == userId)
        .Select(x => new ExamCourse
        {
            examDate = x.examDate,
            examDifficulty = x.examDifficulty,
            Group = x.Group,
            courseName = x.Course.courseName,
            eId = x.eId
        });
    return View(examList.ToList());
}

```

Listing 4: Interogarea examenelor create de profesorul logat

Deoarece am avut nevoie să afișez date din mai multe entități, am creat un View-Model denumit *ExamCourse*, definit mai jos.

```

public class ExamCourse
{
    public DateTime examDate { get; set; }
    public int examDifficulty { get; set; }
    public int? Group { get; set; }
    public string courseName { get; set; }
    public int eId { get; set; }
}

```

Listing 5: ViewModelul ExamCourse

Același principiu l-am folosit și pentru afișarea detaliilor unui examen, id-ul unic al său fiind primit ca parametru al funcției Details. Această funcție are rolul de a centraliza pentru fiecare examen situația studenților.

```
[Authorize(Roles = "Profesor")]  
public IActionResult Details(int? id)
```

Listing 6: Antetul funcției Details

Pentru crearea unui examen am folosit proprietatea *[Bind]*, pentru a proiecta împotriva acțiunii de *"overposting"*. După primirea parametrilor, precum numărul de întrebări, durata și dificultatea examenului, din formularul afișat în View, voi crea o listă cu întrebările cele mai optime pentru examen.

```
List<Question> list = new List<Question>(nrquestions);  
  
var easyQuestion = _context.Question  
    .Where(q => q.Difficulty == ((examDifficulty * 2) - 1)  
    && Enumerable.Range(avgTime - 3, avgTime + 3)  
    .Contains(q.Time) == true && q.CourseId == courseId)  
    .ToList().FirstOrDefault();  
  
var normalQuestions = _context.Question  
    .Where(q => q.Difficulty == (examDifficulty * 2)  
    && Enumerable.Range(avgTime - 3, avgTime + 3)  
    .Contains(q.Time) == true && q.CourseId == courseId)  
    .ToList().Take(nrquestions - 2);  
  
var hardQuestion = _context.Question  
    .Where(q => q.Difficulty == ((examDifficulty * 2) + 1)  
    && Enumerable.Range(avgTime - 3, avgTime + 3)  
    .Contains(q.Time) == true && q.CourseId == courseId)  
    .ToList().FirstOrDefault();  
list.Add(easyQuestion);  
foreach (var i in normalQuestions)  
{  
    list.Add(i);  
}  
list.Add(hardQuestion);
```

Listing 7: Generarea și salvarea întrebărilor pentru un examen

Așadar fiecare examen va avea prima întrebare cea cu dificultatea cea mai redusă, ultima cu dificultatea cea mai ridicată, iar restul vor avea o dificultate cuprinsă între cele două extreme.

Dif. exam	Dif. min	Dif.max
1	1	3
2	3	5
3	5	7
4	7	9
5	8	10

Figura 4.6: Relația dintre dificultatea examenului și a întrebărilor

În Figura 3.6 este reprezentată relația dintre dificultatea examenului și a întrebărilor. Prima coloană reprezintă dificultatea examenului, iar următoarele două reprezintă dificultatea minimă respectiv maximă a unei întrebări ce poate fi asignată unui examen. Relația pentru primele 4 trepte de dificultate între examen și întrebările de tip dificultate medie este :

$$\text{Dificultatea întrebării} = \text{Dificultatea examenului} * 2,$$

iar pentru întrebările de tip dificultate ușoară, respectiv dificilă:

$$\text{Dificultatea întrebării} = \text{Dificultatea examenului} * 2 - 1$$

$$\text{Dificultatea întrebării} = \text{Dificultatea examenului} * 2 + 1$$

După ce lista cu întrebări este generată, am creat o nouă funcție în care adaug în tabela trivială *ExamQuestion*, această listă, împreună cu id-ul examenului creat.


```

ExamQuestion examQuestion = new ExamQuestion();
examQuestion.eId = exam.eId;

for(int i = 0; i<list.Count; i++)
{
    examQuestion.qId = list[i].qId;
    _context.Add(examQuestion);
    _context.SaveChanges();
}

```

Listing 8: Adăugarea întrebărilor pentru examen în baza de date

Exceptând funcțiile de mai sus, pentru utilizatorii cu rolul de **Profesor** am mai creat funcțiile de List, Edit, Update, Create și Delete pentru entitatea *Question* și de List pentru entitatea *FeedBack*.

```

[Authorize(Roles = "Profesor")]
public IActionResult Index()
{
    var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    var query = _context.Question
        .Where(m => m.Course.ApplicationUserId == userId)
        .Select(x => new QuestionCourse
        {
            Text = x.Text,
            Answer = x.Answer,
            Difficulty = x.Difficulty,
            Time = x.Time,
            Course = x.Course.courseName,
            qId = x.qId
        });
    return View(query.ToList());
}

```

Listing 9: Funcția List pentru entitatea *Question*

Folosind javascript, am implementat o funcție(Listing 9) care printează un tabel cu situația studenților după notarea lor.

```

<script type="text/javascript">
    function Print ()
    {
        var divToPrint = document.getElementById("PrintTable");
        newWin = window.open("");
        newWin.document.write(divToPrint.outerHTML);
        newWin.print();
        newWin.close();
    }
</script>

```

Listing 10: Funcție pentru printarea unui tabel

În superclasa *DbContext* am suprascris metoda *OnModelCreating(ModelBuilder modelBuilder)* astfel încât să creez legăturile one-to-one și many-to-many între entități. Printre principalele responsabilități ale clasei *DbContext* se numără:

- **Querying** : transformă interogările LINQ în interogări SQL query și le salvează în baza de date
- **Object Materialization**: transformă datele preluate din baza de date în obiecte ale entităților
- **Persisting Data**: execută operații în baza de date, bazându-se pe starea entităților

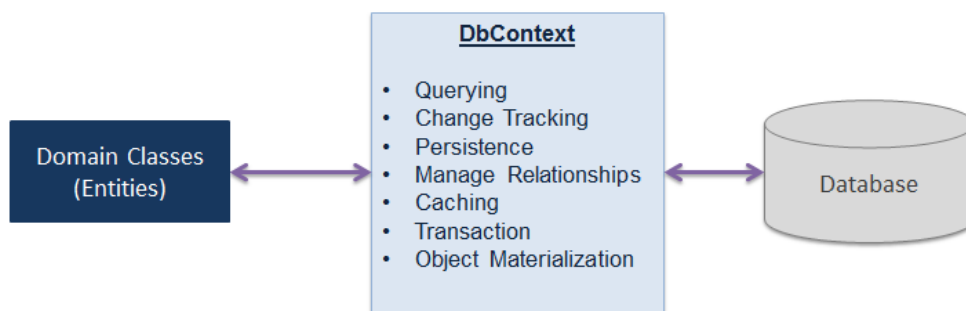


Figura 4.7: Clasa DbContext¹

¹<https://www.entityframeworktutorial.net/images/EF6/dbcontext.png>

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    //One Prof can have many courses
    modelBuilder.Entity<Course>()
        .HasOne(a => a.ApplicationUser)
        .WithMany(c => c.Courses)
        .HasForeignKey(f => f.ApplicationUserId)
        .OnDelete(DeleteBehavior.Cascade);

    // One exam can have many questions
    // One question can appear in more exams
    modelBuilder.Entity<ExamQuestion>().HasKey(eq => new {eq.eId, eq.qId});

    modelBuilder.Entity<ExamQuestion>()
        .HasOne(e => e.Exam)
        .WithMany(eq => eq.ExamQuestions)
        .HasForeignKey(eq => eq.eId)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<ExamQuestion>()
        .HasOne(q => q.Question)
        .WithMany(eq => eq.ExamQuestions)
        .HasForeignKey(eq => eq.qId)
        .OnDelete(DeleteBehavior.Restrict);
}

```

Listing 11: Suprascrierea funcției *OnModelCreating*

4.5 Funcționalitățile aplicației

După logarea și redirecționarea utilizatorilor, funcționalitățile pe care aceștia le pot accesa depinde de rolul lor. Dacă utilizatorul logat este profesor, atunci el va fi redirecționat către Dashboard din Figura 3.8.

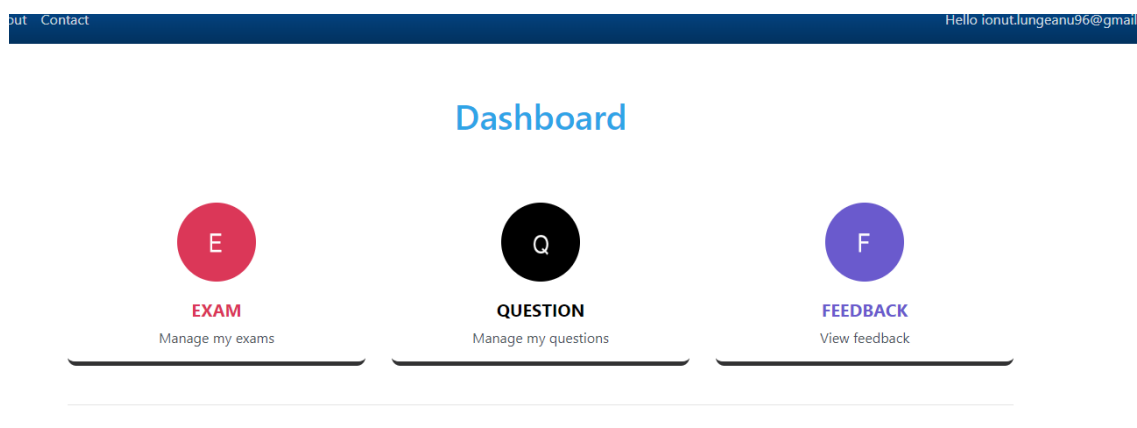


Figura 4.8: Dashboard profesor

Profesorul apoi își poate administra din panoul principal examenele, întrebările sau feedback-urile primite. Șablonul folosit pentru structurarea acestor informații (Figura 3.9) conține informațiile pentru identificarea entităților respective.

Date	Course	Difficulty	Group	Status
15.06.2019 18:07:06	Matematica	★★★★☆	6	Info
22.06.2019 13:38:10	Matematica	★★★★☆	1	Info
22.06.2019 13:49:03	Matematica	★★★★☆	1	Info

Figura 4.9: Examele create de profesorul autentificat

Butonul *Info*, din coloana denumită *Status*, afișează utilizatorului o nouă pagină conținând informațiile despre examenul selectat. Respectând GDPR², informațiile precum numele studenților nu sunt afișate; pentru fiecare student fiind identificat cu ajutorul unui id unic. Acest lucru este benefic și din perspectiva, corectării examenelor, acesta fiind total imparțial și nota este acordată fără ca profesorul să știe identitatea studentului.

În coloana *Grade* este trecută situația studentului la examenul selectat. Aceasta poate fi de 3 feluri : *In exam* (studentul rezolvă în acel moment examenul), *Grade* (studentul a terminat de rezolvat examenul și profesorul poate începe corectarea sa) sau

²General Data Protection Regulation

←
Print

Student	Group	Grade
5f3e682b-0e1c-4203-94b0-460a30716db5	1	<button>Grade</button>
dd7589a2-3dc9-40e8-ac2e-0b83a87b31d0	1	<button>In exam</button>
93ebcfcb-400c-461c-b35f-921347bcf789	1	<button>In exam</button>

Figura 4.10: Statusul examenului

poate fi afișată nota studentului la examen. Butonul de print oferă posibilitatea de a imprima acest tabel.

Pentru a crea un nou examen sau o nouă întrebare, profesorul are două posibilități: din meniul cu examenele/întrebările sale (Figura 3.9) sau din dashboard-ul principal (Figura 3.8).

Create exam

Number of questions

Between 6 and 60

Exam time

Between 15 and 150

Exam difficulty

Between 1 and 5

Group

Course

Matematica

Create

Figura 4.11: Crearea unui examen

După introducerea parametrilor din formularul de la Figura 3.11, examenul este generat, iar subiectele sale sunt afișate în Figura 3.12. Butonul *Send* din colțul din

dreapta sus trimite subiectele studenților, iar cel din colțul opus indică ora la care examenul se încheie.

The exam will end at 01:27

Send

Text	Answer	Difficulty	Time
Stiind ca $\sin a = 1/3$, sa se calculeze $\cos 2a$	7/9	★ ★ ★ ☆ ☆	3
$\sqrt{777}$	27.87	★ ★ ★ ☆ ☆	3
0!	1	★ ★ ★ ☆ ☆	2
ln	Logaritm natural	★ ★ ★ ☆ ☆	2
$\sqrt{1 + \sin(x)} - \sqrt{1 - \sin(x)}$	$2 \cdot \sin(x/2)$	★ ★ ★ ☆ ☆	6

Figura 4.12: Examenul generat

Deși în interfața examenului generat apar și câmpuri precum dificultatea sau timpul fiecărei întrebări, studentul primește un formular (Figura 3.13) în care își trece propunerile pentru rezolvarea subiectelor.

Take exam

Stiind ca $\sin a = 1/3$, sa se calculeze $\cos 2a$

1

$\sqrt{777}$

27,87

0!

1

ln

ln

$\sqrt{1 + \sin(x)} - \sqrt{1 - \sin(x)}$

2

Submit

Figura 4.13: Rezolvarea unui examen

După rezolvarea examenului și trimiterea propunerilor de rezolvare a subiectelor, studentul va avea opțiunea de a oferi feedback în legătura cu examenul.

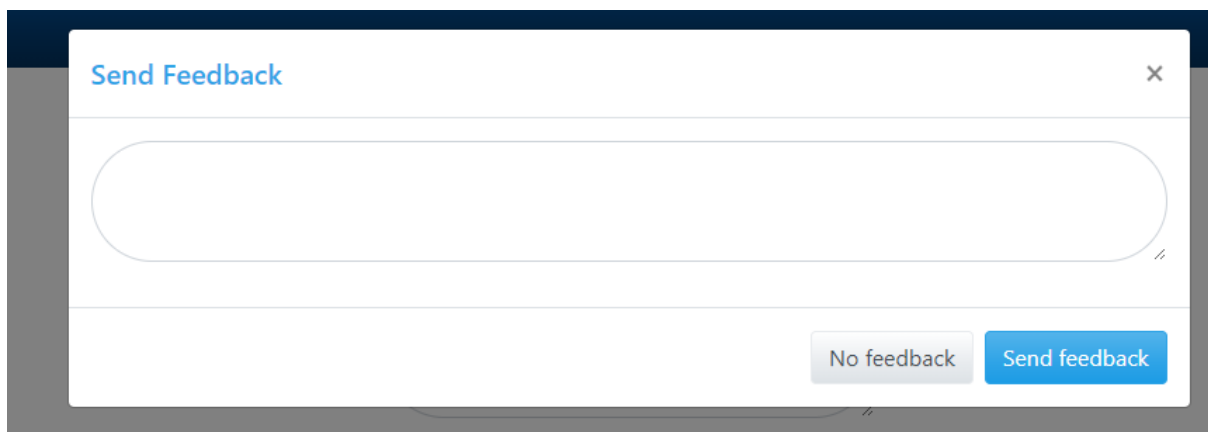


Figura 4.14: Rezolvarea unui examen

După oferirea feedback-ului, studentul este redirecționat către dashboard (Figura 3.15), unde își poate vizualiza situația examenelor sale. Aceasta poate fi *Submitted* (studentul a trimis rezolvarea, dar examenul nu este notat), *Take Exam* (studentul poate rezolva examenul) sau nota obținută.

Course	Date	
Matematica	22.06.2019 13:38:10	Take exam
Matematica	22.06.2019 13:49:03	Submitted
Matematica	26.06.2019 01:11:35	Take exam

Figura 4.15: Dashboard student

Concluzii

Luând în considerare toate premisele prezentate în această lucrare, cred că se poate concluziona faptul că implementarea unei aplicații de tipul Smart Evaluation poate îmbunătăți considerabil "fața" sistemul de evaluare universitară. Pe lângă rapiditate, precizie și economie, calități deja atribuite sistemului pe parcursul lucrării, putem plasa în încheiere, cu idea de evoluție pe care i-l poate conferi acesta, atât profesorilor cât și studenților care se pot folosi de sistem.

Cu toate că ne-am axat pe prezentarea avantajelor aplicației Smart Evaluation, pentru a ușura și mai mult fluxul de lucru, aceasta ar avea nevoie de noi funcționalități. Una din acestea este implementarea unui algoritm de inteligență artificială, care primește ca input suportul de curs și generează întrebările. Altă funcționalitate care poate îmbunătăți acest sistem este integrarea cu un sistem de supraveghere online, precum ProctorU (vezi Anexa1). Posibilitatea ca la rezolvarea examenului de a încărca ca răspuns și fișiere precum diagrame sau fișiere sursă ar fi o altă funcționalitate care ar îmbunătăți aplicația.

Bibliografie

Herbert Schildt, C#: A Beginner's Guide, 2001

Getting Started with EF Core on ASP.NET Core with an Existing Database, 2018

<https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db>

Introduction to Identity on ASP.NET Core, 2019

<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity>

What is online proctoring and why is UNE trialling this in some exams?

<https://www.une.edu.au/current-students/my-course/examinations/olx-project/faqs/faqs>

General Data Protection Regulation

<https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679>

6 Disadvantages of Traditional Paper-based Course Evaluations

<https://explorance.com/blog/6-disadvantages-of-traditional/paper-based-course-evaluations-2/>

proctorU

<https://www.proctoru.com>

The Benefits of Online Testing

<https://www.wonderlic.com/blog/benefits-of-online-testing/>

Anexa 1

Tehnologiile folosite în realizarea aplicației Smart Evaluation sunt:

- **ASP.NET Core** este o platformă folosită pentru dezvoltarea aplicațiilor web, care se pot rula pe Windows, macOS și Linux, este open-source și complet gratuită.

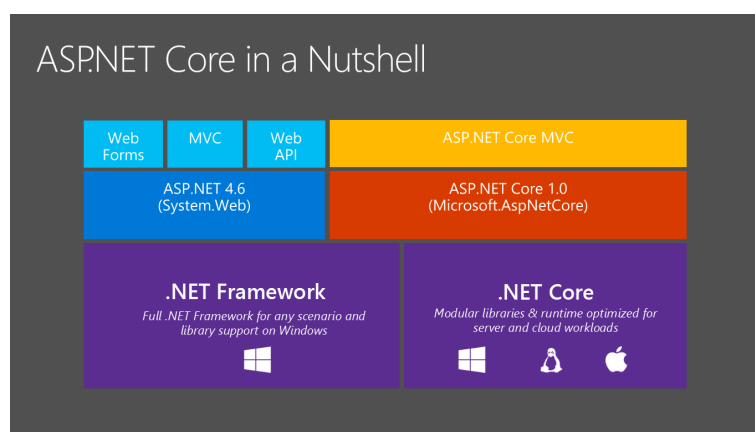


Figura 4.16: ASP.NET Core³

- **.NET** este o platformă gratis folosită pentru a dezvolta diferite tipuri de aplicații precum aplicații web, desktop, aplicații pentru smartphone sau jocuri. Limbajele folosite pentru dezvoltarea aplicațiilor enumerate sunt C#, F# sau Visual Basic.
- **C#** este un limbaj de programare orientat-obiect dezvoltat de către Microsoft la sfârșitul anilor 1990 și adoptat mai târziu ca standard de către ECMA(ECMA-334).

³https://cdn-images-1.medium.com/max/2600/1*T4H-02sKs2d-CJ0Z0WQA5g.png

Anexa 2

Un proctor este un funcționar însărcinat cu supravegherea discliplinelor în universitățile engleze de la Oxford și Cambridge. **ProctorU** este un sistem care se bazează pe proctori pentru a supraveghea examenele online. Așadar poți susține un examen de la domiciliul tău, iar un proctor să te supravegheze prin camera web. Pentru a putea beneficia de acest serviciu un student trebuie să dețină un computer, o cameră web, un microfon și o conexiune rapidă la internet. După conectarea studentului cu proctorul prin camera web, pentru a i se oferi acces la examen studentul trebuie să îndeplinească anumite condiții precum:

- deținerea unui act de identitate valabil
- plasarea unui telefon mobil sau a unei oglinzi pentru a putea observa întreaga încăpăre
- pe birou nu trebuie sa existe niciun obiect
- locația trebuie să fie luminată corespunzător
- pe parcursul examenului nu este permis ieșirea din cadrul camerei web

După ce proctorul va considera că toate condițiile de mai sus sunt îndeplinite, va verifica dispozitivul astfel încât să nu existe alte programe deschise, iar examenul va începe.

Anexa 3

ASP.NET CORE Identity

ASP.NET CORE Identity este un sistem care adaugă funcționalitatea de logare aplicațiilor ASP.NET Core. Principiile ASP.NET Core Identity sunt autorizarea și autentificarea.

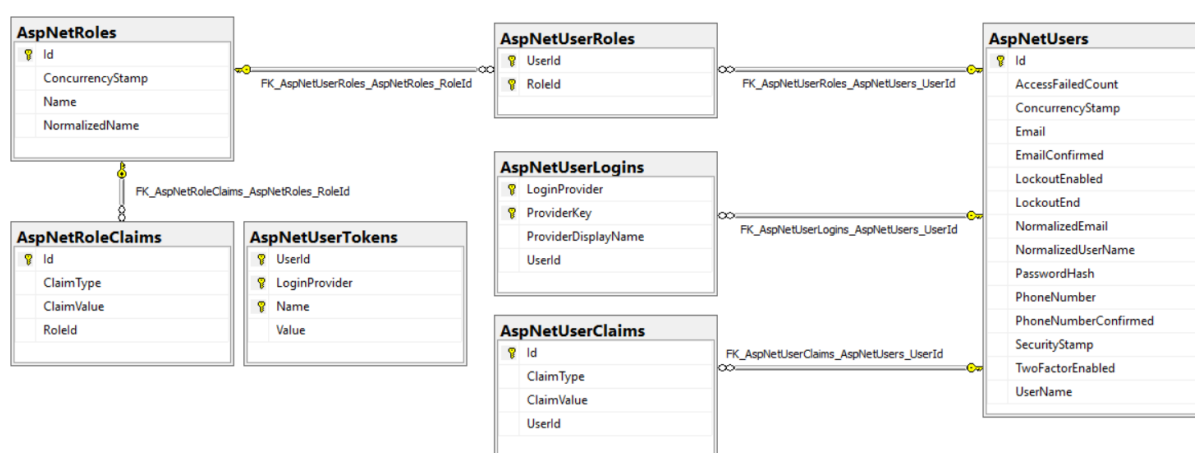


Figura 4.17: ASP.NET CORE Identity Schema⁴

Autorizarea se referă la procesul care determină acțiunile pe care un utilizator poate să le facă. În ASP.NET Core autorizarea oferă două posibilități: bazată pe roluri sau bazată pe politici. Autorizarea bazată pe roluri oferă posibilitatea de a restricționa accesul unor metode altor utilizatori. Un exemplu cum se crează rolurile și cum se utilizează se poate studia în capitolul 3.4 Detalii de implementare(vezi pag 13-14). Autorizarea bazată pe politici oferă accesul metodelor doar utilizatorilor care îndeplinesc anumite condiții. Un exemplu se poate observa în Figura 3.18.

⁴<http://logcorner.com/wp-content/uploads/2018/01/Schema.png>

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc.RazorPages;

[Authorize(Policy = "AtLeast21")]
public class AlcoholPurchaseModel : PageModel
{
}
```

Figura 4.18: ASP.NET CORE Identity Schema