

UNIVERSITATEA DE VEST DIN TIMIȘOARA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

# LUCRARE DE LICENȚĂ

*Coordonator științific:*  
Asist. Dr. Mihail Găianu

*Candidat:*  
Alina Elena Rădescu

Timișoara  
2015

UNIVERSITATEA DE VEST DIN TIMIȘOARA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
Specializarea: MATEMATICĂ INFORMATICĂ

# **PORTAL PENTRU GESTIUNEA LUCRĂRILOR DE LICENȚĂ**

*Coordonator științific:*  
Asist. Dr. Mihail Găianu

*Candidat:*  
Alina Elena Rădescu

Timișoara  
2015

# Abstract

This thesis aims at describing a software solution that solves a common problem that arises every year. Chosing to implement a software application was driven by a personal curiosity towards discovering how to build such an application. The title is a well known one: "System for the management of graduation theses". Each year, there is a process that involves both teachers and students. For the teachers, it implies proposing several thesis titles along with descriptions and bibliography suggestions and for the students it implies chosing the desired theme. After the assignment has been made, both the teacher and student establish deadlines for objectives (steps) that must be achieved for the whole goal. The purpose of this thesis it to develop a system that enables managing the data and the processes described above.

Generally speaking, a management applications aims at implementing the operational flows that make up the business processes and has the following objectives: developing the components for more or less complex systems and developing the operational flows. A manangement system is an ensemble of rules, procedures and means that enable applying the specific methods of a system in order to achieve the objectives.

The real challenge of this theme was from the start identifying the requirements of this type of business case (problem) and identifying a solution for it in order to avoid developing an incomplete or unusable solution. This required learning and discovering the "Business Analysis". The Business Analysis is the discipline that carefully studies and investigates a business case in order to identify the needs and make up a solution that meets those needs. Most of the times, just like in this situation, the solution includes developing a software system, but the solution could also include improving the business processes, organizational changes and strategical planning. Once the problems described by this business case (the management of graduation theses) were analyzed, both the needs that this application needs to meet as well as the entities that are involved in solving the problem were discovered.

Also, a very big challenge for this was chosing the technologies that help the best developing a corehent application with ease. Therefore, before starting developing, as well as during the development, it was necessary to make some decisions that would make reaching the goal easier. To build up the software solution, the following technologies were used:

- C# programming language
- .NET Framework
- ASP.NET
- ASP.NET Web Forms
- Entity Framework
- Microsoft SQL Server
- Internet Information Services (IIS)

# Introducere

În introducerea lucrării de față s-a urmărit prezentarea scopului implementării aplicației software corespunzătoare cât și importanța temei alese.

Alegerea implementării unei aplicații software se datorează în special curiozității personale de a descoperi cum se realizează o astfel de aplicație. Tema aplicației este una binecunoscută: „Portal pentru gestiunea lucrărilor de licență (System for the Management of Graduation Theses)”. Anual se petrece un proces care implică propunerea lucrărilor de licență din partea profesorilor și alegerea lor de către studenți. Ulterior acestui proces, există termene pentru atingerea unor obiective planificate, se fac raportări și evaluări. Scopul acestei teme este de a elabora un sistem care să permită gestiunea proceselor și datelor mai sus amintite prin realizarea unei aplicații software.

În general o aplicație de gestiune are ca scop proiectarea fluxului prelucrărilor în procese de producție și de asemenea are ca obiective: proiectarea de componente pentru sisteme complexe sau mai puțin complexe, proiectarea fluxului de prelucrări de date. Un sistem de gestiune este un ansamblu de reguli, proceduri și mijloace care permit aplicarea metodelor specifice unui sistem fizic pentru realizarea obiectivelor.

Adevărata provocare a subiectului ales a constituit-o încă de la început identificarea cerințelor acestui tip de business și determinarea unei soluții pentru această problemă pentru a evita dezvoltarea unei soluții incomplete sau inutilizabile. Astfel noțiuni elementare despre ceea ce se numește „Business Analysis” au fost necesare.

Analiza de business (Business Analysis) este o disciplină de studiere atentă și investigare pentru identificarea nevoilor și pentru determinarea unei soluții care să acopere aceste nevoi. De cele mai multe ori, ca și în cazul de față, soluția include dezvoltarea unui sistem software, dar soluția poate să includă îmbunătățirea proceselor, schimbări organizaționale și planificare strategică. Odată cu analizarea problemei expuse de această temă au reușit atât nevoile pe care trebuie să le rezolve această aplicație software cât și entitățile implicate în soluționarea problemei.

De asemenea o provocare semnificativă a reprezentat-o alegerea tehnologiilor care să ofere suport pentru dezvoltarea cât mai coerentă și care să faciliteze dezvoltarea software a aplicației. Astfel, atât înainte de începerea dezvoltării, cât și pe parcursul ei, a fost nevoie de luarea unor decizii tehnice care să nu îngreuneze atingerea scopului acestei lucrări. Pentru a facilita atingerea scopului acestei lucrări au fost folosite:

- Limbajul de programare C#
- .NET Framework
- ASP.NET
- ASP.NET Web Forms
- Entity Framework
- Microsoft SQL Server
- Internet Information Services (IIS)



Aplicația software realizată este o aplicație web dezvoltată în limbajul de programare C#. Una dintre cele mai importante tehnologii folosite este „.NET Web Forms”, dar la fel de important este și „Entity Framework” care ușurează interacțiunea cu baza de date. Serverul de

baze de date folosit este „Microsoft SQL Server”. O scurtă descriere despre fiecare tehnologie folosită se află la secțiunea dedicată aplicației software.

Lucrarea de față care include în structura sa 4 capitole și debutează în mod firesc cu descrierea entităților implicate în procesele implementate. Lucrarea se continuă cu descrierea fiecărui proces implementat, pentru ca la finalul acestei lucrări, să fie descrisă aplicația software implementată. De asemenea pe parcursul acestei lucrări există un capitol dedicat tehnologiilor folosite, „Capitolul 3. Tehnologii. Cadrul tehnic”.

# Capitolul 1

## Descrierea entităților implicate în procesele implementate

### Definiția noțiunii de entitate



În „Dicționarul Explicativ al Limbii Române” definițiile pentru entitate sunt:

- „Totalitatea caracteristicilor unui obiect reprezentate sub formă de informații. Existentă sau realitate determinată.”
- „Conținut de sine stătător, existență determinată ca întindere, importanță, valoare”
- „Ceea ce constituie esența, existența unui lucru.”

Așa cum dicționarul descrie o entitate este bine determinată. Entitățile definite în continuare ajută la definirea bazei de date, dar bineînțeles acestea sunt piese importante în definirea proceselor.

### Entitățile utilizatori și roluri

Pentru ca o aplicație software să existe este nevoie să definim utilizatorii acestei aplicații. De asemenea în cazul de față este nevoie să definim utilizatorii aplicației aceasta fiind o aplicație de gestiune. În cazul de față există trei tipuri importante de utilizatori: studenți, profesori, administratori. Odată cu definirea acestor trei tipuri reiese nevoia definirii entității roluri. De asemenea este necesară o nouă entitate care să definească drepturile pentru fiecare rol. Astfel vom avea:

-  Entitatea Rol:
  - numele rolului (de exemplu rolul de „student”)
  - descrierea dreptului
  - drepturi atașate fiecărui rol (de exemplu dreptul de „vizualizare lucrări de licență”)
-  Entitatea Utilizator:
  - grad
  - prenume utilizator
  - nume utilizator
  - utilizator (utilizatorul / username folosit pentru a se autentifica în aplicație)
  - parola (folosită pentru autentificarea în aplicație)
  - adresa de E-mail
  - roluri utilizator (de exemplu rolul de „student”)

Se poate observa că există o legătură între cele două entități definite mai sus. Rolurile unui utilizator sunt legate de acel utilizator printr-o legătură „many-to-many” (mai mulți utilizatori pot avea mai multe roluri).

În aplicația software vor exista trei roluri:


1. Rolul de Administrator
2. Rolul de Profesor
3. Rolul de Student



Aceste roluri vor fi descrise în detaliu în capitolul dedicat aplicației software.

## Entitatea lucrare de licență

Entitatea principală, după cum este menționat și în numele temei, este „lucrarea de licență”. Managementul proceselor incluse în acest portal de gestiune au ca entitate principală „lucrarea de licență”. Informațiile necesare pentru descrierea unei lucrări de licență sunt descrise în continuare.

-  Entitatea Lucrare de Licență:

- nume
- descriere
- profesor coordonator
- bibliografie

## Legături între entități

Entitățile principale implicate în procesele modelate de această aplicație au fost descrise în secțiunile anterioare. Odată cu modelarea proceselor, descrisă în capitolul următor, sunt definite și o serie de legături între aceste entități. Astfel, după cum se poate observa și în descrierea anterioară a entităților, rolurile unui utilizator sunt legate de acel utilizator printr-o legătură „many-to-many” (mai mulți utilizatori pot avea mai multe roluri).

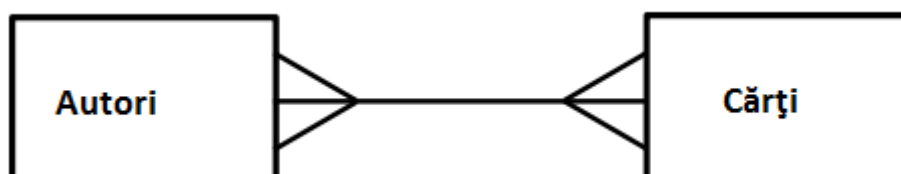


Figure 1: Relație many-to-many

În analiza sistemelor, o relație many-to-many este un tip de cardinalitate care se referă la o relație dintre două entități A și B, unde A poate conține o înregistrare părinte pentru care există mai multe înregistrări copii în B, și vice versa.

De exemplu, pentru entitatea A ne putem gândi ca înregistrările sunt autori, iar pentru entitatea B înregistrările sunt cărțile. Un autor poate să scrie mai multe cărți și o carte poate fi scrisă de mai mulți autori. Figura anterioară exemplifică acest caz.

De asemenea există o legătură și între o lucrare de licență și profesorul coordonator. Astfel reiese o legătură între utilizatorii cu rolul de „profesor” și lucrările de licență. Această legătură între entități este numită „one-to-many”. Astfel un profesor (în cazul aplicației, un utilizator cu rolul de „profesor”) este profesor coordonator pentru mai multe lucrări de licență.

În analiza sistemelor, o relație one-to-many este un tip de cardinalitate care se referă la o relație dintre două entități A și B, în care un element al entității A poate să fie legat de mai mult elemente ale entității B, dar un membru din entitatea B este legat de un singur element din A. Ca și exemplu din viața de zi cu zi ne putem gândii la A ca fiind mame, iar la B ca fiind copiii. O mamă poate avea mai mulți copii, dar un copil poate avea o singură mamă.

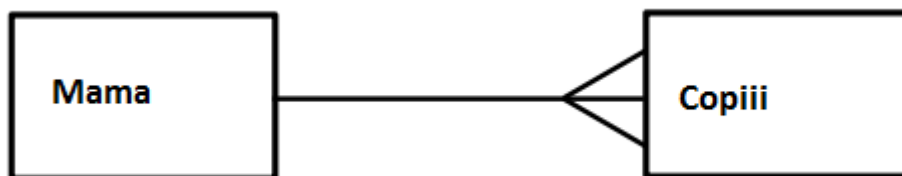


Figure 2: Relație one-to-many

De asemenea o temă a unei lucrări de licență poate să fie asociată unuia sau mai multor studenți. Această asociere se realizează prin procesul de selecție a unei lucrări de licență. O astfel de legătură, ca aceasta dintre studenți și lucrările de licență, poartă numele de legătură „many-to-many”. Mai multe entități de un anumit tip (în cazul de față studenții), sunt legate de mai multe entități de alt tip (în cazul de față lucrările de licență).

Din această structură reiese și structura tabelelor din baza de date, descrisă într-o secțiune următoare din „Capitolul 4. Descrierea aplicației software”.

Mai sus am folosit un termen teoretic, și anume: „cardinalitatea”.

În proiectarea bazelor de date, cardinalitatea, sau principiul fundamental al unui tabel de date cu privire la un alt tabel este un aspect critic. Relația dintre cele două tabele trebuie să fie una precisă și exactă, care să explice cum relaționează fiecare tabelă cu celelalte tabele. În modelele relaționate tabelele pot fi relaționate fie printr-o relație de „one-to-many”, fie printr-o relație de „many-to-many”. Aceasta se numește „cardinalitate” unei tabele date în relație cu o altă tabelă. În capitolul dedicat proiectării bazei de date corespunzătoare aplicației software există o secțiune unde sunt tratate toate aspectele teoretice legate de bazele de date.



## Capitolul 2

### Modelarea și descrierea proceselor

#### Definiții

Modelarea proceselor ajută la înțelegerea implicării mai multor roluri și/sau departamente în executarea activităților din interiorul unei organizații. În acest caz este nevoie de înțelegerea activităților ce vor fi executate de studenți, profesori și administratori în cadrul procesului de alocare a lucrărilor de licență.

#### Un proces:

- descrie cum mai mulți oameni sau grupuri pot colabora pentru a executa o anumită acțiune sau o anumită muncă,
- implică un anumit număr de activități care sunt legate între ele printr-un flux bine definit,
- este repetabil și poate avea mai multe cai care ajung la finalul procesului,
- este inițiat de un eveniment care are loc.

Modelarea proceselor conține o serie de elemente cheie:

- **Activitățile:** Pașii individuali care trebuie urmați pentru ca o parte dintr-o activitate să poată să fie executată. O activitate poate să fie constituită dintr-o singură etapă sau din mai multe etape.
- **Intrările și rezultatul:** Intr-un proces este nevoie atât de informații și date de intrare în procesare, cât și de rezultatul procesării. Altfel procesul nu își are rostul.
- **Decizii:** Pe parcursul unui proces este foarte posibil să fie nevoie de luarea anumitor decizii. În unele cazuri aceste decizii se pot lua automat, în altele sunt luate prin interacțiunea umană. Astfel un proces se împarte în două, sau mai multe, subprocese / activități diferite în funcție de decizia luată. Prin urmare o decizie poate crea fluxuri mutual exclusive sau fluxuri paralele.
- **Evenimente:** Evenimentele apar din exteriorul procesului și sunt rezultatul unei acțiuni, un mesaj primit, sau trecerea unei anumite perioade de timp. Evenimente pot duce la întreruperea sau terminarea unui proces.
- **Fluxurile:** Indică direcția, prin secvențe pas cu pas, a întregului flux de lucru.
- **Rolurile:** După cum se poate observa rolurile descrise în una din secțiunile anterioare reprezintă un tip de persoane sau un tip de grup de persoane. Definirea rolurilor se face ținând cont de modelul organizațional.
- **Asocierea rolurilor cu acțiunile:** Definiște ce activitate poate fi executată de un anumit rol.
- **Punctele terminale:** reprezintă punctul de start și punctul de final ale unui proces.

## Descrierea proceselor

În aplicație există posibilitatea executării unui lanț de acțiuni care determină un proces business. Câteva astfel de procese sunt descrise în această secțiune urmând ca modul lor de implementare folosind aplicația să fie descris în capitolul dedicat acestuia (ultimul capitol al acestei lucrări).

Procesele de administrare a datelor apar atunci când este necesar ca datele stocate să fie actualizate. Există mai multe astfel de situații, de exemplu:

- Schimbarea anului universitar determină schimbarea studenților, prin urmare și actualizarea datelor despre studenți. Această acțiune se realizează în momentul în care datele despre fiecare student care trebuie introdus există. Acțiunea poate fi realizată de operatorul aplicație care are rolul de a actualiza datele.
- Schimbarea unui profesor aduce după sine mai multe operații care trebuie realizate:
  - marcarea profesorului ca fiind inactiv
  - inserarea unui noi profesor și a datelor corespunzătoare
  - inserarea lucrărilor de licență ale profesorului respectiv, valabile pentru anul universitar următor.
- Actualizarea datelor despre lucrările de licență odata cu schimbarea anului universitar.
- Administrarea operatorilor și a rolurilor acestora. Un astfel de proces de administrare poate apărea ca necesitate, de exemplu, în momentul în care se dorește ca un operator să nu mai aibă aceleași drepturi pe care le avea până în momentul respectiv.

În urma proceselor de administrare vor rezulta mai mulți utilizatori ai aplicației. Cei mai importanți sunt utilizatorii cu rol de student, și utilizatorii cu rol de profesor.

Procese mai complexe care folosesc întregul arsenal de date stocate se întâlnesc în special în ceea ce înseamnă gestionarea lucrărilor de licență. Nu este suficient să stocăm datele. Aceste date trebuie folosite în atingerea scopului aceste lucrări. Aplicația gestionează o serie de astfel de procese.

**Asignarea unei lucrări de licență** presupune o serie de pași care trebuie luați în considerare:

1. Studentul alege o lucrare de licență care aparține unui anumit profesor. Acest eveniment duce la un altul care presupune o decizie.
2. Profesorul trebuie să decidă dacă lucrarea va fi atribuită acelui student.
3. În caz afirmativ:
  - (a) Studentul primește confirmarea
  - (b) Studentul împreună cu profesorul stabilesc etapele elaborării lucrării de licență
  - (c) Pentru fiecare etapă este atribuită o dată până la care aceea etapă trebuie îndeplinită
4. În caz negativ studentul va trebui să își aleagă o altă lucrare de licență, dar poate să țină cont de propunerea profesorului (dacă acesta a făcut o astfel de propunere).

Acest proces este descris și vizual în următoarea imagine.

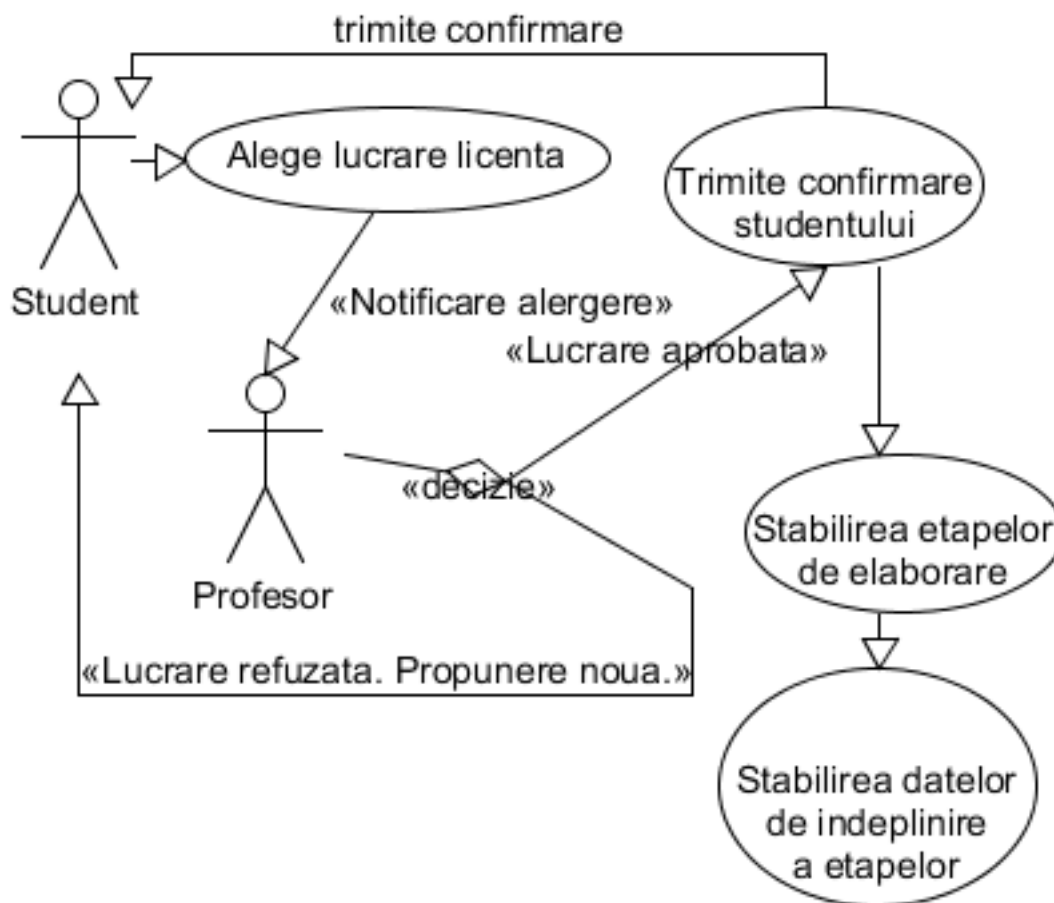


Figure 3: Procesul de alegere a unei lucrări de licență

**Procesul de elaborare a unei lucrări de licență** presupune realizarea etapelor stabilite la finalul procesului de alocare a unei lucrări de licență. Se poate spune că acest proces există în procesul anterior, fiind o necesitate pentru continuarea procesului anterior. Acest proces are nevoie de informații care sunt furnizate de procesul anterior. Nu poate fi realizat fără a avea acele informații și anume etapele de elaborare stabilite. Pentru ușurința utilizării aplicației au fost definite câteva etape standard de elaborare a unei lucrări de licență, dar acestea pot fi modificate de către profesor. Pentru a generaliza vom presupune că există 3 etape de elaborare a unei lucrări de licență numite generic: Etapa 1, Etapa2, Etapa3.

Procesul de elaborare constă în:

1. Elaborarea etapei 1 de către student
2. Evaluarea etapei 1 de către profesor (verificarea satisfacerii termenului limită, comentarii și suport).
3. Îmbunătățirea etapei 1 de către student, dacă este cazul
4. Elaborarea etapei 2 de către student
5. Evaluarea etapei 2 de către profesor (verificarea satisfacerii termenului limită, comentarii și suport)

6. Îmbunătățirea etapei 2 de către student, dacă este cazul
7. Elaborarea etapei 3 de către student
8. Evaluarea etapei 3 de către profesor (verificarea satisfacerii termenului limită, comentarii și suport)
9. Îmbunătățirea etapei 3 de către student, dacă este cazul
10. Încheierea lucrării de licență

Următoarea imagine exemplifică acțiunile pentru o singură etapă.

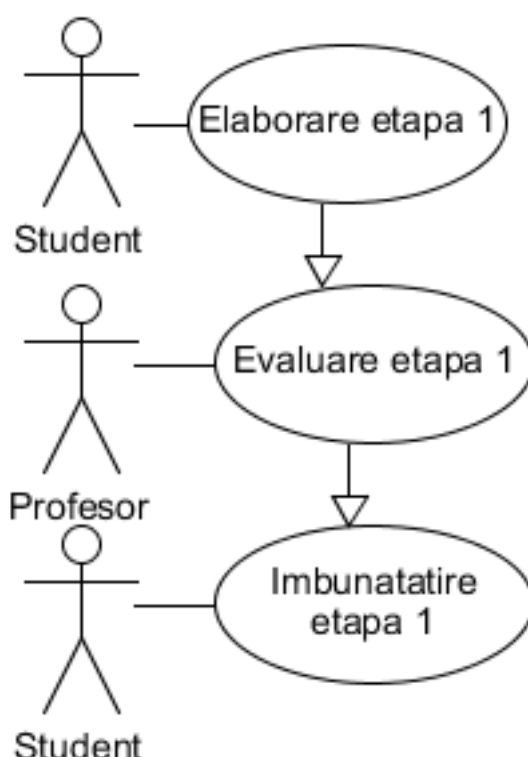


Figure 4: Elaborarea etapelor

În aplicația software am definit aceste etape ca obiective de atins pe durata elaborării lucrării de licență. Obiectivele sunt stabilite de profesorul coordonator imediat după ce o lucrare de licență a fost aleasă de un student, pentru a evita nesatisfacerea termenelor limită. Deoarece în mod normal o lucrare de licență se alege în anul anterior elaborării ei, stabilirea etapelor de elaborare se poate realiza separat de procesul de alegere a unei lucrări de licență, astfel pot rezulta încă doua procese separate:

- Stabilirea etapelor de elaborare. Acest proces este constituit dintr-o singură activitate care va fi realizată de profesor.
- Stabilirea datelor de indeplinire a etapelor de elaborare a unei lucrări de licență. De asemenea și acest proces este compus dintr-o singură activitate care se realizează prin prezența a doi actori: profesorul și studentul.

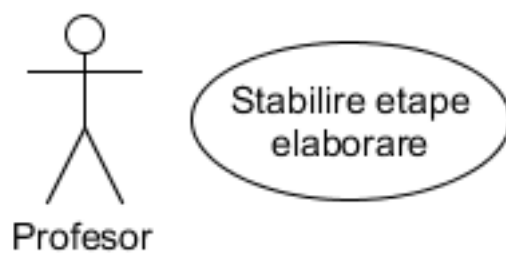


Figure 5: Stabilirea etapelor de elaborare

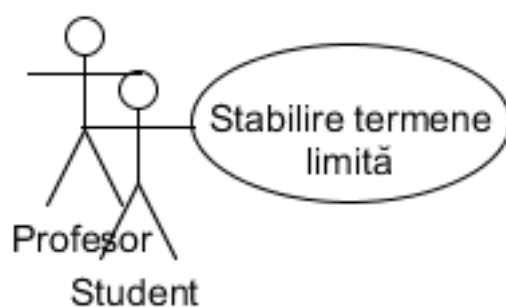


Figure 6: Stabilirea termenelor limită

Lista proceselor poate continua și acestea pot fi împărțite în subproces. Odată cu prezentarea și descrierea aplicației software aceste procese sunt ușor de intuit, astfel acest capitol se limitează la prezentarea proceselor principale.

## Capitolul 3

### Tehnologii. Cadrul tehnic

#### Evoluția tehnicilor de programare

În istoria tehnicilor de programare au existat câteva tehnici care au influențat considerabil ceea ce avea să devină programarea orientată pe obiect. O importanță deosebită au avut următoarele tehnici de programare:

- **Programarea nestructurată:** constă în elaborarea unui program simplu care utilizează numai variabile globale. Complicațiile apar atunci când prelucrarea devine mai amplă, iar datele se multiplică și se diversifică. Există mai multe limbaje de programare în care se utilizează programarea nestructurată. De exemplu: FORTRAN, COBOL, BASIC.
- **Programarea procedurală:** constă în elaborarea unui program principal deservit de subprograme cu parametri formali, variabile locale și apeluri cu parametri efectivi. Se obțin avantaje privind depanarea și reutilizarea codului și se aplică noi tehnici privind transferul parametrilor și vizibilitatea variabilelor. Complicațiile apar atunci când la program sunt asigurați doi sau mai mulți programatori care nu pot lucra simultan pe un același fișier ce conține codul sursă.
- **Programarea modulară:** constă în gruparea subprogramelor cu funcționalități similare în module, implementate și depanate separat.

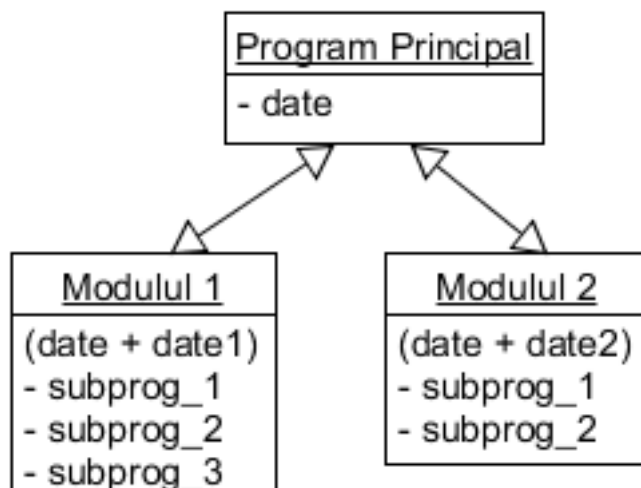


Figure 7: Programarea modulară

Se obțin avantaje privind independența și încapsularea (prin separarea zonei de implementare, păstrând vizibilitatea numai asupra zonei de interfață a modulului) și se aplică tehnici de asociere a procedurilor cu datele pe care le manevrează, stabilind și diferite reguli de acces la date și la subprograme. Modulele sunt „centrate” pe proceduri, acestea gestionând și setul de date pe care le prelucrează (date + date1) din figura următoare.

Dacă, de exemplu, dorim să avem mai multe seturi diferite de date, toate înzestrate comportamental cu procedurile din modulul module1, această arhitectură de aplicație nu este avantajoasă.

- **Programarea orientată obiect:** constă în elaborarea de programe cu noi „tipuri” ce integrează atât datele, cât și metodele asociate creării, prelucrării și distrugerii acestor date. Se obțin avantaje prin abstractizarea programării (programul nu mai este o succesiune de prelucrări, ci un ansamblu de obiecte care prind viață, au diverse proprietăți, sunt capabile de acțiuni specifice și care interacționează în cadrul programului. Intervin tehnici noi privind instanțierea, derivarea și polimorfismul tipurilor obiectuale.

## Programarea orientată obiect

Un **tip de date abstract** este o entitate caracterizată printr-o structură de date și un ansamblu de operații aplicabile acestor date. Considerând, în rezolvarea unei probleme de gestiune a accesului utilizatorilor la un anumit site, tipul abstract USER, vom observa că sunt multe date ce caracterizează un utilizator Internet. Totuși se va ține cont doar de datele semnificative pentru problema dată. Astfel, „culoarea ochilor” este irelevantă în acest caz, în timp ce „data nașterii” poate fi importantă. În aceeași idee, operații specifice ca „se înregistrează”, „comandă on-line” pot fi relevante, în timp ce operația „mănâncă” nu este, în cazul nostru. Evident, nici nu se pun în discuție date sau operații nespecifice („numărul de laturi” sau acțiunea „zboară”).

Operațiile care sunt accesibile din afara entității formează interfața acesteia. Astfel, operații interne cum ar fi conversia datei de naștere la un număr standard calculat de la 01.01.1900 nu fac parte din interfața tipului de date abstract, în timp ce operația „plasează o comandă on-line” face parte, deoarece permite interacțiunea cu alte obiecte (SITE, STOC etc.)

O **instanță** a unui tip de date abstract este o „concretizare” a tipului respectiv, formată din valori efective ale datelor.

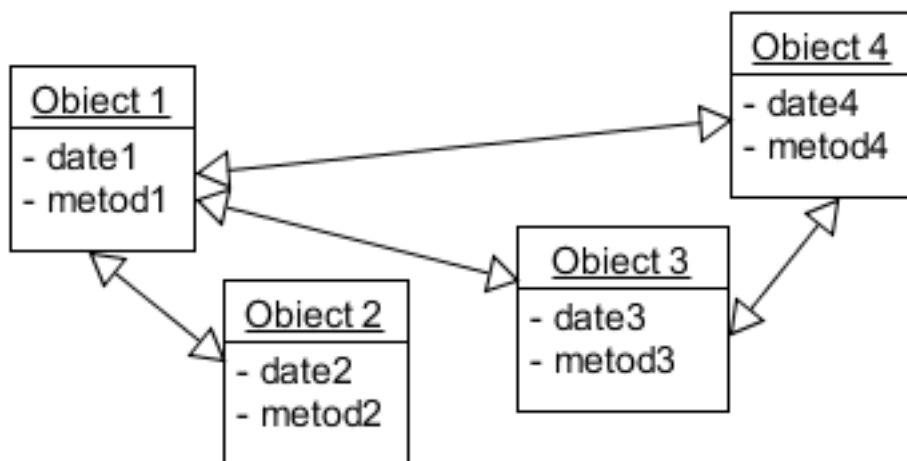


Figure 8: Programarea orientată obiect

Un **tip de date obiectual** este un tip de date care implementează un tip de date abstract. Vom numi operațiile implementate în cadrul tipului de date abstract metode. Spunem că datele și metodele sunt membrii unui tip de date obiectual. Folosirea unui astfel de tip

presupune: existența definiției acestuia, apelul metodelor și accesul la date. Un exemplu de-acum clasic de tip de date abstract este STIVA. Ea poate avea ca date: numerele naturale din stivă, capacitatea stivei, vârful etc. Iar operațiile specifice pot fi: introducerea în stivă (push) și extragerea din stivă (pop). La implementarea tipului STIVA, vom defini o structura de date care să rețină valorile memorate în stivă și câmpuri de date simple pentru: capacitate, număr de elemente etc. Vom mai defini metode (subprograme) capabile să creeze o stivă vidă, care să introducă o valoare în stivă, să extragă valoarea din vârful stivei, să testeze dacă stiva este vidă sau dacă stiva este plină.

Crearea unei instanțe noi a unui tip obiectual, presupune operații specifice de „construire” a noului obiect, metoda corespunzătoare purtând numele de constructor. Analog, la desființarea unei instanțe și eliberarea spațiului de memorie aferent datelor sale, se aplică o metodă specifică numită destructor. O aplicație ce utilizează tipul obiectual STIVA, va putea construi două sau mai multe stive (de cărți de joc, de exemplu), le va umple cu valori distincte, va muta valori dintr-o stivă în alta după o anumită regulă desființând orice stivă golită, până ce rămâne o singură stivă. De observat că toate aceste prelucrări recurg la datele, constructorul, destructorul și la metodele din interfața tipului STIVA descris mai sus. Principalul tip obiectual întâlnit în majoritatea mediilor de dezvoltare (Visual Basic, Delphi, C++, Java, C#) poartă numele de clasă (class). Există și alte tipuri obiectuale (struct, object). O instanță a unui tip obiectual poartă numele de obiect.

La implementare, datele și metodele asociate trebuie să fie complet și corect definite, astfel încât utilizatorul să nu fie nevoit să țină cont de detalii ale acestei implementări. El va accesa datele, prin intermediul proprietăților și va efectua operațiile, prin intermediul metodelor puse la dispoziție de tipul obiectual definit. Spunem că tipurile de date obiectuale respectă principiul încapsulării. Astfel, programatorul ce utilizează un tip obiectual CONT (în bancă) nu trebuie să poarte grija modului cum sunt reprezentate în memorie datele referitoare la un cont sau a algoritmului prin care se realizează actualizarea soldului conform operațiilor de depunere, extragere și aplicare a dobânzilor. El va utiliza unul sau mai multe conturi (instanțe ale tipului CONT), accesând proprietățile și metodele din interfață, realizatorul tipului obiectual asumându-și acele griji în momentul definirii tipului CONT.

Permițând extensia tipurilor de date abstracte, clasele pot avea la implementare:

- date și metode caracteristice fiecărui obiect din clasă (membri de tip instanță),
- date și metode specifice clasei (membri de tip clasă). Astfel, clasa STIVA poate beneficia, în plus, și de date ale clasei cum ar fi: numărul de stive generate, numărul maxim sau numărul minim de componente ale stivelor existente etc. Modificatorul static plasat la definirea unui membru al clasei face ca acela să fie un membru de clasă, nu unul de tip instanță. Dacă în cazul membrilor nestatici, există câte un exemplar al membrului respectiv pentru fiecare instanță a clasei, membrii statici sunt unici, fiind accesați în comun de toate instanțele clasei. Mai mult, membrii statici pot fi referiți fără a crea vreo instanță a clasei respective.

Există o serie de tehnici care se aplică cu ușurință în programarea orientată pe obiect și care sunt folosite și la elaborarea aplicației corespunzătoare:

- **Supraîncărcarea:** Deși nu este o tehnică specifică programării orientată obiect, ea creează un anumit context pentru metodele ce formează o clasă și modul în care acestea pot fi (ca orice subprogram) apelate. Prin supraîncărcare se înțelege posibilitatea de a defini în același domeniu de vizibilitate mai multe funcții cu același nume, dar cu parametri diferiți ca tip și/sau ca număr. Astfel ansamblul format din numele funcției și lista sa de parametri reprezintă o modalitate unică de identificare numită „semnătură” sau „amprentă”. Supraîncărcarea permite obținerea unor efecte diferite ale apelului în contexte



diferite. Datorită tehnicii de supraîncărcare C++, Java și C# permit existența mai multor constructori. Apelul unei funcții care beneficiază, prin supraîncărcare, de două sau mai multe semnături se realizează prin selecția funcției a cărei semnătură se potrivește cel mai bine cu lista de parametri efectivi (de la apel). Astfel, poate fi definită metoda „`comandă_on-line`” cu trei semnături diferite:

- `comandă_online(cod_prod)` cu un parametru întreg (deosemnând comanda unui singur produs identificat prin `cod_prod`).
  - `comandă_online(cod_prod,cantitate)` cu primul parametru întreg și celalalt real
  - `comandă_online(cod_prod,calitate)` cu primul parametru întreg și al doilea boolean
- 
- **Moștenirea:** Pentru tipurile de date obiectuale este posibilă o operație de extindere sau specializare a comportamentului unei clase existente prin definirea unei clase noi ce moștenește datele și metodele clasei de bază, cu această ocazie putând fi redefiniți unii membri existenți sau adăugați unii membri noi. Operația mai poartă numele de **derivare**. Clasa din care se moștenește se mai numește **clasă de bază** sau **superclasă**. Clasa care moștenește se numește **subclasă**, **clasă derivată** sau **clasă descendentă**. Ca și în Java, în C# o subclasă poate moșteni de la o singură superclasă, adică avem de-a face cu moștenire simplă; aceeași superclasă însă poate fi derivată în mai multe subclase distincte. O subclasă, la randul ei, poate fi superclasă pentru o altă clasă derivată. O clasă de bază împreună cu toate clasele descendente (direct sau indirect) formează o ierarhie de clase. În C#, toate clasele moștenesc de la clasa de bază `Object`.
  - **Polimorfism. Metode virtuale:** Folosind o extensie a sensului etimologic, un obiect polimorfic este cel capabil să ia diferite forme, să se afle în diferite stări, să aibă comportamente diferite. Polimorfismul obiectual se manifestă în lucrul cu obiecte din clase aparținând unei ierarhii de clase, unde, prin redefinirea unor date sau metode, se obțin membri diferiți având însă același nume. Astfel, în cazul unei referiri obiectuale, se pune problema stabilirii datei sau metodei referite. Comportamentul polimorfic este un element de flexibilitate care permite stabilirea contextuală, în mod dinamic, a membrului referit. De exemplu, dacă este definită clasa numită `Piesa` (de șah), cu metoda nestatică `muta(pozitie_initiala,pozitie_finala)`, atunci subclasele `Turn` și `Pion` trebuie să aibă metoda `muta` definită în mod diferit (pentru a implementa maniera specifică a pionului de a captura o piesă). Atunci, pentru un obiect `T`, aparținând claselor derivate din `Piesa`, referirea la metoda `muta` pare nedefinită. Totuși mecanismele programării orientate obiect permit stabilirea, în momentul apelului, a clasei proxime căreia îi aparține obiectul `T` și apelarea metodei corespunzătoare (mutare de pion sau tură sau altă piesă). Pentru a permite acest mecanism, metodele care necesită o decizie contextuală (în momentul apelului), se decalră ca metode virtuale (cu modifierul `virtual`). În mod curent, în C# modifierului `virtual` al funcției din clasa de bază, îi corespunde un specificator `override` al funcției din clasa derivată ce redefineste funcția din clasa de bază. O metodă ne-virtuală nu este polimorfică și, indiferent de clasa căreia îi aparține obiectul, va fi invocată metoda din clasa de bază.

## Limbajul de programare C#



Lansat publicului în luna Iunie a anului 2000 și oficial în primăvara anului 2002, C# este un limbaj de programare care combină facilități testate de-a lungul timpului cu inovații de ultim moment. Creatorii acestui limbaj au fost o echipa de la firma Microsoft, condusă de Anders Hejlsberg. Deși limbajul este creat de Microsoft, acesta nu este destinat doar platformelor Microsoft. Compilatoare C# există și pentru alte sisteme precum Linux sau Macintosh. Creat ca instrument de dezvoltare pentru arhitectura .NET, limbajul ofera o modalitate facilă și eficientă de a scrie programe pentru sistemul Windows, internet, componente software, și lista poate continua. C# deriva din două dintre cele mai de succes limbaje de programare: C și C++. De asemenea, limbajul este o “ruda” apropiată a limbajului de programare Java. Pentru o mai buna intelegere a limbajului C# este interesant de remarcat care este natura relațiilor acestuia cu celelalte trei limbaje mentionate mai sus. Pentru aceasta, vom plasa mai întâi limbajul C# în contextul istoric determinat de cele trei limbaje.

Limbajul C a fost inventat de către Dennis Ritchie în anii '70 pe un calculator pe care rula sistemul de operare UNIX. Limbajul C s-a dezvoltat în urma revoluției programării structurate din anii '60. Înainte de programarea structurată, programele erau greu de scris și de înțeles din cauza logicii. O masă încălțită de salturi, apeluri și reveniri, greu de urmărit era cunoscută sub numele de cod spaghetti. Datorită sintaxei sale concise și ușor de utilizat, în anii '80, limbajul C a devenit cel mai răspândit limbaj structurat.

La sfârșitul anilor '70, a apărut o modalitate nouă de programare și anume programarea orientată obiect. Astfel limbajul C a fost extins.

Limbajul C++ a fost creat de către Bjarne Stroustrup începând din 1979, la laboratoarele Bell din Murray Hill, New Jersey. Limbajul a fost denumit inițial C cu clase, iar în 1983 numele acestuia a fost modificat în C++. În esență, C++ reprezintă versiunea orientată obiect a limbajului C. În anii '80, limbajul C++ a suferit dezvoltări și perfecționări masive, astfel ca în anii '90 a devenit cel mai răspândit limbaj de programare.

În 1991, firma Sun Microsystems a demarat lucrul la limbajul de programare Java. Acest limbaj este un limbaj structurat și orientat obiect, cu o sintaxă și filozofie derivate din C++. Aspectele novatoare se referă mai mult la modificările mediului de programare. Aspectul esențial în Java este posibilitatea de a crea cod portabil pe platforme diferite. Înainte de explozia Internetului, majoritatea programelor erau compilate și destinate utilizării pe un anumit procesor și sub un anumit sistem de operare. După apariția Internetului însă, la care sunt conectate sisteme cu procesoare și sisteme de operare diferite, problema portabilității a devenit foarte importantă. Java a realizat portabilitatea prin transformarea codului sursă al programului într-un cod intermediar numit „bytecode”. Acest format intermediar este executat apoi de așa numita Mașina Virtuală Java (MVJ). Asadar, programele Java pot rula în orice mediu în care exista o astfel de mașină virtuală.

Ca parte a ansamblului strategiei .NET, dezvoltată de compania Microsoft, la finele anilor '90 a fost creat limbajul C#.

C# este direct înrudit cu C, C++ și Java. „Bunicul” limbajului C# este C-ul. De la C, C# moștenește sintaxa, multe din cuvintele cheie și operatorii. De asemenea, C# construiește peste modelul de obiecte definit în C++. Relația dintre C# și Java este mai complicată. Java derivă la rândul său din C și C++. Ca și Java, C# a fost proiectat pentru a produce cod portabil. Limbajul C# nu derivă din Java. Între C# și Java există o relație similară celei dintre “veri”, ele derivă din același stramoș, dar deosebindu-se prin multe caracteristici importante. Este foarte asemănător cu limbajul de programare Java și C++, fiind principalul limbaj de programare din Microsoft .NET framework. Diverse surse susțin că C# a fost inventat deoarece compania

Sun Microsystems nu a îngăduit companiei Microsoft să aducă modificări asupra limbajului de programare Java.

C# are o legatură deosebită cu mediul sau de rulare, arhitectura .NET. Pe de o parte, C# a fost dezvoltat pentru crearea codului pentru arhitectura .NET, iar pe de altă parte bibliotecile utilizate de C# sunt cele ale arhitecturii .NET.

C# este un limbaj de programare orientat obiect elegant care oferă posibilitatea dezvoltatorilor de software să creeze o gamă variată de aplicații care rulează pe platforma .NET. C# poate fi utilizat pentru crearea aplicațiilor „client de Windows”, serviciilor web, aplicațiilor client-server, aplicațiilor cu baze de date și multe altele.

Programele dezvoltate folosind limbajul de programare C# rulează pe platforma .NET Framework, o componentă integrală a sistemului de operare Windows care include un sistem virtual de execuție, numit „Common Language Runtime” (CLR), și un set unificat de librării. Codul sursă scris în C# este compilat într-un limbaj intermediar (IL) care se conformează standardului internațional numit CLI (Common Language Infrastructure). Codul limbajului intermediar împreună cu resursele (cum ar fi poze, texte) sunt stocate pe disk în fișiere executabile care au de obicei extensia .exe sau .dll.

C# permite utilizarea programării orientate pe obiect respectând toate principiile enunțate anterior. Toate componentele limbajului sunt într-un fel sau altul, asociate noțiunii de clasă. Programul însuși este o clasă având metoda statică `Main()` ca punct de intrare, clasă ce nu se instanțiază. Chiar și tipurile predefinite `byte`, `int` sau `bool` sunt clase sigilate derivate din clasa `ValueType` din spațiul `System`.

### Declararea unei clase

**Sintaxa:** `[atrib] [modificatori] class [nume_clasă] [:clasa_de_bază] [corp_clasă]`

Atributele reprezintă informații declarative cu privire la entitatea definită.

Modificatorii reprezintă o secvență de cuvinte cheie dintre:

- modificatori de acces:
  - new
  - public
  - protected
  - internal
  - private
- modificatori de moștenire:
  - abstract
  - sealed (modificator de moștenire)

Clasa de bază este clasa de la care moștenește clasa curentă și poate exista o singură astfel de clasă de bază în cadrul unei moșteniri.

**Corpul clasei** este un bloc de declarații ale membrilor clasei:

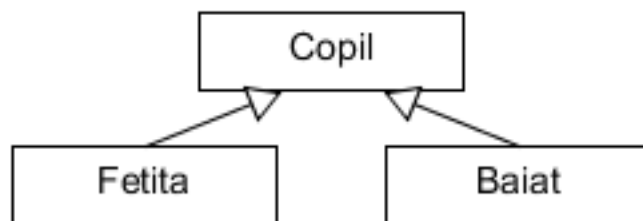
- constante (valori asociate clasei),
- câmpuri (variabile),
- tipuri de date definite de utilizator,

- metode (subprograme),
- constructori,
- un destructor,
- proprietăți (caracteristici ce pot fi consultate sau setate),
- evenimente (instrumente de semnalizare),
- indexatori (ce permit indexarea instanțelor din cadrul clasei respective) și
- operatori.

Constructorii și destructorul au ca nume numele clasei proxime din care fac parte. Metodele au nume care nu coincid cu numele clasei sau al altor membri (cu excepția metodelor, conform mecanismului de supraîncărcare). Metodele sau constructorii care au același nume trebuie să difere prin semnătură. Se pot defini date și metode statice (caracteristice clasei) și un constructor static care se execută la inițializarea clasei propriu-zise. Ele formează un fel de "context" al clasei. Se pot defini date și metode nestatice (de instanță) care se multiplică pentru fiecare instanță în parte în cadrul operației de instanțiere; Ele formează contextele tuturor instanțelor clasei respective.

Exemplul următor definește o ierarhie de clase (conform figurii alăturate):

```
public abstract class Copil
public class Fetita: Copil {
}
public sealed class Baiat:
Copil { }
```



Modificatorul `abstract` este folosit pentru a desemna faptul că nu se pot obține obiecte din clasa `Copil`, ci numai din derivatele acesteia (`Fetita`, `Baiat`), iar modificatorul `sealed` a fost folosit pentru a desemna faptul că nu se mai pot obține clase derivate din clasa `Baiat` (de exemplu, subclasele `Baiat_cuminte` și `Baiat_rau`).

## Contructori

**Sintaxa:** `[atrib] [modificatori] [nume_clasă] ([listă_param_formali]) [:inițializator] [corp_constr]`

Modificatori: `public`, `protected`, `internal`, `private`, `extern`

Inițializator: `base([listă_param])`, `this([listă_param])` ce permite invocarea unui constructor anume înainte de executarea instrucțiunilor ce formează corpul constructorului curent. Dacă nu este precizat niciun inițializator, se asociază implicit inițializatorul `base()`.

Corpul constructorului este format din instrucțiuni care se execută la crearea unui nou obiect al clasei respective (sau la crearea clasei, în cazul constructorilor cu modificatorul static). Pot exista mai mulți constructori care se pot diferenția prin lista lor de parametri. Constructorii nu pot fi moșteniți. Dacă o clasă nu are definit niciun constructor, se va asigura automat constructorul fără parametri al clasei de bază (clasa `Object`, dacă nu este precizată clasa de bază).

Instanțierea presupune declararea unei variabile de tipul clasei respective și inițializarea acesteia prin apelul constructorului clasei (unul dintre ei, dacă sunt definiți mai mulți) precedat de operatorul `new`. Acestea se pot realiza și simultan într-o instrucțiune de felul:

```
[Nume_clasă] [nume_obiect]=new [Nume_clasă] ([listă_param])
```

## Destructor

**Sintaxa:** `[atrib] [extern] ~[nume_clasă]() [corp_destructor]`

Corpul destructorului este format din instrucțiuni care se execută la distrugerea unui obiect al clasei respective. Pentru orice clasă poate fi definit un singur constructor. Destructorii nu pot fi moșteniți. În mod normal, destructorul nu este apelat în mod explicit, deoarece procesul de distrugere a unui obiect este invocat și gestionat automat de [Garbage Collector](#).

## Metode

**Sintaxa:** `[atrib] [modificatori] [tip_returnat] [nume]([listă_param_formali]) [corp_metoda]`

Modificatori: `new`, `public`, `protected`, `internal`, `private`, `static`, `virtual`, `abstract`, `sealed`, `override`, `extern`

Tipul rezultat poate fi un tip definit sau `void`.

Lista de parametri formali este o succesiune de declarații despărțite prin virgule, declararea unui parametru având sintaxa: `[atrib] [modificator] [tip] [nume]`

Modificatorul unui parametru poate fi `ref` (parametru de intrare și ieșire) sau `out` (parametru care este numai de ieșire). Parametrii care nu au niciun modificator sunt parametri de intrare. Un parametru formal special este parametrul tablou cu sintaxa: `[atrib] params [tip] [] [nume]`.

Pentru metodele abstracte și externe, corpul metodei se reduce la un semn ;

Semnătura fiecărei metode este formată din numele metodei, modificatorii acesteia, numărul și tipul parametrilor. Numele metodei trebuie să difere de numele oricărui alt membru care nu este metodă. La apelul metodei, orice parametru trebuie să aibă același modificator ca la definire. Invocarea unei metode se realizează prin sintagma `[nume_obiect].[nume_metoda]` (pentru metodele nestatice) și respectiv `[nume_clasă].[nume_metoda]` (pentru metodele statice).

## Proprietăți

Proprietatea este un membru ce permite accesul controlat la datele-membru ale clasei.

**Sintaxa:** `[atrib] [modificatori] [tip] [nume_proprietate] {[metode_de_acces]}`

Observațiile privind modificatorii și numele metodelor sunt valabile și în cazul proprietăților. Metodele de acces sunt două: `set` și `get`. Dacă proprietatea nu este abstractă sau externă, poate să apară una singură dintre cele două metode de acces sau amândouă, în orice ordine. Este o manieră de lucru recomandabilă aceea de a proteja datele membru (câmpuri) ale clasei, definind instrumente de acces la acestea:

- pentru a obține valoarea câmpului respectiv (`get`) sau
- de a memora o anumită valoare în câmpul respectiv (`set`).

Dacă metoda de acces `get` este perfect asimilabilă cu o metodă ce returnează o valoare (valoarea datei pe care vrem să o obținem sau valoarea ei modificată conform unei prelucrări suplimentare specifice problemei în cauză), metoda `set` este asimilabilă cu o metodă care ia un parametru de tip valoare (de intrare) și care atribuie (sau nu, în funcție de context) valoarea respectivă câmpului. Cum parametrul corespunzător valorii transmise nu apare în structura sintactică a metodei, este de știut că el este implicit identificat prin cuvântul `value`. Dacă se supune unor condiții specifice problemei, se face o atribuire de felul `câmp=value`.

Definirea în clasa `Copil` a proprietății `Nume`, corespunzătoare câmpului protejat ce reține, sub forma unui șir de caractere, numele copilului respectiv. Se va observa că proprietatea este moștenită și de clasele derivate `Fetita` și `Baiat`.

```
public class Copil
{...
```

```

    string nume; //este implicit protected

    public string Nume //proprietatea Nume
    {
        get
        {
            if (char.IsUpper(nume[0]))
                return nume;
            else
                return nume.ToUpper();
        }
        set { nume = value; } //metoda set
    }
    public Copil()
    {
        Nume = Console.ReadLine();
    }
}

class Fetita:Copil
{
    public override void se_joaca()
    {
        Console.WriteLine("{0} leagana papusa.",this.Nume);
    }
}

```

### Evenimente și delegări

Evenimentele sunt membri ai unei clase ce permit clasei sau obiectelor clasei să facă notificări, adică să anunțe celelalte obiecte asupra unor schimbări petrecute la nivelul stării lor. Clasa furnizoare a unui eveniment publică (pune la dispoziția altor clase) acest lucru printr-o declarație de eveniment care asociază evenimentului un delegat, adică o referință către o funcție necunoscută a căreia i se precizează doar antetul, funcția urmând a fi implementată la nivelul claselor interesate de evenimentul respectiv. Este modul prin care se realizează comunicarea între obiecte. Tehnica prin care clasele implementează metode (handler-e) ce răspund la evenimente generate de alte clase poartă numele de *tratare a evenimentelor*.

**Sintaxa:** [atrib] [modificatori] event [tip\_delegat] [nume]

Modificatorii permisi sunt aceiași ca și la metode.

Tipul delegat este un tip de date ca oricare altul, derivat din clasa sigilată *Delegate*, din spațiul *System*.

Definirea unui tip delegat se realizează prin declararea:

[atrib] [modificatori] delegate [tip\_rezultat] [nume\_delegat] ([listă\_param\_formali

Un delegat se poate defini și în afara clasei generatoare de evenimente și poate servi și altor scopuri în afara tratării evenimentelor.

### Interfețe

Interfețele sunt foarte importante în programarea orientată pe obiecte, deoarece permit utilizarea polimorfismului într-un sens mai extins.

O interfață este o componentă a aplicației, asemănătoare unei clase, ce declară prin membrii săi (metode, proprietăți, evenimente și indexatori) un ”comportament” unitar aplicabil mai multor clase, comportament care nu se poate defini prin ierarhia de clase a aplicației.

Pentru metodele din cadrul unei interfețe nu se dă nici o implementare, ci sunt pur și simplu specificate, implementarea lor fiind furnizată de unele dintre clasele aplicației.

Nu există instanțiere în cazul interfețelor, dar se admit derivări, inclusiv moșteniri multiple.

## **.NET Framework (Platforma .NET)**



### **Prezentare**

.NET este un cadru (framework) de dezvoltare software unitară care permite realizarea, distribuirea și rularea aplicațiilor-desktop Windows și aplicațiilor WEB. Tehnologia .NET pune laolaltă mai multe tehnologii (ASP, XML, OOP, și lista poate continua) și limbaje de programare (C++, C#, și la fel lista poate continua) asigurând totodată atât portabilitatea codului compilat între diferite calculatoare cu sistem Windows, cât și reutilizarea codului în programe, indiferent de limbajul de programare utilizat.

Pentru a dezvolta aplicații pe platforma .NET este bine să avem 3 componente esențiale:

- un set de limbaje (C#, Visual Basic .NET, C++, Pascal, sau altele mai mult sau mai puțin cunoscute),
- un set de medii de dezvoltare (Microsoft Visual Studio)
- și o bibliotecă de clase pentru crearea serviciilor Web, aplicațiilor Web și aplicațiilor desktop Windows.

Când dezvoltăm aplicații .NET, putem utiliza:

- Servere specializate - un set de servere Enterprise .NET (din familia SQL Server 2000, Exchange 2000 etc), care pun la dispoziție funcții de stocare a bazelor de date
- Servicii Web (în special comerciale), utile în aplicații care necesită identificarea utilizatorilor
- Servicii incluse pentru dispozitive non-PC

Componenta **.NET Framework** stă la baza tehnologiei .NET, este ultima interfață între aplicațiile .NET și sistemul de operare și actualmente conține:

- Limbajele C#, VB.NET, C++ și J#. Pentru a fi integrate în platforma .NET toate aceste limbaje respectă niște specificații OOP numite Common Type System (CTS). Ele au ca elemente de bază: clase, interfețe, delegări, tipuri valoare și referință, iar ca mecanisme: moștenire, polimorfism și tratarea excepțiilor.
- Platforma comună de executare a programelor numită Common Language Runtime, utilizată de toate cele 4 limbaje
- Ansamblul de biblioteci necesare în realizarea aplicațiilor desktop sau Web numit Framework Class Library



Componenta .NET Framework este formată din compilatoare, biblioteci și alte executabile utile în rularea aplicațiilor .NET.

.NET Framework include o vastă librărie, de peste 4000 de clase, organizate în ceea ce se numim „namespace”, care furnizează o varietate foarte mare de funcționalități, de la manipularea textelor furnizate ca intrări, la analizarea fișierelor XML, dar oferă de altfel și „Windows Forms Controls”. De asemenea .NET framework oferă suport pentru crearea aplicațiilor web.

## ASP.NET



### ASP.NET

ASP.NET este un model unic de dezvoltare web care include serviciile necesare pentru a construi aplicații web cu un minim de cod scris. ASP.NET este parte a .NET Framework.

Unul din primele elemente fundamentale ale WWW (World Wide Web) este HTML (Hypertext Markup Language), care descrie formatul primar în care documentele sunt distribuite și văzute pe Web. Multe din trăsăturile lui, cum ar fi independența față de platforma, structurarea formatarei și legăturile hipertext, fac din el un foarte bun format pentru documentele Internet și Web.

Stilurile pun la dispoziția creatorilor de site-uri noi posibilități de personalizare a paginilor Web. Un stil reprezintă un mod de a scrie un bloc de text (adică anumite valori pentru font, mărime, culoare, aliniere, distanțe față de margini etc).

**ASP (Active Server Pages)** reprezintă o tehnologie creată de Microsoft pentru a crea pagini web dinamice, ce are la bază stocarea și execuția scripturilor pe serverul Web.

Serverele Web care suportă tehnologia ASP sunt în număr de două:

- PWS - Personal Web Server (Windows 98, Me) și
- IIS – **Internet Information Server** (Windows NT, 2000 și ulterior).

Principalele caracteristici ale tehnologiei ASP sunt:

- Permite accesarea și actualizarea ușoară a bazelor de date;
- Viteză mare de execuție;
- Securitate ridicată, datorită faptului că scriptul nu poate fi vizualizat în browser;
- Generarea dinamică a răspunsurilor către clienții WEB.

Microsoft .NET este o platforma de dezvoltare software, care are menirea de a oferi programatorilor un mediu în care sunt puse la dispoziție diverse servicii cum sunt: managementul firelor de execuție, managementul duratei de viață a obiectelor (Garbage Collection), tratarea excepțiilor, etc; și care determină o viteză mult mai mare a dezvoltării aplicațiilor decât direct pe platforma Win32, adică direct pe sistemul de operare Windows.

Inovația adusă de Microsoft, constă în dezvoltarea unei platforme comune pentru mai multe limbaje de programare, platformă care permite dezvoltarea unei aplicații în orice limbaj dorește programatorul. Acest lucru este posibil prin dezvoltarea unui interpretor – Common Language Runtime – care transformă codul de program dintr-un limbaj oarecare, compatibil cu platforma .NET, în limbajul Microsoft Intermediate Language. Codul rezultat în limbajul Microsoft Intermediate Language, se poate apoi compila pentru orice sistem de operare care are instalat



.NET Framework. Paginile web dinamice create cu ajutorul ASP pot îngloba mai multe tipuri de tehnologii Web existente.

ASP .NET reprezintă o evoluție a ASP bazată pe o nouă tehnologie dezvoltată de Microsoft, si anume platforma: .NET Framework. Tehnologia ASP .NET, aduce îmbunătățiri semnificative față de ASP, cele mai evidente fiind următoarele:

- Execuția este mai rapidă;
- Este independent de programul de navigare pe Internet;
- Codul aplicației este compilat si executat de server; acesta, în cazul ASP, este interpretat pe măsura parcurgerii scripturilor;
- Utilizează noțiunea de code behind, adică separă partea de prezentare de partea de execuție a unei aplicații WEB;
- Favorizează reutilizarea codului, ceea ce în cazul ASP simplu, era o problemă, singura „reutilizare” care se făcea în ASP, fiind aceea de copiere a codului;
- Serializarea/deserializarea cu ușurință a obiectelor;
- Asigură interoperabilitatea între aplicații WEB, între aplicații WEB si alte categorii de aplicații;
- Securitate crescută;

Microsoft ASP .NET este următoarea generație a tehnologiei de dezvoltare a aplicațiilor Web. Ea preia tot ce este mai bun de la Active Server Pages (ASP) la fel ca si serviciile bogate si facilitățile oferite de Common Language Runtime (CLR) si multe alte facilități noi. Rezultatul este o nouă modalitate de dezvoltare web rapidă, scalabilă si robustă care permite o mare flexibilitate cu foarte puține linii de cod scrise.

ASP.NET offeră trei platforme pentru crearea aplicațiilor web:

- ASP.NET Web Forms
- ASP.NET MVC
- ASP.NET Web Pages

**ASP.NET Web Forms (.aspx pages)** se adresează programatorilor care preferă programarea declarativă și bazată pe controale. **Web Forms** reprezintă partea centrală pe care se bazează ASP .NET. Acestea reprezintă elementele de interfață cu utilizatorul care dau aplicațiilor web aspectul si comportamentul dorit. Formularele Web sunt similare formularelor Windows prin faptul că oferă proprietăți, metode si evenimente controalelor plasate pe ele. Totusi, aceste elemente de interfață sunt afisate prin intermediul limbajului HTML, iar în cazul utilizării Microsoft Visual Studio .NET poate fi folosită interfața familiară de tip drag-and-drop pentru crearea aspectului formularelor Web. Formularele Web sunt constituite din două componente: partea vizuală (fișierul .ASPX) si codul din spatele formularului, care este stocat în fișiere separate.

ASP.Net Web Forms este:

- Bazat pe tehnologia Microsoft ASP.NET, în care codul care rulează pe server generează dinamic pagina web de ieșire la dispozitivul de browser sau client.
- Compatibil cu orice browser sau dispozitiv mobil, astfel codul HTML generat păstrează stilurile și alte caracteristici de la un browser la altul.

- Compatibil cu orice limbaj de programare acceptat de limbajul comun .NET, cum ar fi Microsoft Visual Basic și Microsoft Visual C#.
- Construit pe Microsoft .NET Framework. Acest lucru oferă toate beneficiile cadrului, inclusiv un mediu gestionat, de siguranță și de moștenire.
- Flexibil pentru că există posibilitatea adăugării de controale create de utilizator sau din alte surse exterioare.

ASP.NET Web Forms oferă:

- Separarea codului HTML sau a oricărui alt cod pentru UI, de logica de aplicație
- O suită bogată de sistem de control pentru activități obișnuite, inclusiv acces la date.
- Suport pentru client-side scripting care execută în browser.
- Suport pentru o varietate de alte capacități, inclusiv de rutare, securitate, performanță, internaționalizare, testare, depanare, eroare de manipulare și de gestionare de stări.

Caracteristici ale ASP.NET Web Forms:

- Controale Server - controalele de tip Server ASP.NET Web sunt obiecte pe paginile web ASP.NET, care se execută atunci când se solicită pagina și care notifică browserul.
- Pagini Master - pagini master ASP.NET permit unui aspect consistent pentru paginile din aplicația.
- Lucrul cu date - ASP.NET oferă multe opțiuni de stocare, recuperarea, și afișarea datelor.

Pentru a prezenta informații în navigatorul clientului folosim formularele Web ASP.NET care oferă o abstractizare în modelul de programare, un model orientat obiect și bazat pe evenimente. Acest mediu de lucru beneficiază de toate facilitățile oferite de platforma .NET (siguranța tipurilor, mediu de execuție controlat, moștenire) și reprezintă o înlocuire a clasicelelor formulare HTML. Componenta vizuală este reprezentată de un fișier cu extensia .aspx acționând ca un container pentru HTML, text static și controale server care pot fi afișate în browser, iar logica aplicației este reprezentată de un fișier cu extensia .cs (pentru limbajul Visual C#) sau .vb (pentru Visual Basic.NET). Fișierele .aspx mai sunt referite ca pagini ASP.NET. Aceasta tehnica de separare a codului de partea de prezentare este numită „code-behind programming”.

Logica aplicației reprezintă în fapt o clasă care extinde funcționalitatea clasei

`System.Web.UI.Page`

Această clasă conține metode care tratează diferite evenimente ce apar în timpul execuției formularului Web la server (de exemplu, dacă metoda `Page_Load` este conectată la evenimentul `Load` al clasei de bază `Page`, atunci aceasta este apelată la fiecare acces al unui formular Web), proprietăți (de exemplu, prin proprietatea `IsPostBack` putem afla dacă o pagină Web este la primul acces sau la accesări ulterioare), atribute corespunzătoare unor controale din pagina WebForms și alte date membre necesare implementării aplicației. O pagină WebForms, la procesarea pe serverul Web, poate fi privită ca un program executabil pentru care ieșirea standard o reprezintă browserul sau dispozitivul client. În acest model, pagina trece printr-o serie de stadii de procesare: inițializare, procesare și eliberare.

În ordinea apariției, acestea sunt:

- **Init**, eveniment care inițializează pagina și în care proprietățile controalelor sunt actualizate. Aici este corect să inițializăm controale care se adaugă dinamic la pagina sau variabile necesare înainte de inițializarea paginii;

- **Load** poate fi numit locul în care utilizatorul își inițializează codul. Evenimentul este generat de fiecare dată când pagina este încărcată după ce controalele au fost inițializate la pasul anterior;
- **Tratarea evenimentelor** utilizator, reprezintă stagiul în care sunt tratate evenimentele generate de client cum ar fi: schimbarea unui text într-un control, apăsarea unui buton etc. Trebuie reținut că aceste evenimente nu sunt tratate într-o anumită ordine pe server, iar tratarea lor are loc după apariția unui eveniment Click care trimite formularul la server (a unui submit);
- **PreRender**, eveniment care poate fi folosit pentru a face ultimele actualizări asupra paginii Web înainte ca aceasta să fie generată la client;
- **Render**, eveniment care generează la client reprezentarea HTML a paginii Web ASP.NET încărcată de la server;
- **Unload** este ultimul eveniment care se execută înainte ca pagina să fie eliberată. Evenimentul este util de folosit atunci când dorim să efectuăm ultimele operații de eliberare a resurselor: închiderea fișierelor, a conexiunilor la baza de date și eliberarea obiectelor din memorie.

Există două tipuri de bază în care pot fi împărțite **controalele ASP.NET**:

- **HTML Controls**: reprezintă elemente HTML care pot fi programate la nivelul serverului și expun un model obiectual restricționat la capacitățile elementelor HTML pe care le asigază.
- **Web Controls**: adică facilități superioare controalelor HTML incluzând controale mult mai complexe, cum ar fi controlul calendar, iar modelul obiect nu reflectă neapărat sintaxa HTML.

Controalele HTML sunt asemenea elementelor HTML folosite cu ajutorul Frontpage sau al oricărui alt editor HTML. Pot fi folosite și elementele standard HTML într-un Formular Web, de exemplu pentru a crea o casetă de text: `<input type="text" id=txtFirstName size=25>`

ASP.NET definește un al doilea tip de controale - **Web Controls**.

Numite și controale inteligente, ele pot fi caracterizate prin:

- oferă un bogat și consistent model obiectual de programare;
- detectează automat tipul navigatorului, iar afișarea la client va fi optimizată în funcție de capacitățile acestuia;
- pentru unele controale se pot defini șabloane (template-uri) de afișare;
- posibilitatea de a controla generarea evenimentelor pe server;
- posibilitatea de a trimite evenimente unui container din interiorul acestuia (de exemplu, un control de tip buton în interiorul unui tabel);
- legarea la surse de date a tuturor proprietăților controalelor pentru a influența afișarea la execuție.

Sunt definite în spațiul cu numele `System.Web.UI.WebControls` și moștenesc, direct sau indirect, clasa de bază `WebControl`.

În general toate informațiile care sunt afișate în formulare web sunt stocate sau trebuie stocate. O modalitate ușoară de a stoca datele este folosirea unei baze de date. Descrierea a ceea ce înseamnă un „System de baze de date relaționat” împreună cu descrierea structurii bazei de date folosite pentru realizarea acestei aplicații se află în capitolul 4 dedicat aplicației software. Înainte de a ajunge la acest capitol sunt prezentate câteva informații despre tehnologiile folosite pentru accesarea ușoară a datelor și pentru stocarea lor.

## Entity Framework



Entity Framework este un cadru (framework) de mapare relațional obiectuală pentru platforma .Net. Per ansamblu este un set de tehnologii ADO.Net ce susține dezvoltarea de aplicații software orientate pe date. Arhitecții și dezvoltatorii de aplicații orientate pe date s-au străduit să atingă două obiective foarte diferite:

1. Trebuie să modeleze entitățile, relațiile și logica de business a problemei pe care o rezolvă;
2. Trebuie să lucreze cu motorul de date folosit pentru a stoca și regăsi datele.

Acest framework permite lucrul cu date sub forma unor obiecte și proprietăți specifice. Cu Entity Framework rândurile și coloanele din tabele sunt returnate ca obiecte și nu se mai folosesc comenzile normale de manipulare a datelor din bazele de date (SELECT, UPDATE, INSERT, etc.). Datele se translatează din formă tabelară în obiecte. Astfel cu mai puține linii de cod se pot realiza mult mai multe operații.

Prima versiune de Entity Framework, Entity Framework v 1, a fost inclusă în .Net Framework 3.5 Service Pack 1 și Visual Studio 2008 Service Pack1 și lansată în August 2008.

Arhitectura Entity Framework conține următoarele elemente:

- Furnizor specific de surse de date;
- Map provider – un furnizor specific bazei de date ce traduce commanda SQL specifică Entity Framework într-o interogare nativă a bazei de date;
- View mapping – din schema relațională crează view-uri ale datelor corespundente modelului conceptual;
- Query and update pipeline – procesează interogări, filtrează și face update cererilor;
- Metadata services – gestionează metadatele legate de entități, relații și mapări;
- Tranzacții;
- Conceptual layer API – expune modelul de programare;
- Embedded database – include o bază de date încorporată pentru interogarea datelor relaționale;
- Desing tools – simplifică task-ul de mapare a schemei conceptuale la schema relatională;

- Programming layer – expune EDM-ul (Entity Data Model) ca unități ce pot fi utilizate de limbajul de programare;
- Object services – cod generat automat pentru clasele CLR ce dețin aceleași proprietăți ca o entitate, dând astfel posibilitatea instanțierii entităților
- Web services – expun entitățile ca servicii web;

Entity Framework are un loc bine definit în aplicațiile în a căror dezvoltare este implicată această tehnologie. În figura următoare este ilustrat locul pe care îl ocupă Entity Framework în elaborarea unei aplicații.

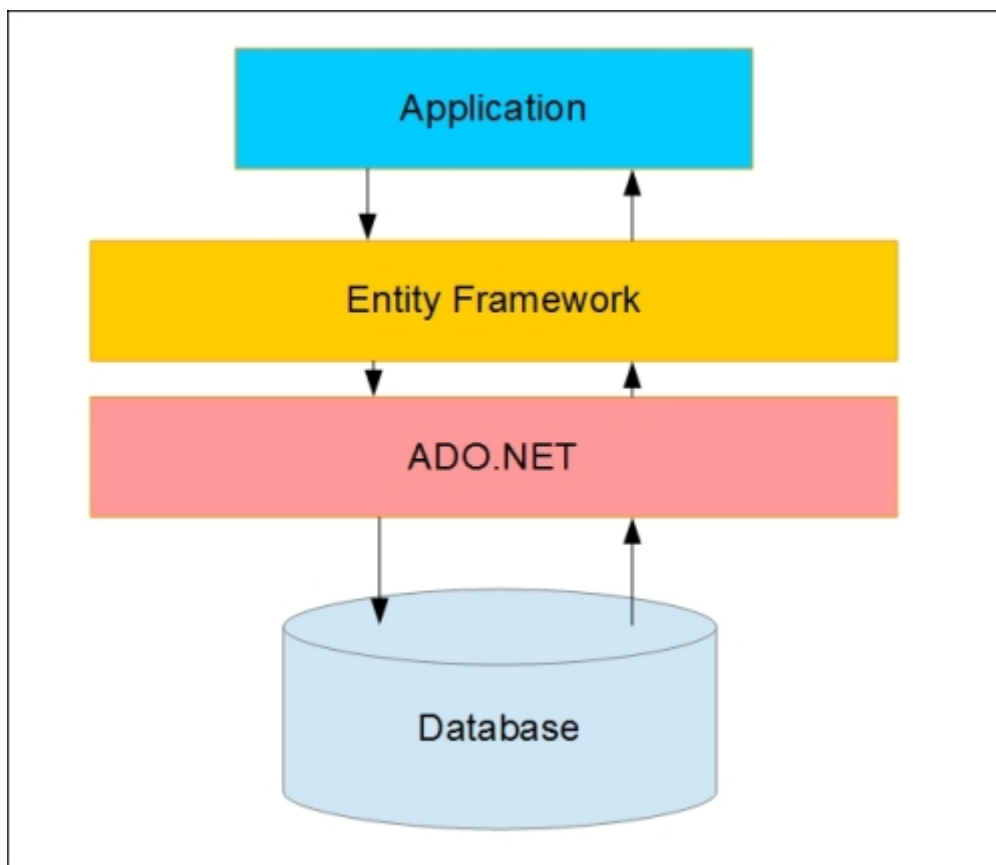


Figure 9: Entity Framework într-o aplicație

Entity Framework folosește un model numit **Entity Data Model** (EDM), dezvoltat din Entity Relationship Modeling.

Entity Relationship Model se ocupă în principal de entități și asocierile din care acestea fac parte. În Microsoft Visual Studio, Entity Data Model Wizard generează o mapare 1:1 a schemei bazei de date la chema conceptuală. În chema relațională elementele sunt compuse din tabele, în care cheile primare și cheile externe tin tabele corelate în strânsă legătură. În contrast **Entity Types** definește schema conceptuală a datelor. Tipurile de entități sunt o agregare a unor tipuri de câmpuri multiple, fiecare câmp mapând o coloană din baza de date, câmpuri ce pot conține informații din mai multe tabele fizice. Tipurile de entități pot fi corelate unele cu altele, independent de relațiile din schema fizică. Entitățile corelate sunt expuse similar prin intermediul unui câmp al cărui nume denotă relația în care acestea se află. Schema logica și maparea ei pe schema fizică este reprezentată ca un Entity Data Model și specificată ca un

fișier XML. Entitățile sunt instanțe ale tipurilor de entități, reprezentând instanțe individuale ale obiectelor (de exemplu profesor sau lucrare de licență) la care se referă informația. Tipul de entitate definește clasa de care aparține o entitate, dar și proprietățile pe care o entitate le are. Proprietățile descriu anumite aspecte ale entității dându-i acesteia un nume și un tip. Oricare două tipuri de entități pot avea o legătură printr-o relație. Cum au fost definite și mai sus, există mai multe tipuri de relații care se bazează pe multiplicitate:

- relațiile de one-to-one (unul-la-unul)
- relațiile de one-to-many (unul-la-mai mulți)
- relațiile de many-to-many (mai mulți-la-mai mulți)

Un tip de relație poate avea atașată o acțiune sau o operație. Unei relații i se poate specifica să efectueze o acțiune atunci când o operație este executată asupra unei entități cu care are legătură.

ADO.Net Entity Framework folosește o variantă de limbaj structurat de interogare denumit Entity SQL ce a fost creat pentru a scrie interogări și actualizări asupra entităților și relațiilor la nivel conceptual.

Componentele principale ale ADO.NET sunt DataSet și Data Provider. Ele au fost proiectate pentru accesarea și manipularea datelor. Din cauza existenței mai multor tipuri de surse de date este necesar ca pentru fiecare tip de protocol de comunicare să se folosească o bibliotecă specializată de clase. .NET Framework include SQL Server.NET Data Provider pentru interacțiune cu Microsoft SQL Server, Oracle Data Provider pentru bazele de date Oracle și OLE DB Data Provider pentru accesarea bazelor de date ce utilizează tehnologia OLE DB pentru expunerea datelor (de exemplu Access, Excel sau SQL Server versiune mai veche decât 7.0).

Furnizorul de date permite unei aplicații să se conecteze la sursa de date, execută comenzi și salvează rezultate. Fiecare furnizor de date cuprinde componentele Connection, Command, DataReader și DataAdapter. Accesul direct la date presupune construcția unei comenzi SQL, conectarea la baza de date, executarea comenzii și deconectarea de la baza de date fără memorarea directă a rezultatelor. Dacă se lucrează în mod deconectat atunci se păstrează o copie a datelor într-un obiect de tip DataSet și acestea pot fi prelucrate și după deconectarea de la baza de date. Modelul accesului direct la date corespunde paginilor ASP.NET unde nu este necesară memorarea datelor pentru perioade mai lungi de timp. O pagină ASP.NET este încărcată la cerere și accesul se încheie atunci când rezultatul cererii este furnizat user-ului, ceea ce înseamnă o pagină care are de obicei o viață de numai câteva secunde. Rezultă așadar că o interogare directă a datelor presupune executarea pașilor următori:

1. crearea obiectelor de tip Connection, Command și DataReader;
2. obținerea informațiilor din baza de date cu ajutorul obiectelor de tip DataReader și afișarea acestora într-un controler de pe un formular web;
3. închiderea conexiunii;
4. trimiterea paginii către utilizator; în acest moment nu mai avem o legătură directă între ceea ce vede utilizatorul și datele din baza de date, obiectele de tip ADO.NET fiind distruse.

Adăugarea, ștergerea sau modificarea datelor se realizează în doi pași:

1. crearea obiectelor de tip Connection, Command;
2. executarea comenzii directe.

Înainte de orice operație cu o sursă de date externă, trebuie realizată o conexiune (legătură) cu acea sursă. În cazul lucrării de față sursa este baza de date creată în Microsoft SQL Server.

## Microsoft SQL Server



Microsoft SQL Server este un sistem de gestionare de baze de date relaționale (RDBMS), produs de compania americană Microsoft. Microsoft SQL Sever folosește o variantă de SQL numita Transact-SQL(T-SQL), care este o implementare de SQL-92 cu unele extensii pentru procedurile stocate și tranzacții. Cel mai răspândit tip de baze de date este cel relațional, în care datele sunt stocate în tabele. Pe lângă tabele, o bază de date relațională mai poate conține și alte obiecte cum ar fi vederi, indecși, proceduri și funcții stocate, declanșatori, constrângeri de integritate.

Codul de bază pentru Microsoft SQL Server își are originile în Sybase SQL Server și a reprezentat intrarea Microsoft pe piața bazelor de date pentru întreprinderi, concurând cu Oracle, IBM și Sybase. Microsoft, Sybase și Ashton-Tate s-au unit pentru a crea și a scoate pe piață prima versiune numita SQL Server 4.2 pentru Win OS/2. Mai tarziu Microsoft a negociat pentru drepturi de exclusivitate la toate versiunile de SQL Server scrise pentru sistemele de operare Microsoft. Sybase și-a schimbat ulterior numele în Adaptive Server Enterprise, pentru a evita confuzia cu Microsoft SQL Server.

Versiunea SQL Server 2008, lansată pe 6 august 2008 cu nume de cod Katmai aduce și ea, așa cum era de așteptat, alte îmbunătățiri. Acum oferă suport pentru stocarea datelor multimedia și adăuga noi tipuri de date (geometry, geography, hierarchy și mult așteptatul date fără datetime). Versiunea de SQL Server Management Studio inclusă în SQL Server 2008 acceptă IntelliSense pentru SQL.

Trebuie precizat că Microsoft oferă SQL Server Express Edition, versiune gratuită a serverului. Cu toate că nu oferă restricții în ceea ce privește numărul bazelor de date sau a utilizatorilor concurenți, este limitată la folosirea unui singur procesor, a 1 Gb de memorie și max. 4Gb a fișierelor de date.

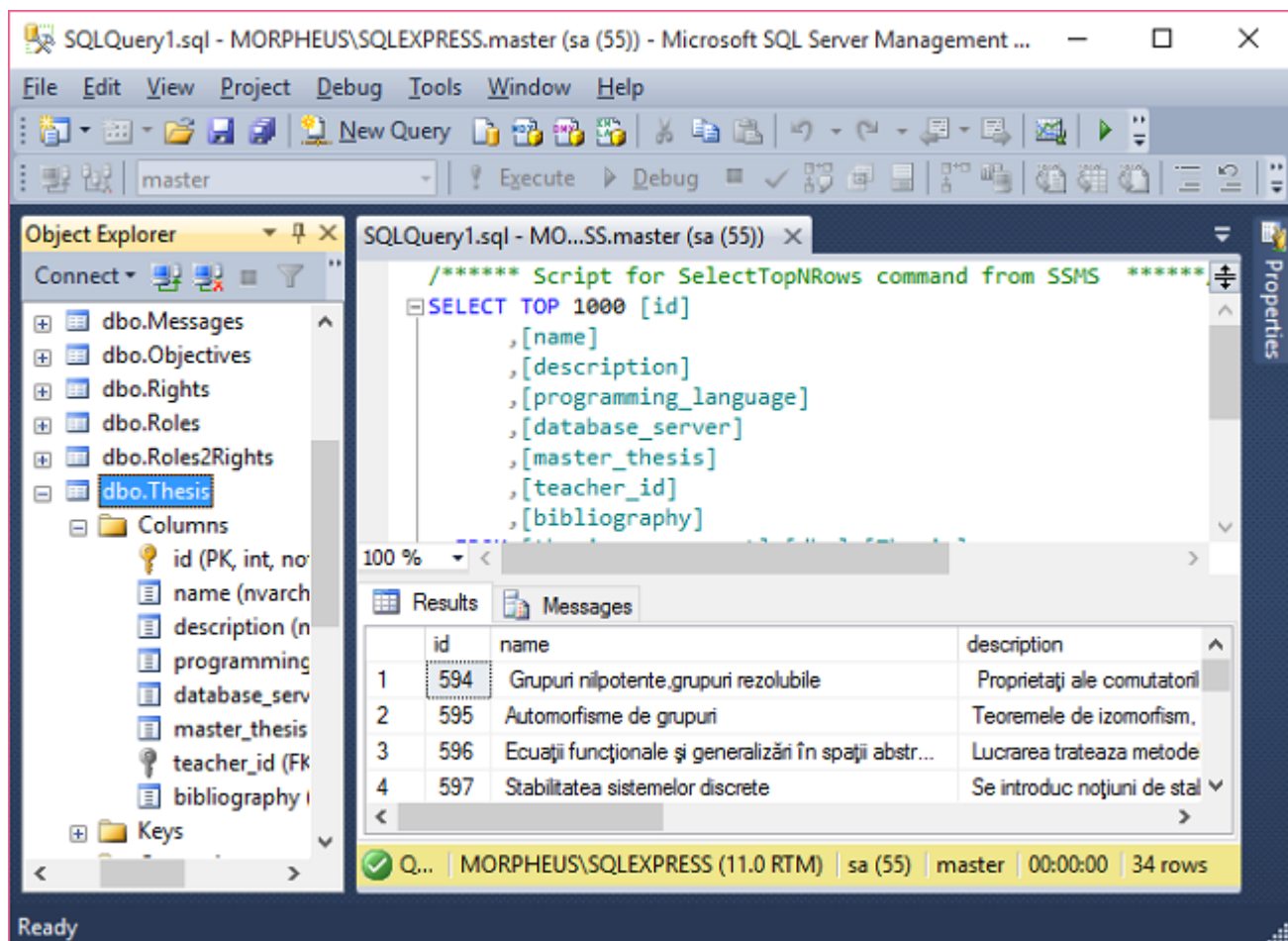


Figure 10: SQL Server Management Studio

Putem spune despre Microsoft SQL Server că este o soluție integrată de management și analiză a datelor, care ajută organizațiile de orice dimensiune să:

- Dezvolte, implementeze și administreze aplicații la nivel de întreprindere mai sigure, scalabile și fiabile
- Maximizeze productivitatea IT prin reducerea complexității creării, implementării și administrării aplicațiilor pentru baze de date.
- Partajeze date pe mai multe platforme, aplicații și dispozitive pentru a facilita conectarea sistemelor interne și externe.
- Controleze costurile fără a sacrifica performanța, disponibilitatea, scalabilitatea sau securitatea.

Microsoft SQL Server include o serie de componente dintre care cele mai importante componente care au fost utilizate și la realizarea acestei aplicații sunt:

- SQL Server Database Engine: Include motorul de baze de date, elementul central pentru stocarea, procesarea și securizarea datelor, replicare, căutare full-text, instrumente pentru gestionarea datelor XML și relationale.
- Aplicația SQL Server Management Studio



Gestionarea serverului SQL se face foarte ușor prin aplicația SQL Server Management Studio (figura anterioară). Elementul central al acestei unelte este panelul Object Explorer, ce permite utilizatorului să răsfoiască, să selecteze sau să întreprindă orice altă acțiune asupra obiectelor de pe server.

De asemenea integrarea produselor Microsoft pentru realizarea unei aplicații software se face foarte ușor datorită utilizării prietenoase și ușoare de care dispun produsele Microsoft. De cele mai multe ori din doar câteva clickuri se poate realiza legătura dintre produsele Microsoft.

Pentru dezvoltarea aplicației a fost folosit Microsoft Visual Studio 2012, un mediu de dezvoltare foarte prietenos și ușor de folosit. Alegerea acestui mediu și în general a tehnologiilor Microsoft a fost ușor de luat. Microsoft pe lângă aceste tehnologii oferă și documentația bine structurată necesară pentru un dezvoltator care nu s-a întâlnit cu aceste tehnologii, făcând astfel dezvoltarea aplicației mult mai ușoară.

MSDN (Microsoft Developer Network) Library este o librărie care conține documentația tehnică oficială pentru orice înseamnă tehnologie Microsoft. Conținutul acestei librării online se adresează în special dezvoltatorilor software care doresc să utilizeze Microsoft.

# Capitolul 4

## Descrierea Aplicației Software

### Introducere

Așa cum am prezentat și în introducerea generală, lucrarea de față conține și aplicația software pentru gestiunea lucrărilor de licență. În general în elaborarea unei aplicații de gestiune se pornește de la datele de intrare care trebuie gestionate. Atfel va trebui ca aceste date să fie stocate înainte ca ele să intre în fluxul efectiv de gestionare. În secțiunile următoare sunt prezentate atât informații despre stocarea datelor cât și informații despre prelucrarea datelor folosind aplicația software implementată. Aplicația software implementată este o aplicație web, este un „Portal pentru gestiunea lucrărilor de licență”. Interfața web este realizată în limba Engleză, dar aceasta poate fi ușor tradusă și utilizată și în alte limbi.

Acest capitol începe prin a prezenta baza de date folosită pentru stocarea datelor și se continuă cu explicații despre cum poate fi folosită aplicația software.

### Baza de Date

Stocarea datelor se realizează prin folosirea unei baze de date relaționate. O bază de date este și ea la rândul ei gestionată de un sistem de gestiune a bazelor de date (în limba Engleză este cunoscut sub numele de Database Management System). Sistemele de gestiune a bazelor de date reprezintă totalitatea programelor utilizate pentru crearea, interogarea și întreținerea unei baze de date. Include două categorii de module: module care sunt comune cu cele ale sistemelor de operare ale calculatoarelor și module cu funcții specifice bazei de date. Sub-sistemele monitor conțin programele de control ale perifericelor și sistemul de gestiune a fișierelor. Sub-sistemele externe sunt alcătuite din procesorul de definiție și programul de administrare. Alături de acestea există programare de descriere a bazei de date și cerei de prelucrare. Între utilizator și sistem există două interfețe:

- definirea bazei de date
- utilizarea bazei de date

Definirea unei baze de date se execută sub controlul procesorului de definiție, capabil să prelucreze programe de descriere, formulate folosind limbaje specializate cunoscute sub denumirea de limbaje de definiție a datelor.

Sistemul de gestiune folosit pentru această baza de date este: Microsoft SQL Server. Mai multe informații au fost prezentate în capitolul anterior.

Entitatea fundamentală dintr-o bază de date este „tabela”. Baza de date folosită în implementarea acestei aplicații este constituită din următoarele tabele:

- **Users:** în această tabelă sunt stocați toți utilizatorii aplicației fie ei profesori, administratori sau studenți
- **Roles:** această tabelă stochează rolurile utilizatorilor; în aplicație există trei roluri: rolul de administrator, rolul de profesor și rolul de student
- **User2Roles:** tabela aceasta leagă un utilizator de rolul lui în această aplicație; este o tabela de legătură prin intermediul căreia se realizează o legătură de „many-to-many” (mai mulți-la-mai mulți) între utilizatori și roluri

- **Thesis:** în această tabelă sunt stocate toate lucrările de licență împreună cu descrierea corespunzătoare fiecărei lucrări
- **Assignments:** această tabelă stochează asocierile dintre o lucrare de licență și un student; prin intermediul ei se rețin informații legate de asignarea lucrărilor de licență
- **Objectives:** tabelă care stochează obiectivele care trebuie atinse pe parcursul elaborării unei lucrări de licență; în aplicație există 3 obiective implicite: etapa de documentare generală, întocmirea structurii preliminare, redactarea lucrării de licență
- **Messages:** pentru fiecare obiectiv din tabela anterioară un profesor poate adăuga unul sau mai multe mesaje; aceste mesaje sunt stocate în această tabelă

Figura următoare conține diagrama bazei de date realizată folosind aplicația „SQL Server Management Studio”.



Figure 11: Diagrama bazei de date

Fiecare tabela este constituită din mai multe coloane, fiecare coloană având o semnificație bine definită. Descrierea în detaliu a fiecărei tabele este realizată în rândurile următoare:

- **Users:**

- **id (int):** cheia primară (primary key) a tabelului; fiecare entitate se identifică unic prin această cheie primară
- **username (nvarchar):** numele utilizatorului folosit pentru accesarea aplicației; este unic în această tabelă
- **password (nvarchar):** parola utilizatorului folosită pentru accesarea aplicației
- **firstname (nvarchar):** nume utilizator
- **lastname (nvarchar):** prenume utilizator
- **email (varchar):** adresa de email a utilizatorului
- **cnp (varchar):** codul numeric personal al studentului
- **title (varchar):** încadrarea profesorului
- **specialization (nvarchar):** secția în cadrul căreia este înscris studentul

- **Roles:**

- **id (int):** cheia primară (primary key) a tabelului de roluri; fiecare entitate se identifică unic prin această cheie primară
- **name (varchar):** numele rolului (în cazul aplicației de față există 3 roluri: Administrator, Profesor, Student preferite)
- **description (varchar):** descrierea rolului (nu este obligatorie)
- **bid (int):** indicator unic pentru rol (pentru ușurarea implementării)

- **User2Roles:**

- **id (int):** cheia primară (primary key) a tabelului de „Users2Roles”; fiecare entitate se identifică unic prin această cheie primară
- **user\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Users”
- **role\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Roles”

- **Thesis:**

- **id (int):** cheia primară (primary key) a tabelului de „Thesis”; fiecare entitate se identifică unic prin această cheie primară
- **name (nvarchar):** numele lucrării de licență
- **description (nvarchar):** descrierea lucrării de licență
- **teacher\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Users”
- **bibliography (nvarchar):** bibliografia corespunzătoare lucrării de licență

- **Assignments:**

- **id (int):** cheia primară (primary key) a tabelii de „Assignments”; fiecare entitate se identifică unic prin această cheie primară
- **student\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Users”
- **thesis\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Thesis”
- **approved (bit):** are valoarea 0 sau 1; are valoarea 1 dacă o lucrare de licență a fost aprobată pentru a fi asignată unui student
- **status\_message (nvarchar):** mesajul pe care profesorul poate să îl transmită studentului în momentul în care aprobă sau dezaprobă o lucrare de licență

● **Objectives:**

- **id (int):** cheia primară (primary key) a tabelii de „Objectives”; fiecare entitate se identifică unic prin această cheie primară
- **assignment\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Assignments”
- **title (nvarchar):** Titlul obiectivului care trebuie atins pe parcursul elaborării unei lucrări de licență
- **index (int):** determină ordinea obiectivelor pentru elaborarea unei lucrări de licență
- **target (datetime):** termenul limită pentru atingerea obiectivului
- **completion (datetime):** data la care s-a realizat obiectivul
- **grade (float):** nota obținută pentru realizarea obiectivului

● **Messages:**

- **id (int):** cheia primară (primary key) a tabelii de „Messages”; fiecare entitate se identifică unic prin această cheie primară
- **objectiv\_id (int):** cheie străină (foreign key) care reprezintă relația către cheia primară din tabela „Objectives”
- **text (nvarchar):** textul mesajului
- **date (datetime):** data la care a fost adăugat mesajul

## **Administrarea datelor folosind aplicația**

Se pune problema cum putem face rost de aceste date și cum ajung ele în aplicație. După cum a fost descris în secțiunea anterioară există mai multe roluri în aplicație. Pentru ca aplicația să fie utilizabilă am creat un administrator al acestei aplicații.

Odată ce un utilizator cu rol de administrator se autentifică în baza de date folosind formularul de autentificare din figura următoare, doar anumite informații sunt afișate.

The image shows a web browser window with the address bar displaying 'localhost:8625/Login.aspx'. The page has a dark blue header with the text 'System for the Management of Graduation Theses' in white. Below the header, there is a login form with two input fields: 'Username:' and 'Password:'. A 'Login' button is positioned below the password field.

Figure 12: Formularul de autentificare

### Rolul de administrator

Informațiile care sunt afișate și la care este necesar ca un administrator să aibă access sunt afișate în următoarea figură și descrise în cele ce urmează:

- **Roles:** posibilitatea de vizualizare a rolurilor ce pot fi folosite în aplicație. Deoarece rolurile ajung în strânsă legătură cu implementarea aplicației acestea nu pot fi modificate.
- **Administrators:** lista de utilizatori cu rolul de administrator. Un administrator poate să adauge alți utilizatori cu rol de administrator și de asemenea poate să modifice rolul asignat unui utilizator
- **Teachers:** lista de utilizatori cu rolul de profesor. Un administrator poate să adauge utilizatori cu rolul de profesor și de asemenea poate să modifice utilizatorii cu rolul de profesor existenți.
- **Students:** lista de utilizatori cu rolul de student. Un administrator poate să adauge utilizatori cu rolul de student și de asemenea poate să modifice utilizatorii cu rolul de student existenți.

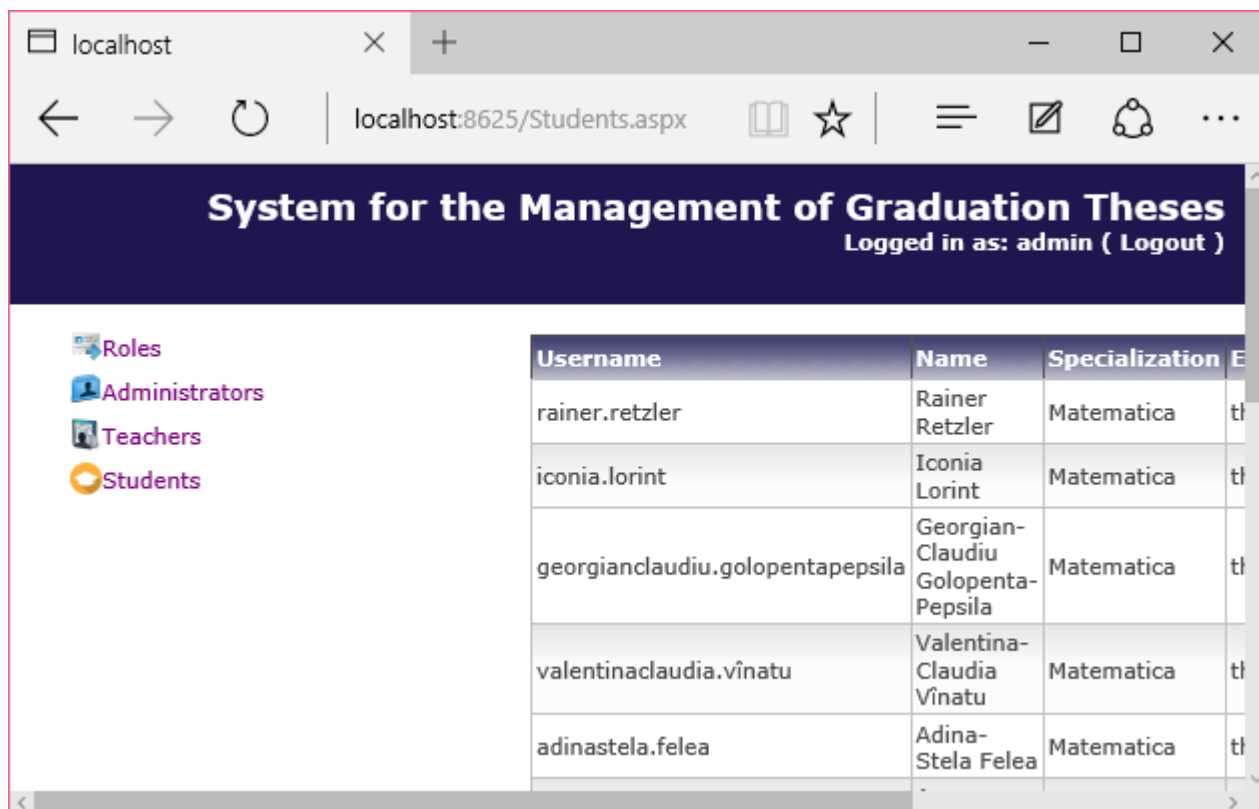


Figure 13: Meniul administratorului

### Rolul de profesor

La fel ca și utilizatorul cu rol de administrator, cel cu rol de profesor are access doar la anumite date ale aplicației. Odată ce un utilizator cu rolul de profesor se autentifică în aplicație un meniu special pentru acest rol este afișat în aplicația software. Meniul utilizatorului cu rolul de profesor este exemplificat în imaginea următoare, odată cu exemplificarea afișării lucrărilor de licență asignare profesorilor.

Descrierea meniului afișat pentru rolul de profesor:

- **Assignments:** conține lista de lucrări de licență asignare studentilor. Aici profesorul poate adăuga obiective care se vor atinge pe durata elaborării lucrării de licență și poate monitoriza gradul de atingere a acestor obiective.

**System for the Management of Graduation Theses**  
Logged in as: petru.jebelean ( Logout )

Assignments  
Requests  
Theses

Name	Description	Bibliography	
Operatori compacți și aplicații	Proprietăți de baza ale operatorilor liniari și compacți, cu aplicații în teoria ecuațiilor diferențiale și integrale, precum și în teoria aproximării.	R. Cristescu, Analiza Funcțională, Ed. Didactică și Pedagogică, București, 1983 (orientativ)	<a href="#">View</a>
Ecuații integrale	Ecuații integrale de tip Volterra și Fredholm, liniare și neliniare.	T. Havarneanu, Ecuații Integrale, Ed. Alexandru Myler, Iași, 2007 (orientativ)	<a href="#">View</a>

Figure 14: Meniul profesorului (Assignments)

- **Requests:** conține lista de cereri făcute de către studenți folosind meniul dedicat lor. Aici profesorul poate vizualiza cererile studenților și de asemenea poate să aprobe sau să dezaprobe o cerere. Odată cu aprobarea sau dezaprobară unei cereri de licență profesorul poate să adauge mesaje explicative. Cerile de licență sunt afișate în figura următoare:



localhost

localhost:8625/StudentsReq

## System for the Management of Graduation Theses

Logged in as: petru.jebelean ( Logout )

- Assignments
- Requests
- Theses

Student Name	Thesis Name	Thesis Description	Status	Info
Rainer Retzler	Ecuatii integrale	Ecuatii integrale de tip Voltera si Fredholm, liniare si neliniare.	Approved	<a href="#">View</a>
Alina Elena Radescu	Ecuatii integrale	Ecuatii integrale de tip Voltera si Fredholm, liniare si neliniare.	Pending	<a href="#">View</a>

Figure 15: Meniul profesorului (Requests)

- **Theses:** conține lista de lucrări de licență împreună cu datele corespunzătoare fiecărei lucrări de licență. Aici profesorul are posibilitatea să adauge noi lucrări de licență, să editeze lucrările deja adăugate și să șteargă lucrările de licență care nu sunt asignare studenților.

**System for the Management of Graduation Theses**  
Logged in as: petru.jebelean ( Logout )

Assignments  
Requests  
Theses

Name	Description	Theacher Name	Bibliography	Allocated
Operatori compacti și aplicații	Proprietăți de baza ale operatorilor liniari și compacti, cu aplicații în teoria ecuațiilor diferențiale și integrale, precum și în teoria aproximării.	Petru Jebelean	R. Cristescu, Analiza Funcțională, Ed. Didactica și Pedagogică, București, 1983 (orientativ)	0
Ecuatii integrale	Ecuatii integrale de tip Volterra și Fredholm, liniare și neliniare.	Petru Jebelean	T. Havarneanu, Ecuatii Integrale, Ed. Alexandru Myler, Iasi, 2007 (orientativ)	2

Figure 16: Meniul profesorului (Theses)

După cum am amintit și ulterior, odată ce o lucrare de licență este asignată unui student și aceasta apare în lista de lucrări de licență asignate studenților, profesorul coordonator poate adăuga obiective care trebuie atinse pe durata elaborării lucrării de licență prin intermediul următorilor pași:

1. Vizualizează lucrarea de licență
2. Stabilește dacă obiectivele implicite se aplică în acest caz, iar dacă nu poate să steargă acele obiective care nu se aplică în cazul lucrării de licență aleasă
3. Crează noi obiective, stabilindu-le și ordinea în care acestea trebuie atinse
4. Stabilește împreună cu studentul termenele limită pentru atingerea obiectivelor

În imaginea următoare este afișat formularul de adăugare / modificare a obiectivelor pentru o lucrare de licență:

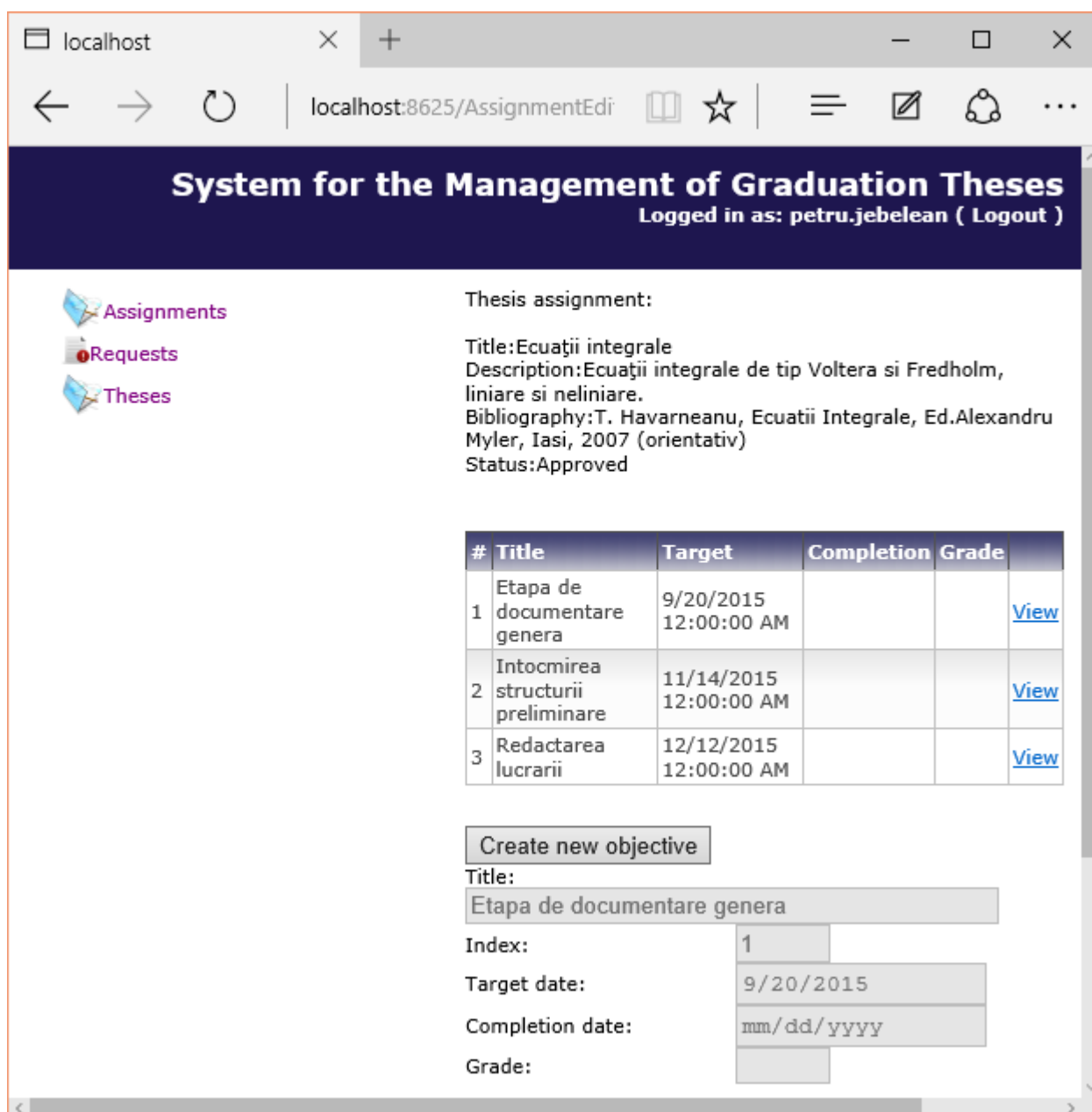


Figure 17: Adăugarea și modificarea obiectivelor

### Rolul de student

După cum se poate observa și în imaginea următoare corespunzătoare meniului utilizatorului cu rolul de student, un astfel de utilizator are cele mai puține drepturi.

Un student poate să vadă lista completă de lucrări de licență și poate să aleagă o lucrare de licență din aceeași listă. Odată ce o lucrare este aleasă va trebui să aștepte confirmarea pe care o va primi de la profesorul coordonator al lucrării de licență.

Imediat ce o astfel de lucrare de licență este aprobată studentul poate să stabilească împreună cu profesorul coordonator obiectivele care trebuie atinse pe parcursul elaborării lucrării de licență. De asemenea profesorul poate să stabilească aceste obiective și ulterior, împreună cu studentul, să stabilească termenele limită pentru atingerea fiecărui obiectiv. Aceste obiective vor putea fi vizualizate de către student chiar imediat după ce acesta se autentifică. Pagina de „Assignment” este dedicată pentru a afișa toate informațiile de care are nevoie studentul pentru a elabora lucrarea de licență aleasă.

Meniul studentului se află în imaginea următoare:

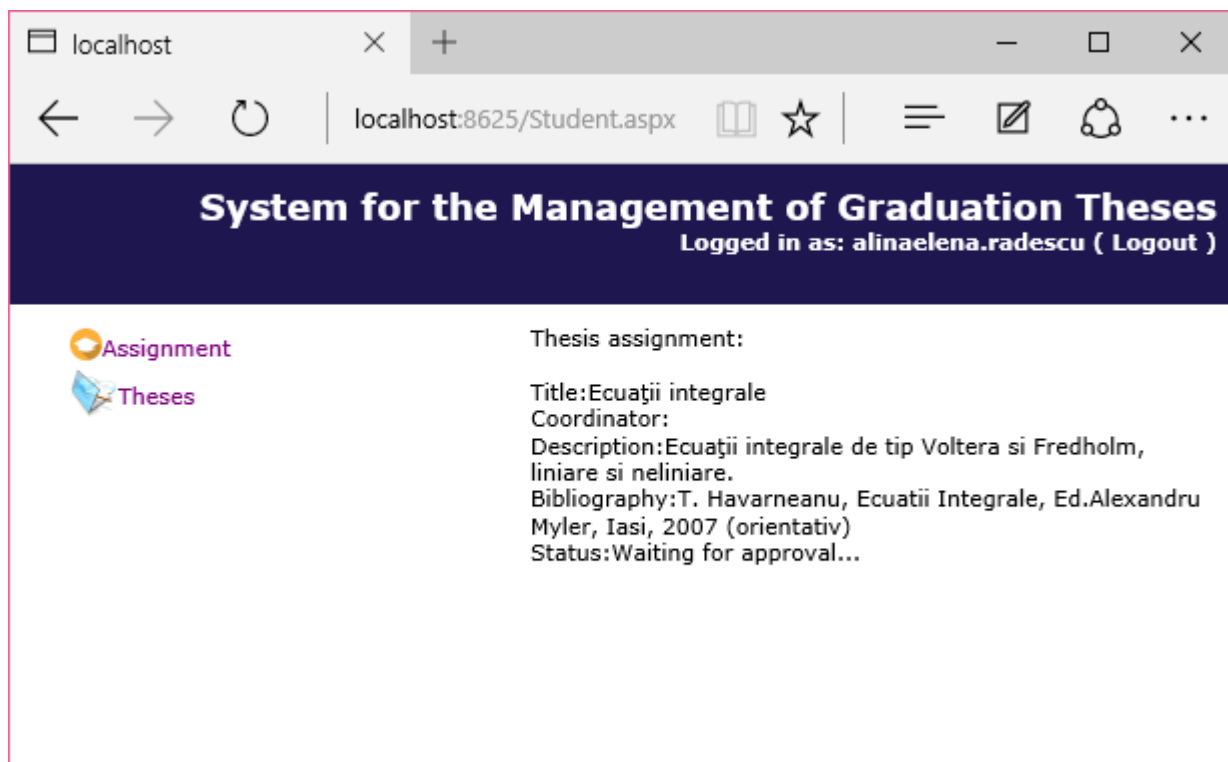


Figure 18: Meniul studentului

Odată ce un obiectiv este îndeplinit de către un student, profesorul va adăuga data la care acel obiectiv a fost atins și de asemenea va exista o etapă de evaluare a obiectivului respectiv în care profesorul îi va atribui un calificativ pentru realizarea obiectivului.

Acest proces este descris și într-o secțiune anterioară, aplicația oferind posibilitatea stocării datelor de ieșire ale fiecărui proces.

## Concluziile lucrării

Odată cu elaborarea acestei lucrări o serie de aspecte tehnice au fost atinse, astfel lucrarea a devenit interesantă mai ales din acest punct de vedere. De asemenea modelarea proceselor pentru ceea ce înseamnă gestiunea lucrărilor de licență a reprezentat partea cea mai importantă din elaborarea acestei lucrări.

Din punct de vedere tehnic, tehnologiile Microsoft oferă posibilitatea implementării de aplicații software rapid și eficient datorită documentației pe care Microsoft o pune la dispoziție, dar și datorită modului în care diferite tehnologii Microsoft pot fi folosite ca un tot-unitar.

Posibile îmbunătățiri ulterioare pot fi aduse aplicației software.

Procesul de identificare a nevoilor (Business Analysis) este necesar și foarte important în dezvoltarea unei astfel de aplicații care implică mai mulți actori în realizarea proceselor și a acțiunilor.

## Bibliografie

„Informatică de gestiune”, Lector Doctor Flavia Micota

„Business Analysis” - „Obținerea, analiza și documentarea cerințelor în proiecte”, [www.trilex.ro](http://www.trilex.ro)

[https://ro.wikipedia.org/wiki/Bază\\_de\\_date](https://ro.wikipedia.org/wiki/Bază_de_date)

<https://msdn.microsoft.com>

„All About Programming Language”, David Bolton

„Introducere în Programarea .Net Framework”, Inspector General M.E.C.T. Nușă Dumitriu-Lupan, profesor Rodica Pinte, profesor Adrian Niță, profesor Mioara Niță, profesor Cristina Sichim, profesor Nicolae Olăroiu, cadru didactic asociat Universitatea Politehnică Timișoara Mihai Tătăran, Developer Community Lead, Microsoft România Petru Jucovski, Team Lead Microsoft Student Partners Tudor-Ioan Salomie

„Integrarea sistemelor informatice .NET vs Java”, profesor Ilie Mihai Gabriel

<https://msdn.microsoft.com> - Microsoft Developer Network

# Contents

<b>Abstract</b>	<b>1</b>
<b>Capitolul 1</b>	<b>4</b>
<b>Descrierea entităților implicate în procesele implementate</b>	<b>4</b>
Definiția noțiunii de entitate . . . . .	4
Entitățile utilizatori și roluri . . . . .	4
Entitatea lucrare de licență . . . . .	5
Legături între entități . . . . .	5
<b>Capitolul 2</b>	<b>7</b>
<b>Modelarea și descrierea proceselor</b>	<b>7</b>
Definiții . . . . .	7
Descrierea proceselor . . . . .	8
<b>Capitolul 3</b>	<b>12</b>
<b>Tehnologii. Cadrul tehnic</b>	<b>12</b>
Evoluția tehnicilor de programare . . . . .	12
Programarea orientată obiect . . . . .	13
Limbajul de programare C# . . . . .	16
.NET Framework (Platforma .NET) . . . . .	21
ASP.NET . . . . .	22
Entity Framework . . . . .	26
Microsoft SQL Server . . . . .	29
<b>Capitolul 4</b>	<b>32</b>
<b>Descrierea Aplicației Software</b>	<b>32</b>
Introducere . . . . .	32
Baza de Date . . . . .	32
Administrarea datelor folosind aplicația . . . . .	35
<b>Concluziile lucrării</b>	<b>43</b>
<b>Bibliografie</b>	<b>44</b>