



Universitatea  
Transilvania  
din Brașov  
**FACULTATEA DE MATEMATICĂ  
ȘI INFORMATICĂ**

Programul de studii:  
Informatică,  
Învățământ la distanță

# **Lucrare de licență**

## **Aplicație de Planificare a Studiului Individual**

Autor:

**Florin ASAVEI**

Coordonator științific:

**Lect. Univ. Dr. Răzvan BOCU**

Brașov, 2021





## Cuprins

Introducere .....	4
Scurtă descriere a temei.....	4
Motivația alegerii lucrării.....	4
Structura Lucrării.....	4
1. Abordarea temei.....	6
1.1 Scopul lucrării.....	6
1.2 Alte abordări în domeniu.....	6
1.3 Designul aplicației Web.....	9
1.3.1 Specificațiile unei aplicații web.....	9
1.3.2 Designul aplicației Learning Map.....	9
2. Managementul entităților și al identității .....	20
2.1. Modelarea Entităților .....	20
2.1.1 Prezentarea generală a entităților.....	20
2.1.2 Modelul Entitate-Relație.....	21
2.1.3 Modelul Entitate-Date.....	21
2.1.4 Diagramele de structură (statice) .....	23
2.1.5 Digramele de stare (dinamice).....	23
2.2. Faza de implementare a bazei de date.....	25
2.2.1 Abordarea code-first în proiectarea bazelor de date.....	25
2.2.2 Sistemul de migrări pentru versionarea bazei de date .....	26
2.2.3 Popularea bazei cu date initiale prin metoda de Seed .....	27
2.3. Managementul identităților și al utilizatorilor.....	27
2.3.1 Managementul identității și accesului (IAM).....	28
2.3.2 Autentificarea prin tokeni JWT.....	28
2.3.3 Microsoft Identity Platform.....	29
3. Grafuri aplicate în reprezentări vizuale.....	32
3.1 Concepte Generale.....	32



3.1.1 Tipuri de grafuri.....	32
3.1.2 Reprezentarea unui graf.....	33
3.2. Vizualizarea grafică a datelor.....	36
3.2.1 Alegerea unei biblioteci javaScript pentru vizualizarea datelor .....	37
3.2.2 Implementarea folosind ngx-graph.....	42
4. Arhitectura aplicației .....	43
4.1 Separarea Preocupărilor – SoC (Separation of Concerns).....	43
4.1.1 Modelul Client-Server ca și fundament al tehnologiilor web .....	43
4.1.2 Tipul de arhitectură MVC și variațiuni ale acestuia .....	44
4.1.3 Arhitectura stratificată .....	44
4.1.4 CQRS.....	45
4.2 Arhitectura REST pentru designul unui API .....	45
4.2.1. Constrainările REST.....	45
4.2.1.1 Modelul Client-Server în REST .....	46
4.2.2. API-uri HTTP ce implementează REST .....	47
4.2.3. CRUD folosind verbe HTTP.....	47
4.2.4. Statusuri de răspuns HTTP.....	48
4.2.5 Documentarea API-ului folosind OpenAPI.....	48
4.3 Modelul „Clean Architecture” definit de Uncle Bob.....	48
4.3.1. Concepte de Clean Code și bune practici.....	49
4.3.2. Implementarea arhitecturii “Clean Code” în .Net.....	49
4.4 Securitate .....	50
4.4.1. HTTPS.....	50
4.4.2. OpenId Connect și OAuth2 .....	53
5. Tehnologii folosite.....	56
5.1 .Net 5 .....	56
5.1.1 Ecosistemul .Net.....	56
5.1.2. ASP.Net Core WebAPI.....	57
5.1.3 EntityFramework Core.....	58



5.2. Angular.....	58
5.2.1. Ecosistemul javaScript/TypeScript.....	59
5.2.2. Versiunea curentă a framework-ului Angular.....	59
5.2.3. Angular CLI.....	60
5.3. Microsoft SQL Server .....	60
5.3.1. Avantajele bazelor de date relaționale.....	60
5.3.2. Microsoft SQL Server 2019.....	61
5.4. Microsoft Azure .....	61
5.4.1. Concepte generale .....	61
5.4.2 IaaS, PaaS și SaaS .....	62
5.4.2. Servicii oferite de Azure.....	63
6. Instructiuni de utilizare ale aplicăiei Learning Map.....	68
6.1 Managementul utilizatorilor.....	68
6.2. Managementul resurselor (trasee, capitole, resurse) .....	69
6.3. Logarea progresului de către student .....	70
6.4. Vizualizarea datelor.....	70
6.5. Instructiuni de depanare .....	71
7. Concluzii și perspective de viitor .....	72
7.1. Concluzii.....	72
7.2 Posibilități de extindere.....	72
Listă de figuri.....	73
Listă de tabele .....	74
Bibliografie și Webografie .....	75



## Introducere

### Scurtă descriere a temei

În această lucrare, am documentat realizarea unei aplicații web de planificare a studiului individual și a traseelor educaționale, folosind o arhitectură de tip client-server cu tehnologii moderne (Angular și .Net 5) și unelte pentru vizualizarea datelor care utilizează concepte din teoria grafurilor. Aplicația este publicată în Cloud folosind Microsoft Azure.

Premisa aplicației este următoarea: un student își dorește să parcurgă un plan de învățare într-o formă cât mai lizibilă și mai atractivă, urmând un traseu configurabil care să-l ajute să-și dea seama în orice moment unde se află și cât mai are de studiat până când ajunge la obiectivul stabilit.

Cu ajutorul aplicației Learning Map putem crea trasee ce conțin capitulo și mai apoi resurse educaționale, dar putem defini și dependințe între acestea.

### Motivația alegerii lucrării

Cred că mulți studenți s-au simțit uneori "pierduți pe drum", confuzi sau copleșiți de volumul de informație pe care trebuie să-l parcurgă. Personal, m-am regăsit de multe ori în această ipostază; de aceea, motivația de a crea o soluție informatică pentru această problemă este și una intrinsecă. Apoi, următorul motiv ar fi dorința de a profundiza tehnologiile web și arhitectura software modernă; de aceea am ales ca și infrastructură platforma Microsoft pentru partea de back-end și framework-ul Angular pentru partea de front-end; acestea fiind unele dintre cele mai populare tehnologii din domeniu; ambele fiind într-o continuă expansiune.

Am urmat o arhitectură curată, separând atât cele două componente, pentru a putea explora mai multe tehnologii și pentru a permite dezvoltarea individuală a lor. Tot despre arhitectură, pot spune că există separare atât între back-end și front-end cât și în cadrul modulelor, între nivelele lor.

### Structura Lucrării

Lucrarea este împărțită în 7 capitulo care conțin în mare următoarele informații:

- Capitolul 1: **Abordarea temei**, descrie pe scurt ideea aplicației, conceptele generale și metodologia de lucru împreună cu tehnica de versionare și DevOps.
- Capitolul 2: **Managementul entităților și al identității**, detaliază modelarea bazei de date, prezintă diagramele de structură și de comportament, managementul de utilizatori și rolurile acestora dar și cum funcționează partea de autentificare și autorizare.



- Capitolul 3: **Grafuri aplicate și reprezentări vizuale**, este capitolul ce conține aportul personal cel mai mare, aici fiind vorba despre esența rezolvării problemei propuse.
- Capitolul 4: **Arhitectura aplicație**, prezintă partea tehnică în detaliu și descrie cum interacționează toate componentele sistemului. Am detaliat modelul RESTfull, arhitectura Clean Code dar și elemente de securitate și bune practici.
- Capitolul 5: **Tehnologii folosite**, este probabil cea mai interesantă parte a lucrării, deoarece descrie partea practică, conținând exemple și soluții. Aici am insistat pe tehnologiile Cloud deoarece mi se pare un domeniu de mare actualitate și interes.
- Capitolul 6: **Instrucțiuni de utilizare**, descrie pe scurt principalele cazuri de utilizare și funcționalități ale aplicației.
- Capitolul 7: **Concluzii și perspective de viitor**, reprezintă rezumatul lucrării, împreună cu elementele esențiale, concluziile finale și posibilitățile de extindere.



## 1. Abordarea temei

În societatea modernă, cu provocările ei specifice, procesul de învățare este unul continuu și inevitabil, vedem asta în din ce în ce mai multe profesii, de la medicină până la diverse tipuri de inginerie. Domenii precum Informatică ne solicită să fim mereu la curent și să ne îmbunătățim cunoștințele periodic. De aceea, în această lucrare am ales domeniul educației în sine, mai exact planificarea unui proces de învățare.

Un alt aspect, tot al societății moderne, este că datorită volumului imens de informații, devine foarte ușor să ne pierdem concentrarea și tot mai greu să ținem minte ce avem de făcut, de aceea au apărut o gamă tot mai largă de aplicații pentru îmbunătățirea productivității, de la calendar până la liste inteligente. În esență, aplicația este o combinație între un calendar de activități, o listă de sarcini și un graf orientat.

### 1.1 Scopul lucrării

Lucrarea prezentă are ca scop crearea unei aplicații web care să ajute la gestionarea și urmărirea unui traseu educațional organizat în capitole și resurse educaționale în scopul vizualizării cât mai clare a evoluției unui student.

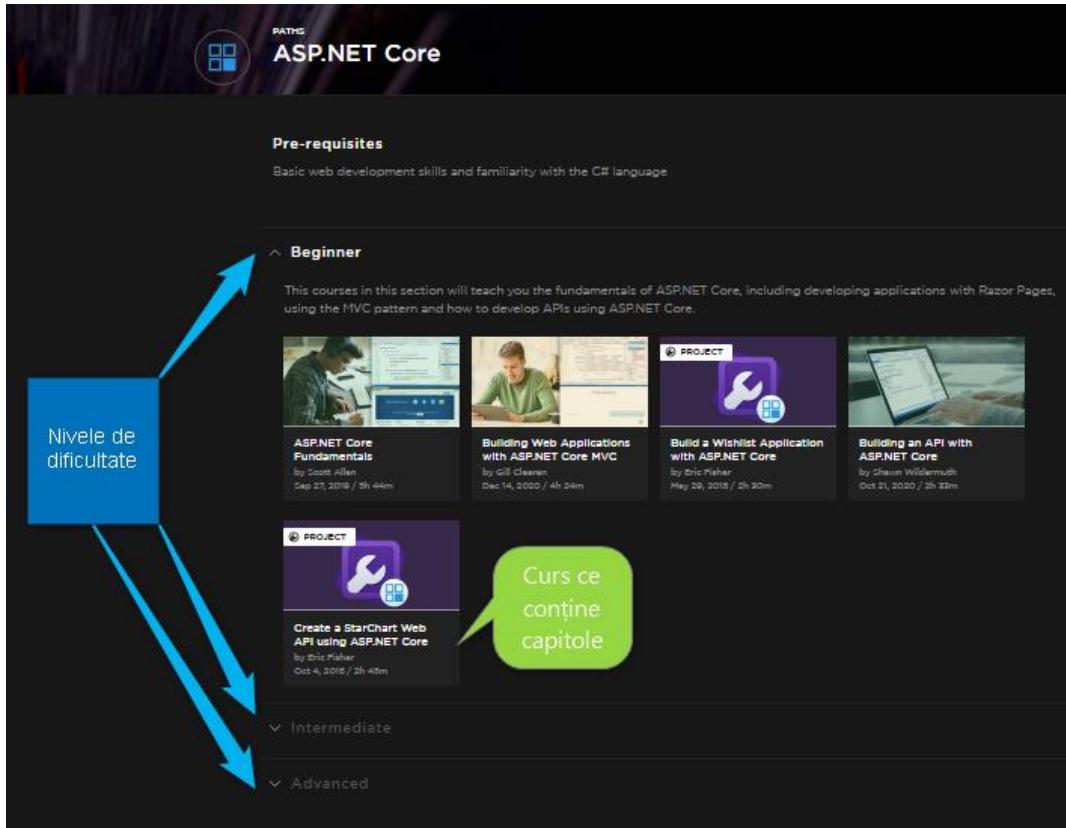
Pentru reprezentarea traseelor s-au folosit structuri de date de tip graf și concepte de vizualizare a datelor utilizând librăria ngx-graph. Această funcționalitate fiind, din punctul meu de vedere, punctul forte al aplicației și conține contribuția personală cea mai mare. De asemenea, sunt descrise aspectele de arhitectură și tehnologiile folosite..

### 1.2 Alte abordări în domeniu

Conceptul de productivitate în procesul de educație nu este unul nou, având în vedere că domeniul IT devine tot mai atractiv (chiar și pentru reconversii profesionale), majoritatea platformelor au un mod plăcut de a-și prezenta produsele, alcătuite în general din cursuri. Un concept foarte la modă este acela de "gamification" și mai ales în învățământ, se vorbește tot mai des despre educația prin joacă. Si, în orice joc, trebuie să știi unde te află. Un alt domeniu ce ia amploare este acela al științei datelor (data science) din categoria cărora face parte și vizualizarea datelor. Aici sunt foarte multe platforme și unelte care pot reprezenta datele într-un mod vizual foarte atractiv. Scopul acestei lucrări nu este să implementeze de la zero o astfel de platformă ci să folosească tehnologii consacrate pentru a reprezenta datele.

Sursele principale de inspirație au fost platformele de e-Learning și cursuri online unde am observat că și se prezintă un traseu recomandat. Însă nu toate platformele au acest concept, platforma Udemy lasă la latitudinea utilizatorului ce cursuri dorește să urmeze.

Majoritatea furnizorilor de cursuri și tutoriale au un astfel de traseu recomandat pe diverse tematici. Platforma PluralSight spre exemplu, promovează conceptul de "Learning Path" unde avem diverse capituloare care conțin cursuri recomandate. Spre exemplu, pentru cineva care își dorește să devină programator în ASP.NET Core, ei recomandă un traseul reprezentat astfel:



Figură 1 – Exemplu platformă e-learning

Evident că cei de la PluralSight vor recomanda doar cursurile lor, aceste trasee sunt predefinite și nu pot fi modificate de către utilizatori.

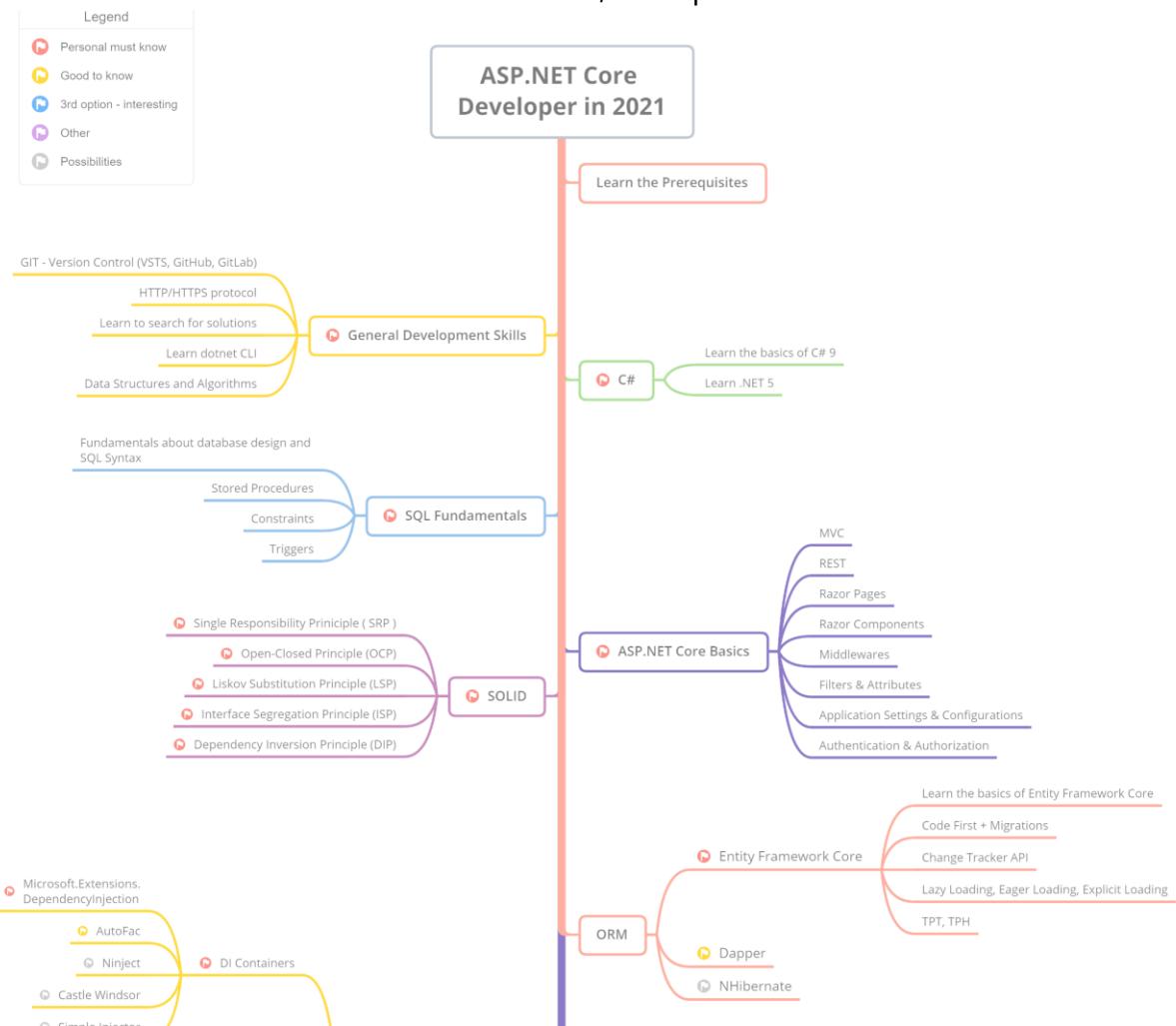
Un alt exemplu de platformă ce implementează conceptul de "Learning Path" într-un mod mai flexibil este platforma recent lansată Microsoft Learn, care oferă și posibilitatea de personalizare a traseului și sugerează diverse alte cursuri similare. Avantajul față de PluralSight este că Microsoft oferă aceste resurse în mod gratuit. Însă nici aceștia nu oferă o vedere de ansamblu suficient de detaliată, lipsind elementul de vizualizare a datelor.

Ca și inspirație principală, am găsit o pagină web în care erau prezentate principalele resurse pe care ar trebui să le parcurgă un programator web care vrea să stăpânească Framework-ul ASP.Net Core. Această pagină web, postată public pe GitHub avea în Decembrie 2020 peste 5000 de aprecieri (stele GitHub), iar acum (Mai 2021) are peste 6200, semn că din ce în ce mai mulți oameni se gândesc să abordeze această cale spre a deveni programator web .Net.

Pagina cu pricina se poate găsi aici: <https://github.com/MoienTajik/AspNetCore-Developer-Roadmap>



Autorul paginii a alcătuit o listă statică de resurse recomandate pe care apoi le-a aranjat într-o formă grafică sub formă de ierarhie. Deși această poză este una statică, ideea de a genera un astfel de traseu folosind date dinamice, mi se pare una de mare valoare.



Figură 2 – Exemplu traseu educațional

O altă abordare, care este de fapt o ocolire a problemei, ar fi să folosim o aplicație generică pe care să o modelă la nevoile noastre. În cazul în care am dori să ne creăm singuri o astfel de listă de cursuri, cărți și diverse alte resurse, pe lângă clasicele bookmark-uri în browser, am putea folosi aplicații de tipul "To Do" care ne permit să ne creăm liste, subliste și să bifăm înregistrările din listă. Piața este deja supra-saturată de astfel de aplicații, de la liste de cumpărături inteligente până la aplicații ce ne pun să bifăm dacă am băut sau nu apă. Un exemplu este Microsoft Todo, o aplicație generică ce poate fi folosită pentru scopuri diverse, inclusiv pentru organizarea resurselor educationale.

Problema cu aceste aplicații de tip "To do" este că sunt multe prea generale și nu ne satisfac cerințe precum: crearea de dependințe între înregistrările listei, adăugarea de câmpuri definite de utilizator, cum ar fi: nivel de dificultate, timpul estimat, etc. Aceste cerințe ne-ar duce mai degrabă spre o zonă de aplicații business pentru managementul proiectului de tipul JIRA sau Asana care ar fi prea greu de configurat și adaptat pentru nevoie noastră, depășind un grad de complexitate acceptabil.



Lipsa unei platforme care să ne permită să ne organizăm resursele educaționale de pe diverse platforme, m-a determinat să dezvolt propria aplicație care să rezolve această problemă.

### 1.3 Designul aplicației Web

Primul pas în cadrul oricărei aplicații software este cel de stabilire a funcționalităților și de documentație inițială. Vom stabili criteriile după care vom dezvolta Produsul Minim Viabil – MVP (Minimum Viable Product). Conceptul de MVP a fost popularizat de către Eric Ries în cartea sa Lean Startup [8]. Scopul unui MVP este să valideze că soluția noastră funcționează într-un timp cât mi scurt.

Practic un MVP este o soluție intermediară fiind prima versiune ce se poate testa într-un mediu real de producție.

Puteam spune că aplicația cu pricina, Learning Map este de fapt un MVP deoarece este un prototip funcțional care rezolvă problema dată cu un minim de efort, fără să fie însă la un nivel de producție care să concureze cu aplicații existente.

#### 1.3.1 Specificațiile unei aplicații web

Atunci când creăm un produs informatic, acesta trebuie să fie însotit și de un minim de instrucțiuni. Însă această parte de scriere a specificațiilor este un mare consumator de timp și conform principiilor Agile (descrise în capitolul 1.3.2.1 Managementul Proiectului și planificarea sarcinilor), este mai important să livrăm un produs funcțional decât o documentație stufoasă. Ba mai mult, tendința actuală este de a crea aplicații care să fie cât mai intuitive și care să nu aibă nevoie de prea multe explicații. Se spune că o interfață web este ca o glumă, dacă trebuie explicată, înseamnă că nu e suficient de bună.

Crearea unui MVP urmează de obicei o serie de etape care s-au dovedit eficiente [8]:

1. Analiza de piață
2. Înțelegerea utilizatorilor finali
3. Definirea produsului final (brainstorming)
4. Definirea funcționalităților pentru MVP

#### 1.3.2 Designul aplicației Learning Map

Ca și public țintă, ne adresăm în primul rând studenților dar și instituțiilor sau organizațiilor educaționale. Un student poate fi în același timp și profesor (folosind autorizarea pe bază de roluri) și își poate organiza singur resursele educaționale, apoi își poate înregistra mai departe progresul per fiecare resursă definită. Actorul principal în acest sistem este studentul.

În același timp, aplicația trebuie să fie scalabilă, extensibilă, modulară și portabilă.



Astfel, putem spune că funcționalităile de bază ale aplicației sunt:

- Gestionarea de utilizatori și roluri
- Gestionarea de resurse (trasee, captoare, URL-uri)
- Actualizarea progresului per resurse
- Vizualizarea datelor (legăturile dintre captoare) și al progresului

Atunci când vorbim de design, ne referim atât la partea de grafică dar și la partea funcțională a aplicației.

Ca și tip de aplicație web, Learning Map se încadrează în categoria SPA (Single Page Application), caracteristica principală a acestora fiind că nu necesită reîncărcarea completă a paginii. Aplicațiile SPA, odată încărcate în browser, vor înlocui bucăți de conținut pe măsură ce au nevoie de el, oferind o experiență fluidă și rapidă.

Ca și design grafic, am folosit elemente din librăria Bootstrap 4 împreună cu iconițe din setul FontAwesome. Componentele au un aspect curat și cu colțuri drepte, nerotunjite, respectând principiile de material design introduse de cei de la Google. Pe scurt aceste principii de design recomandă următoarele practici [13]:

- Folosirea culorilor simple, dintr-o gamă cât mai mică
- Evitarea elementelor ne-esențiale cum ar fi umbre excesive sau animații obosite
- Adaptarea în funcție de dimensiunile ecranului (responsive)
- Folosirea spațierii pentru a scoate în evidență conținutul
- Folosirea marginilor drepte, tăioase, pentru a oferi un aspect minimalist, industrial

### 1.3.2.1 Managementul Proiectului și planificarea sarcinilor

*"If you fail to plan, you are planning to fail!"*

- Benjamin Franklin

Planificarea proiectului este utilă chiar și atunci când dezvoltarea este făcută de către o singură persoană. De-a lungul timpului au existat mai multe metodologii de dezvoltare a softului (MDS). S-a dovedit că modul de lucru iterativ-incremental este preferat de către specialiștii în I.S. (Inginerie Software). [2]

Metodologia AGILE este una dintre cele mai populare abordări din domeniu. Principiile de bază ale acestei metodologii au fost popularizate în cadrul unei întâlniri din 2001 care a avut loc la un resort montan din Utah, SUA. La acea întâlnire au participat 17 programatori foarte populari în acel moment (și poate și mai populari în zilele noastre) care au semnat "Manifestul pentru dezvoltarea agilă de software". [12]

Acest manifest este scurt însă foarte relevant și acum, peste mai bine de 20 de ani.

Printre cei 17 programatori ce au semnat acest manifest îl regăsim pe:

- Robert C. Martin (Uncle Bob) – Autorul unor cărți precum "Clean Code" și „Clean Architecture”(un model arhitectural pe care l-am urmat pentru realizarea Web API-ului acestei aplicații)

- Ward Cunningham, Kent Beck și Ron Jeffries – Autorii metodologiei Extreme Programming

- Martin Fowler – Un autor foarte respectat, specializat în subiecte precum POO, UML

- Jeff Sutherland și Ken Schwaber – creatorii metodologiei SCRUM

Statistic, un proiect care folosește metodologii Agile are șanse mai mari de reușită decât atunci când folosim Waterfall. În general aceste metodologii se folosesc când lucrăm în echipe însă nu ne oprește nimeni să folosim aceste practici și când lucrăm la propriul produs de unii singuri.

### 1.3.2.2 Metodologia AGILE

Metodologia Agile reprezintă o abordare de management de proiect care permite unei echipe dintr-o organizație, să se concentreze asupra obiectivelor sale. [17]

Agile este o practică ce promovează iterată continuă a dezvoltării și testării pe tot parcursul ciclului de viață al dezvoltării software al unui proiect, unde atât activitățile de dezvoltare, cât și activitățile de testare sunt concurente, spre deosebire de modelul Cascadă (Waterfall). Agile, utilizează succesiuni de lucru iterative și incrementale, care sunt denumite și sprinturi. Metoda de dezvoltare software Agile încurajează răspunsuri flexibile la schimbare și este una dintre cele mai simple și eficiente procese pentru a transforma o viziune pentru o nevoie de afaceri în soluții software. Agile descrie abordări de dezvoltare software unde se utilizează o planificare continuă, învățare, îmbunătățire, colaborare în echipă, dezvoltare evolutivă și livrare la timp.

În Agile, întâlnim câteva dintre următoarele abordări [17]:

- Metoda Agile propune o abordare incrementală și iterativă a proiectării software
- Procesul Agile este împărțit în modele individuale la care lucrează designerii
- Clientul are oportunitatea de a vedea produsul din timp și frecvent astfel încât poate aduce decizii și modificări noi la proiect
- Proiectele mici pot fi implementate foarte repede, pe când pentru proiecte mari, este dificil de estimat timpul de dezvoltare
- Erorile pot fi remediate în mijlocul proiectelor
- Procesul de dezvoltare este iterativ, iar proiectul este executat în intervale scurte de 2-4 săptămâni și nu necesită o planificare amplă
- Fiecare interatăie are propria fază de testare. Permite executarea testării de regresie de fiecare dată când sunt lansate noi funcționalități.



- În testarea Agile, la sfârșitul unei iterării, caracteristicile produsului care pot fi livrate sunt trimise clientului, iar funcțiile noi sunt disponibile imediat. Este util să menține o bună legătură cu clientul
- Testările și programatorii lucrează împreună
- La sfârșitul fiecărui sprint, se validează funcționalitățile din punct de vedere al utilizatorului

În aplicarea metodei Agile, într-un proiect, se țin cont de mai multe aspecte, pentru a știi dacă această metodă este corespunzătoare. Se iau în calcul specificațiile proiectului, dimensiunea și designul proiectului. În Agile specificațiile se pot modifica în funcție de cerințele clientului, cerințele nu sunt întotdeauna clare și chiar dacă arhitectura proiectului este realizată conform cerințelor actuale, ea este concepută pentru a fi flexibilă. În momentul în care există un proiect mic, este necesară formarea unei echipe mici.

De asemenea, se mai iau în calcul planificarea proiectului care este internalizată și controlul asupra proiectului este unul calitativ.

Din punct de vedere al tipului de client și al programatorului, metodologia Agile, urmărește o colaborare ușoară din ambele părți, clienți care sunt dispuși să fie prezenti și să da un feedback la fiecare etapă a proiectului și programatori analitici și organizați.

Nu în ultimul rând, trebuie luat în calcul și riscul acestei metode, unde, șansele de apariție a riscurilor necunoscute sunt mai mari, ceea ce poate avea un impact major în proiect, însă, schimbările, reautorizările într-un proiect, nu sunt costisitoare, fără de o metodă non-Agile.

Printre avantajele metodei Agile se pot număra:

- Livrarea de software este continuă
- Testarea se poate face în paralel cu dezvoltarea
- Clienții sunt mulțumiți, deoarece după fiecare sprint, caracteristicile funcționale sunt livrate către ei și astfel pot vedea dacă așteptările le-au fost îndeplinite
- Interacțiunile zilnice sunt necesare între clienți și dezvoltatori
- În cazul în care clientul dorește o modificare, aceasta poate fi inclusă în versiunea curentă a produsului, de asemenea, modificările cerințelor sunt acceptate chiar și în ultimele etape ale dezvoltării.

### 1.3.2.3 User Story-urile aplicației

Conceptul de "User Story" este unul fundamental în metodologia Agile. Un "User story" nu este altceva decât o definire informală a unei funcționalități a sistemului din punctul de vedere al unui utilizator. Pentru definirea lor, se folosesc limbajul natural, deci nici măcar pseudocod, un user story putând fi înțeles cu ușurință și de către persoane care nu au cunoștințe tehnice. Sunt necesare în schimb cunoștințe despre domeniul de aplicare al produsului software.



Un User Story ar trebui să aibă o structură de genul "Ca și ... ar trebui să .... pentru a ...". Ca și exemplu, atunci când am implementat funcționalitatea ca un student să poată înregistra progresul pe o resursă educațională am definit următorul "User Story".

Ca și utilizator având rolul de student ar trebui să:

- Văd resursele educaționale grupate în capitole

Pentru a putea să:

- Înregistrez progresul (între 0 și 100) pentru fiecare resursă
- Vizualizez și Actualizez oricând acest progres

Un user story nu trebuie confundat cu un caz de utilizare (use case). Cazurile de utilizare sunt unități de sine stătătoare, bine definite care definesc cerințele funcționale ale sistemului pe când un User Story se concentrează mai mult pe ceea ce își dorește utilizatorul să se întâmple. Ambele concepte sunt neutre din punct de vedere tehnologic și folosesc limbajul natural. User Story-urile sunt o evoluție a cazurilor de utilizare, apărute odată cu metodologia Agile.

În aplicația Learning Map, principalele "User Stories" sunt :

- Ca și Administrator, ar trebui să pot adăuga utilizatori și a le atribui roluri, pentru a le putea oferi acces în sistem.
- Ca și Utilizator Înregistrat, ar trebui să-mi pot gestiona singur detaliile contului și să-mi actualizez numele, data nașterii și numărul de telefon.
- Ca și Utilizator Înregistrat ar trebui să vă doar zonele care mă privesc, conform rolurilor mele, pentru a evita acțiuni nepermise sau nedorite.
- Ca și Profesor, ar trebui să pot gestiona traseele, capitolele și resursele educaționale pentru a le distribui către studenți.
- Ca și Profesor, ar trebui să pot gestiona dependințele dintre capitole pentru a genera reprezentări vizuale.
- Ca și Student, ar trebui să văd resursele educaționale definite de profesor și să actualizez progresul.
- Ca și Student, ar trebui să văd progresul meu într-o formă cât mai plăcută pentru a ști exact unde mă aflu în planul de învățare.
- Ca și Profesor, ar trebui să văd progresul studentilor pentru a-l analiza.

Nu este necesar să definim toate User Story-urile de la început, acestea se pot defini și pe parcurs. Singura regulă este că dacă un user story este deja definit și adăugat într-un sprint, acesta nu are voie să se mai modifice până la următoare iteratie, deoarece riscăm să dezechilibram sistemul.

Regula generală a metodologiei Agile este că nimic nu e bătut în cuie, totul se poate schimba în funcție de cerințe.



Un exemplu de User Story definit folosind platforma Azure DevOPS:

USER STORY 18

18 Update Profil utilizator

Unassigned 0 comments Add tag

State: New Area: learning-map  
Reason: New Iteration: learning-map\Sprint 1

Description

Ca și utilizator înregistrat,  
Ar trebui să pot să completez numele meu și data de naștere.  
Opentru a afișa numele propriu în meniul de navigare (optional și pe pagina de pornire).  
Iar data nașterii va fi utilizată pentru statistici.

Learning Map Home Learn Map preview Map progress Manage Learning Paths Manage Users API developer@localhost Logout

Afisează Numele Propriu în loc de email

Figură 3 – User Story în Azure DevOps

#### 1.3.2.4 Metodologia de lucru și versionarea

Pornind de la User Story-urile definite în secțiunea precedentă, vom începe să construim funcționalitățile aplicației. Ne propunem să avem mereu o versiune stabilă a aplicației, chiar dacă nu este finală, dar la care să putem adăuga oricând funcționalități noi, în funcție de nevoi (care se pot schimba pe parcurs). Motivul pentru care ne dorim să avem mereu o versiune funcționabilă publicată pe un server anume, este să nu blocăm dinamica echipei. Având această versiune funcțională, se pot face teste manuale asupra versiunii curente în timp ce se lucrează la următoarea versiune.

În orice sistem informatic, versionarea este esențială, există numeroase sisteme de versionarea care au evoluat pe parcursul timpului. Chiar și atunci când lucrăm singuri la un proiect, este de mare ajutor să avem un istoric al schimbărilor și să putem comuta cu ușurință de la o versiune la alta.

#### Ce este un VCS (Version Control System)

De-a lungul istoriei recente a tehnologiei, au fost mai multe sisteme de versionarea menite să faciliteze colaborarea și menținerea unui istoric. De departe, cel mai folosit în acest moment este sistemul GIT, atât pentru proiecte mici cât și pentru proiecte foarte mari, atât pentru proiecte personale, open-source cât și pentru proiecte personale.

Conceptul principal al lui VCS este cel de "repository", care nu e altceva decât un container unde putem salva fișiere text sau binare. Un repository nu ne limitează neapărat la a stoca



doar bucăți de cod, însă este de preferat să nu îl aglomerăm cu fișiere inutile. Prin analogie, un repository e un fel de Google Drive pentru codul nostru.

Principalele avantaje ale folosirii unui VCS sunt:

- Păstrează istoricul modificărilor pentru a reveni la versiuni anterioare și pentru a inspecta când și cine a modificat o bucată de cod
- Poate crea branch-uri ce conțin versiuni paralele, acestea putând oricând să fie integrate (combinată) folosind procesul de "merge".

Practic există două categorii de sisteme de versionare [5]

- Sisteme de Versionare Centralizate (CVCS – Centralized Version Control System)
  - ex: SVN, CVS
- Sisteme de Versionare Distribuite (DVCS - Distributed Version Control System)
  - ex: GIT, Mercurial

Deosebirea principală între aceste două tipuri de sisteme este că în cazul CVCS avem un server central care va conține versiunea de referință a codului pe care trebuie să o sincronizăm periodic, practic dacă se întâmplă ceva cu acel server, va fi dificil de refăcut tot istoricul modificărilor; iar într-un DVCS toți utilizatorii au o copie locală a serverului de versionare și pot recrea oricând întreg sistemul.

Avantajele folosirii unui DVCS față de un CVCS sunt:

- mult mai ușor de lucrat cu branch-uri
- funcționează și offline, aplicarea modificărilor făcându-se în 2 pași: comit și push
- considerabil mai rapid

GIT este un program gratuit și open-source, introdus prima dată în kernel-ul Linux. Acest program este scris în C pentru a beneficia de maximul posibil de performanță. Practic, putem instala acest program gratuit pe calculatoarele noastre și ne putem crea un repository (sau mai multe) local. Atunci când instalăm Visual Studio, acesta va instala automat și programul GIT, însă îl putem instala și separat de Visual Studio.

Ca și mod de lucru, git se poate utiliza din linia de comandă însă acestă practică este întâlnită de obicei la programatorii mai experimentați. În genera, poate editoarele, inclusiv Visual Studio, au instrumente vizuale pentru a chema comenzi GIT: commit, pull, push, merge, etc.

Pentru a putea fi folosit online, GIT este găzduit de diversi furnizori, principali fiind: GitHub, BitBucket, GitLab și Azure Repos.

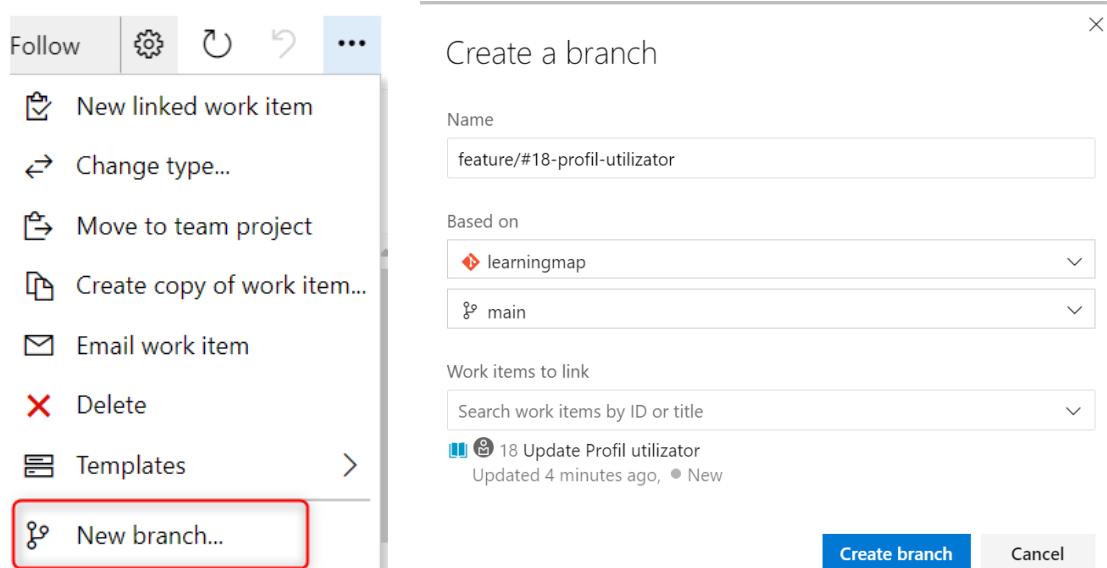
Am ales Azure Repos deoarece se integrează cel mai bine cu produsele Microsoft și este potrivit pentru proiecte private.

De asemenea, Azure Repos oferă multe funcționalități gratuite chiar și pentru proiecte private, cum ar fi: politici de branch-ing, politici de securitate, verificări de merge, etc.



Pornind de la User Story-ul cu Id-ul #18 (Update Profil utilizator), vom crea un branch nou, pornind din branch-ul master.

Mulțumită integrărilor excelente oferite de platforma Azure DevOPS, putem face acest lucru chiar din meniul asociat cardului de User Story:



Figură 4 - Creare branch

### 1.3.2.5 GitHub Flow

Atunci când lucrăm cu GIT este bine să respectăm anumite metodologii de lucru care s-au dovedit a fi de succes de-a lungul timpului. Am putea bineînțeles, să folosim GIT-ul cu un singur branch, însă pentru a beneficia de toate avantajele sale, se recomandă folosirea mai multor branch-uri.

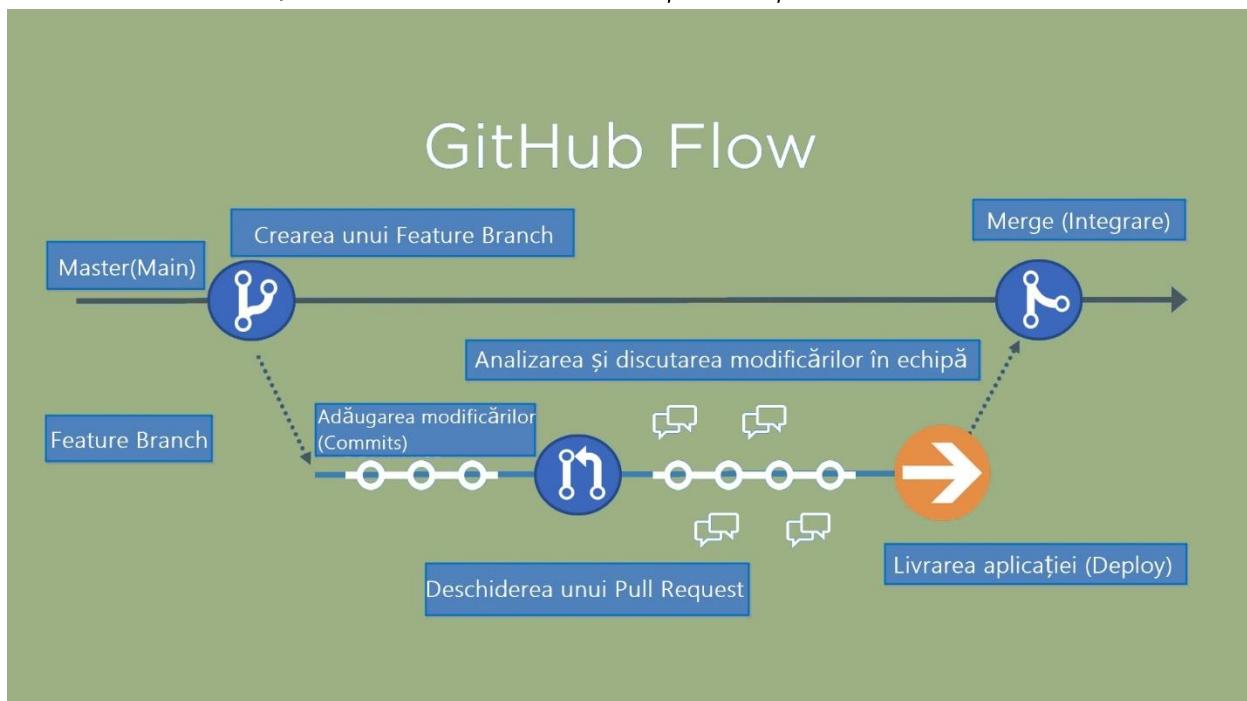
Deși nu folosim platforma GitHub, modul de lucru popularizat de aceștia, a devenit un standard în cadrul proiectelor versionate folosind GIT. Un flow similar, dar ceva mai complicat este Git Flow, care implică existența mai multor branch-uri, cum ar fi :

- Branch de dezvoltare
- Branch de verificare a calității (testare)
- Branch-uri de HotFix sau Bug-uri
- Branch-uri de Pre-Release

Evident, un astfel de sistem de branch-ing este potrivit pentru proiecte mai mari în care colaborează o echipă mai numeroasă de dezvoltatori. Pentru cazul aplicație noastre, GitHub Flow este cel mai potrivit.

Acest flow este unul simplu, pentru dezvoltare rapidă, unde au loc publicări regulate ale codului sursă în producție. Se folosesc două categorii mari de branch-uri:

- Branch-ul “main” (sau “master”) ce conține codul stabil, aflat în producție
- Branch-urile paralele de dezvoltare a funcționalităților noi (“feature branch”)



Figură 5 - GitHub Flow

În metodologia aleasă, fiecare User Story nou, va fi dezvoltat pe un feature branch separat. Mai mult, am limitat ca modificațile pe branch-ul “main” să se facă doar prin Pull Request și merge-urile să fie de tip Squash. Mai exact, atunci când se va realiza merge-ul, toate commit-urile din branch-ul intermediu vor fi comasate într-unul singur, pentru a crea istoric cât mai curat.

#### 1.3.2.5.1 Ce este un Pull Request

Un Pull Request este un mecanism prin care dezvoltatorii anunță echipa că au terminat o funcționalitate. Dar un Pull Request este mai mult decât o simplă notificare, este în primul rând un forum de discuție al dezvoltatorilor unde se analizează funcționalitatea ce urmează să fie integrată. [5]

Acest mecanism nu este unul integrat în programul GIT ci este o funcționalitate extra, oferită de platformele ce găzduiesc aceste repository-uri online.

De asemenea, Pull Request-ul este metoda recomandată prin care se face merge dintr-un branch în altul, altfel spus, un Pull Request nu e altceva decât o cerere de acceptare a modificaților făcute asupra codului Sursă. Aceste mecanisme sunt adesea folosite în procesele specifice DevOps, după cum vom vedea în secțiunea următoare.

Pentru nu a polua directorul de git cu branch-uri redundante, se recomandă ștergerea branch-ului intermediu dinspre care s-a pornit Pull Request-ul.



În cazul în care PR-ul nu se acceptă (din diverse motive), modificările nu sunt aduse din branch-ul sursă în branch-ul destinație.

### 1.3.2.6 DevOPS

În industria IT modernă, termenul de DevOps a devenit foarte popular, mulți spun că acesta este viitorul în ceea ce privește modalitatea de a organiza activitatea la locul de muncă. DevOPS este o combinație de filozofii, practici și instrumente care contribuie la creșterea productivității unei echipe și ajută atât la livrarea mai rapidă a aplicației dar și la creșterea calității.

Istoric vorbind, DevOPS este urmașul metodologiei Agile. Progresele din ultimul deceniu au dus la nevoia unei abordări de ansamblu a ciclului de viață a unui produs software. Putem să spunem că DevOPS este o mentalitate în IT care încurajează comunicarea, colaborarea, automatizarea și integrarea între programatori și partea operațională pentru a îmbunătăți procesul de livrare din punct de vedere al timpului dar și calitativ.

#### 1.3.2.6.1 Terminologia DevOPS

În DevOPS folosim conceptul de pipeline(conductă). Un pipeline este un set de procese automate care leagă partea de dezvoltare a aplicației( scris cod) cu partea de operațională (aplicația finală să ajungă la client). [14]

Integrarea Continuă (CI - Continous Integration) reprezintă un set de practici care încurajează dezvoltatorii să implementeze modificări mici și să facă commituri dese. Procesul de CI se ocupă de compilarea aplicației, executarea testelor automate și obținerea fișierelor executabile sau binare.

Livrarea Continuă (CD – Continous Delivery) se referă la faptul că, aplicația se va publica automat în mediile corespunzătoare (dezvoltare, testare sau producție). Aceste procese, dacă s-ar face manual, ar avea un cost operațional destul de mare, care crește pe măsură ce trebuie să publicăm aplicația la mai mulți clienți.

#### 1.3.2.6.2 Azure DevOPS

Pentru a completa suita Azure, am ales să integrez procesele de DevOps folosind platforma Azure Pipelines.

AzureDevOps folosește sintaxa YAML pentru a defini pipeline-urile, practic putem defini întregul proces de CI/CD folosind un singur pipeline. [14]

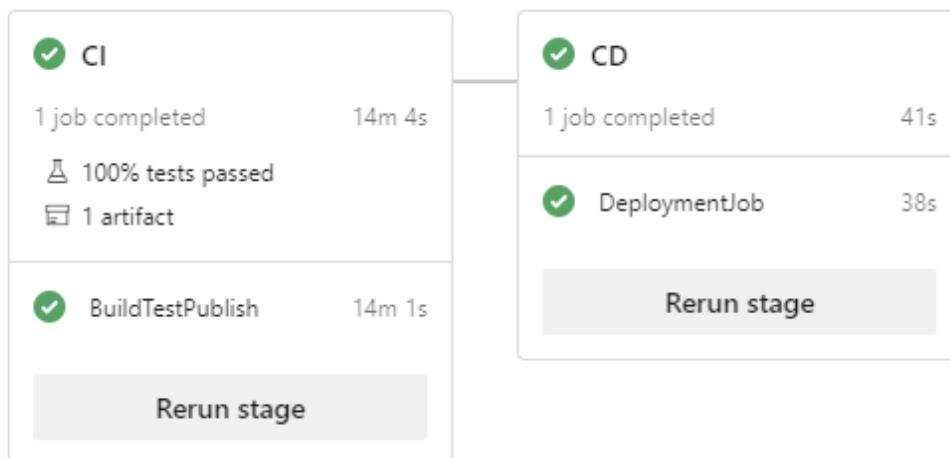


YAML, prescurtat de la "YAML Ain't Markup Language," este de fapt un superset peste JSON care a devenit foarte popular pentru fișierele de configurare. Spre deosebire de JSON, YAML se folosește de indentare pentru a delimita obiectele.

Fișierele YAML ce definesc acest pipeline se găsesc în codul sursă al proiectului (în directorul ci-cd)

Pipeline-uri are două etape (stages):

- Etapa CI (Continous Integration) care se ocupă de
  - Compilare
  - Rularea testelor automate
  - Generarea fișierelor ce trebuie să ajungă pe server (drop)
- Etapa CD (Continous Delivery) dependentă de etapa anterioară
  - Preia drop-ul generat de pasul anterior și îl publică în Azure



Figură 6 - Pipeline Azure DevOps



## 2. Managementul entităților și al identității

Orice sistem orientat pe obiect va conține mecanisme de gestionare a stării obiectelor precum și a interacțiunii dintre acestea. Folosind mecanisme precum moștenirea, polimorfismul, abstractizarea și încapsularea, limbajul C# ne permite să scriem un cod orientat pe obiect eficient și ușor de înțeles.

### 2.1. Modelarea Entităților

În orice tip de sistem informatic, dar mai ales în cadrul sistemelor orientate pe obiect, este foarte important să ne gândim ce tipuri de obiecte vom avea și ce relații vor exista între acestea. La nivel conceptual, termenul de obiect sau entitate, se referă la același lucru, adică la o abstractizare(reprezentare) a unui "ceva" din lumea reală.

Este important să avem aceste modele schițate înainte de a ne apuca să scriem cod, deși unii programatori sărăcăuți peste acest pas important. Bineînțeles, ele se pot modifica pe parcurs. Limbajul UML ne stă la dispoziție pentru a crea aceste modele conceptuale de la care vom porni implementarea propriu-zisă. Autorul Martin Fowler(unul dintre autorii care au pus fundamentele metodologiei Agile) susține că principalul rol al diagramelor UML este să schițeze și să dea o vedere de ansamblu, nu neapărat să descrie în detaliu structura sistemului, ele ar trebui să poată fi înțelese și de persoane mai puțin tehnice. Practic diagramele ne ajută să înțelegem cum arată și ce face sistemul, fără a citi codul.

#### 2.1.1 Prezentarea generală a entităților

Ca și tipuri de entități avem următoarele categorii de obiecte care alcătuiesc domeniul aplicației:

- Utilizator(entitate generică din platforma .Net) cu roluri asociate
- Entități de bază: Trasee, Capitole și Resurse educaționale
- Entități de legătură: conexiunile dintre capitole care vor alcătui graful
- Entități cu scop de logare: progresul fiecărui student per capitol

Entitățile de bază sunt cele asupra cărora se vor executa operațiunile CRUD (Create, Read, Update, Delete). Între aceste entități există relații structurale de tip părinte-copil. Traseul este entitatea rădăcină, nimic nu poate exista în afara unui traseu.

Utilizatorii sunt în esență tot niște simple entități care se pot adăuga, șterge sau modifica. Vom vedea exact cum se face acest lucru folosind platforma Microsoft Identity.

Entitatea de legătură definește de fapt dependințele unui capitol față de alte capitole. Aceste capitole sunt de fapt noduri incidente într-un graf.



Ultima entitate, cea de logare, este cea care înregistrează progresul unui utilizator cu rolul de student raportat la o entitate de bază de tip resursă educațională.

### 2.1.2 Modelul Entitate-Relație

De obicei, proiectarea unei baze de date implică 3 etape: stabilirea cerințelor, faza de proiectare și faza de implementare.

Prima fază, cea ce stabilire a cerințelor, este cea în care construim diagrame pentru a înțelege modelul de business. O astfel de diagramă, este diagrama ER (entitate-relație).

Modelul entitate-relație, prescurtat ER este una dintre cele mai populare metode de proiectare a bazelor de date, popularizat de Peter Chen în 1976.

Numim entitate, un obiect(lucru) care poate fi identificat distinct, spre exemplu: un angajat, un document, etc. Entitățile sunt de obicei substantive.

Diagramele ER ne ajută să înțelegem relațiile dintre entități.

În funcție de gradul de asociere dintre două entități, există 3 categorii mari de relații: one-to-one, one-to-many și many-to-many.

În aplicația noastră putem identifica exemple din fiecare tip de relație. O relație one-to-one este ceea cea între tabela de utilizatori și detaliile unei persoane. Orice utilizator va conține o singură legătură către o entitate de persoană. Aceste două entități se țin separat din motive de siguranță dar și pentru a decupla detaliile suplimentare de cele esențiale utilizate de platforma Microsoft Identity.

Vom avea de a face cel mai des cu relații de tipul one-to-many. Spre exemplu, un capitol poate avea unul sau mai multe subiecte.

### 2.1.3 Modelul Entitate-Date

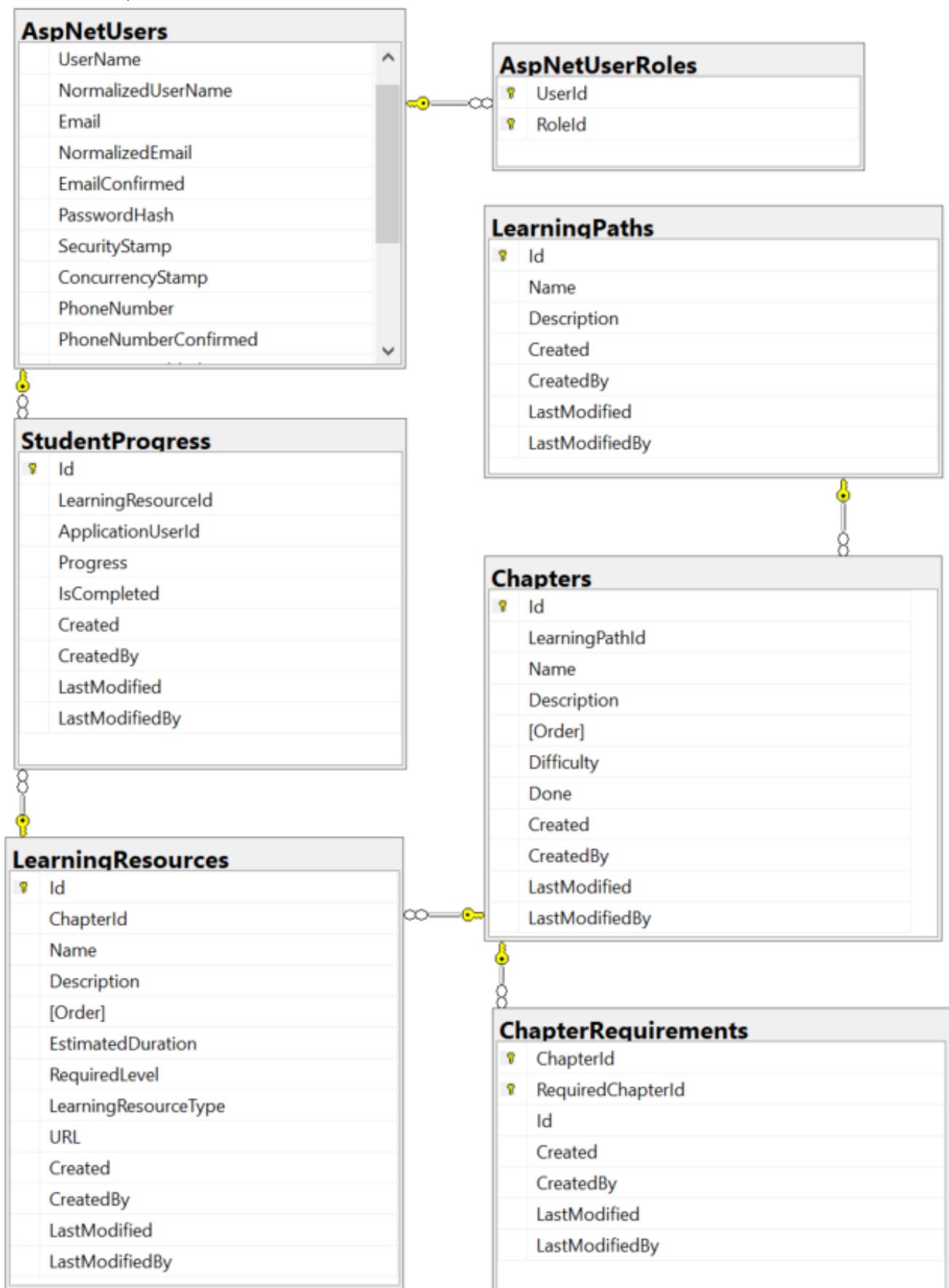
Modelul Entitate-Date (EDM) este bazat pe modelul ER al lui Chen, adresând provocările ce apar odată cu persistarea datelor în mai multe formate. Este teoretic ce descrie structura datelor sub formă de entități și relații. Entitățile și relațiile descrise în model pot fi văzute ca și abstractizări ale obiectelor și legăturilor acestora [20].

Acest model conține 3 sub-componente:

- Modelul de Stocare – reprezintă schema reală a bazei de date (table, view-uri, proceduri stocate și legături între acestea).
- Modelul Conceptual – un limbaj derivat din XML, conținând clase model, independente de baza de date
- Modelul de Mapare – face legătura între cele două de mai sus



În aplicația noastră, schema reală a bazei de date este următoarea:



Figură 7 - schema bazei de date



## 2.1.4 Diagramele de structură (statice)

Acest tip de diagrame descriu aspectele statice ale sistemului informatic.

Printre tipurile de diagrame statice amintim [1] :Diagrama de clase, Diagrame de obiecte, Diagrama de componente, Diagrama de desfășurare (deployment diagram), Diagrama de pachete.

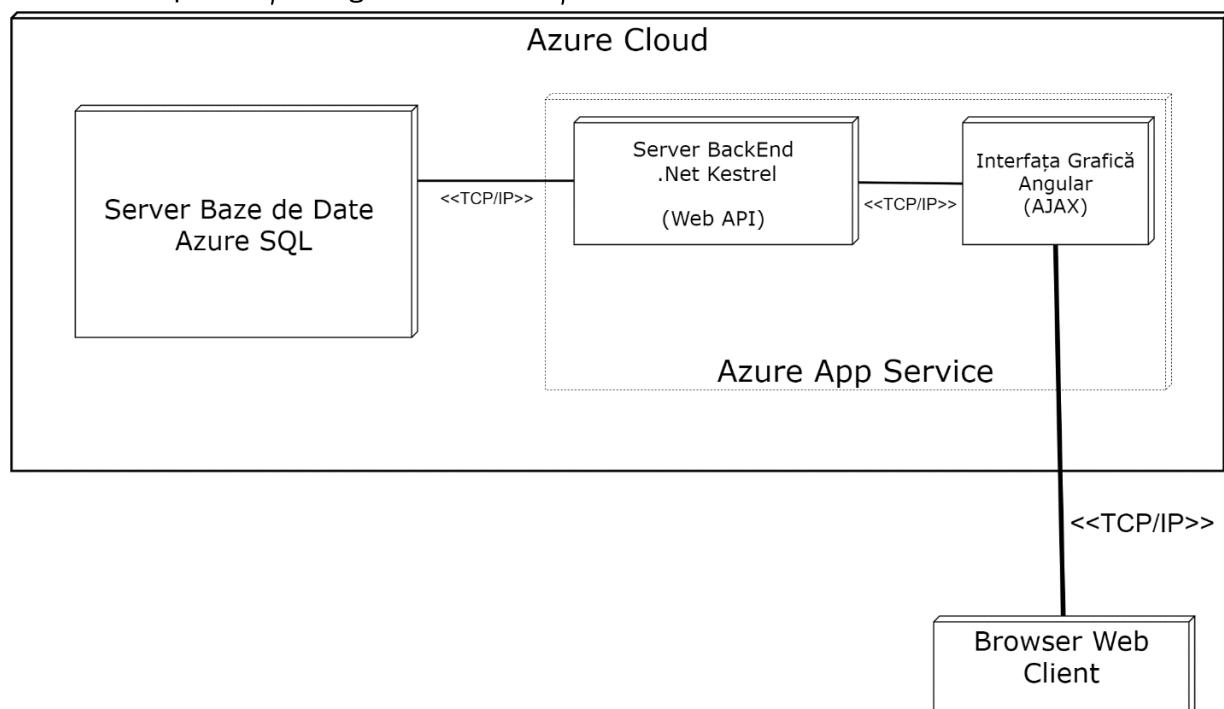
Pentru modelarea diagramelor am folosit softul gratuit ArgoUML și Drow.io

### 2.1.4.1. Diagrama de desfășurare

Diagrama de desfășurare prezintă partitōnarea și topologia componentelor și obiectelor active (server, baze de date, etc) pe locația lor fizică, detaliind locul de amplasare al acestora în cadrul infrastructurii sistemului. Chiar dacă vorbim de o arhitectură Cloud, în spate sunt tot componente fizice interfațate prin serviciile oferite de acel cloud.

Diagramele de desfășurare evidențiază nodurile sistemului și se pot utiliza pentru modelarea topologică a hardware-ului. Ele sunt importante pentru vizualizarea sistemelor client/server unde există o separare clară între datele persistente în sistem și interfața utilizatorului.

În mediu de producție diagrama de desfășurare va arăta astfel:



Figură 8 - Diagrama de desfășurare

## 2.1.5 Digramele de stare (dinamice)

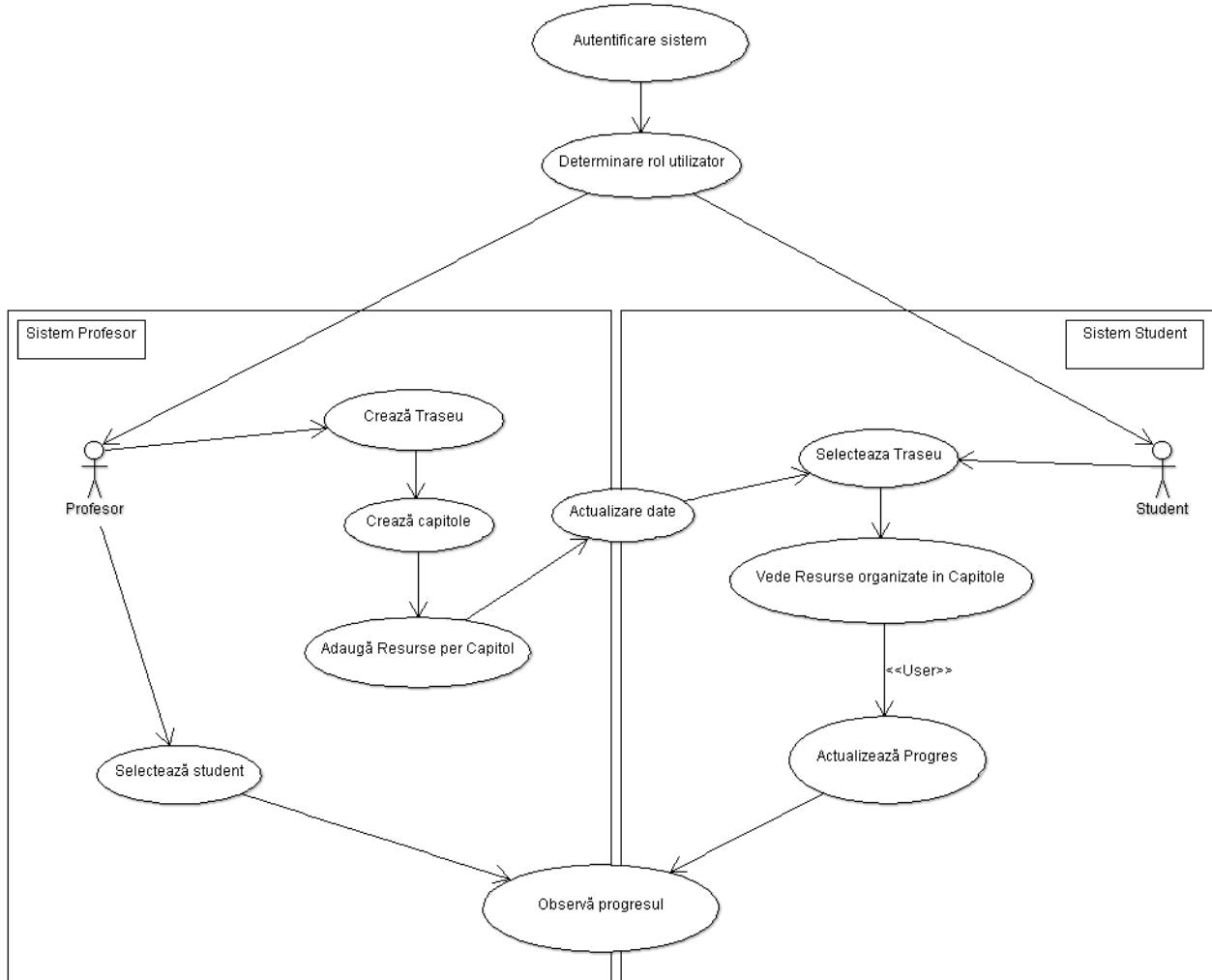
Se mai numesc și diagrame comportamentale deoarece descriu comportamentul sistemului.



Printre tipurile de diagrame dinamice amintim [1] : Diagrama a contextelor de utilizare (use case diagram), Diagrama de secvență, Diagrama de colaborare, Diagrama de stare, Diagrama de activități.

#### 2.1.5.1 Diagrama cazurilor de utilizare

Acest tip de diagramă descrie cel mai bine comportamentul sistemului.



Figură 9 - Diagrama cazurilor de utilizare

Observăm că sistemul este împărțit în două mari subsisteme: sistemul profesorului și sistemul studentului. Această separare este de fapt una logică pentru că, după cum vom vedea în subcapitolul 2.3. (Managementul identităților și al utilizatorilor), un utilizator poate de fapt să aibă mai multe roluri, practic el poate fi și student și profesor în același timp, această decizie fiind luată de administratorul sistemului.

Nu am inclus în această diagramă și partea de gestionare a entităților întrucât nu prezintă o logică prea complicată, acel modul este doar un CRUD cât se poate de generic.



## 2.2. Faza de implementare a bazei de date

Bazele de date, la fel ca majoritatea produselor software, au evoluat enorm în ultimii ani. Avem numeroase tipuri de baze de date pe care le putem folosi în funcție de nevoile aplicației noastre. În .Net, când vrem să interacționăm cu o bază de date relațională, avem următoarele posibilități:

- Folosind direct ADO.NET [3]
- Folosind un ORM (object–relational mapper): Entity Framework, Dapper, NHibernate

Entity Framework este cea mai populară metodă de lucru cu bazele de date în .Net.

Există 3 paradigmă ce pot fi folosite în Entity Framework [21]:

- Model-first: presupune crearea unei diagrame vizuale din care se generează atât baza de date cât și clasele C#
- Database-first: la fel ca și în cazul Model-first, există o diagramă vizuală ce reprezintă baza de date doar că această diagramă este actualizată automat
- Code-first: este cea mai modernă abordare, care elimină complet diagrama vizuală și actualizează structura bazei de date pornind despre DTO-uri.

### 2.2.1 Abordarea code-first în proiectarea bazelor de date

Atunci când comunicăm cu baza de date, pentru a avea acces la entitățile din bază, trebuie să le transformăm în obiecte pe care limbajul de programare din back-end să le înțeleagă (în cazul nostru C#). Practic trebuie să transformăm rezultatul unui query din baza de date într-un obiect de C#. Un astfel de obiect este numit DTO (Data Transfer Object) deoarece singurul lui rol este să preia datele din baza de date și eventual să le transfere înapoi. Nu este obligatoriu să folosim același DTO pentru citire și pentru scriere, este posibil ca anumite priorități să nu fie disponibile pentru scriere, spre exemplu data în care a fost creată entitatea.

În literatura de specialitate, un obiect foarte simplu care nu depinde de alte librării externe, este numit POCO (Plain Old CLR Object), acest termen variază în funcție de limbajul pe care îl folosim; spre exemplu, în JAVA, Martin Fowler l-a numit POJO (Plain Old Java Object). Un DTO este un POCO ce nu conține comportament (metode) ci doar date (proprietăți și câmpuri).

Un DTO este foarte util în cazul în care vrem să reprezentăm structura unei entități din baza de date. În .Net avem mai multe opțiuni pentru a obține obiecte din baza de date. Putem folosi ADO.Net spre exemplu însă va trebui să sincronizăm tot timpul DTO-ul nostru în funcție de structura bazei de date sau invers. Pentru a simplifica acest proces de sincronizare și pentru nu fi nevoie să actualizăm permanent modelul în care "vărsăm" datele dinspre baza de date, putem folosi un ORM (Object-Relationship Mapper).



## 2.2.2 Sistemul de migrări pentru versionarea bazei de date

În orice tip de sistem informatic, versionarea este un aspect esențial, nu e de mirare că și atunci când lucrăm cu o bază de date, avem nevoie să o versionăm pentru a ști exact ce structură avem și dacă se potrivește cu DTO-urile noastre din codul server-site. Datorită faptului că am ales o abordare code-first, vom versiona (folosind GIT) de fapt codul care reprezintă migrarea ce trebuie executată. Acest cod se va găsi în același proiect în care se află și codul pentru logica de back-end, baza de date fiind în strânsă legătură cu partea de serve-side (mai mult decât se leagă de partea de client-side).

O migrare reprezintă un set de schimbări ce trebuie executate asupra bazei de date, cum ar fi: adăugarea unei tabele sau a unei coloane, modificarea sau ștergerea acestora, practic orice modificare supra schemei bazei de date. Migrarea nu are ca și scop modificarea datelor din baza de date, pentru astfel de operațiuni există alte procese, cum ar fi procesul de "Seed" descrise mai jos.

Putem intui cu ușurință că această componentă de gestionare a migrărilor trebuie să fie prezentă în orice sistem informatic modern. Aproape toate framework-urile din orice limbaj de programare au un sistem de migrare propriu, în PHP, framework-ul Laravel spre exemplu, dispune de un sistem de migrări foarte similar cu cel din .Net. Sunt foarte puține cazuri în care am avea nevoie să ne cream noi propriul sistem în care să scriem cod SQL și să-l executăm folosind ADO.Net spre exemplu.

În ecosistemul .Net, atunci când folosim Entity Framework, avem la dispoziției un utilitar ce poate fi accesat din linia de comandă în două moduri:

- Folosind linia de comandă a managerului de pachete pentru .Net, nuGet din interiorul editorului Visual Studio
- Folosind utilitarul 'dotnet' din linia de comandă care este poate fi accesat din orice tip de linie de comandă (PowerShell, CMD, Bash) pe orice sistem de operare.

În oricare dintre aceste variante, există doi pași:

Pasul 1: generarea migrării ca și cod C# reprezentat printr-o clasă ce implementează interfața **Migration** care definește două metode: **Up** pentru actualizarea bazei de date și **Down** pentru operațiunea inversă.

Pasul 2: aplicarea migrării asupra bazei de date. Acest pas se execută de regulă automat, atunci când pornește aplicația, dar se poate executa și manual folosind unul din cele două utilitare descrise mai sus.

Pentru a profita de ecosistemul .Net Core, am ales a doua variantă, și am creat două scripturi de PowerShell pe care le putem executa prin banalul dublu-click, pentru a ne face



viață mai ușoară. În aceste scripturi am specificat și numele proiectului unde sunt definite DTO-urile ce definesc structura bazei de date.

Astfel, pentru pasul 1 (generarea codului de migrare) vom executa următoarea comandă:

```
dotnet ef migrations add "NumeleMigrării"
```

Iar pentru pasul 2 ( aplicarea migrării) vom executa următoarea comandă:

```
dotnet ef database update
```

În cazul în care există mai multe migrări în așteptare, acestea se vor executa pe rând. Sistemul de migrare este suficient de deștept încât să știe dacă mai are de executat migrări sau nu. Istoricul acestor migrări este păstrat în tabela de sistem `__EFMigrationsHistory`.

### 2.2.3 Popularea bazei cu date inițiale prin metoda de Seed

Orice sistem informatic are nevoie de un pas inițial în care se definesc variabilele de sistem, utilizatorii standard și diverse configurații. Acest pas mai poartă denumirea de provizionare (provisioning).

În cazul aplicației noastre, în partea de provizionare creăm utilizatorul administrator, cu care putem mai apoi crea restul utilizatorilor și entităților. Acest utilizator are următoarele informații de autentificare(ele se pot schimba după prima autentificare).

Username: `administrator@localhost` | Parola : `Administrator1!`

## 2.3. Managementul identităților și al utilizatorilor

Orice aplicație, dar mai ales o aplicație web, trebuie să aibă un sistem de autentificare multiplă pentru a putea fi folosită de utilizatori diferiți.

Atunci când un utilizator este creat, el poate avea unul sau mai multe roluri. Aceste roluri sunt folosite pentru a oferi acces limitat la anumite funcționalități. Această tehnică de autorizare pe bază de roluri [23] este una dintre cele mai populare metode de controlare a accesului.

În aplicația noastră, am definit 3 roluri, un utilizator putând avea unul sau mai multe:

- Administrator: Poate crea, modifica sau șterge alți utilizatori.
- Profesor: Poate crea, modifica sau șterge resursele (trasee, capitole, resurse educaționale) precum și legăturile dintre ele.
- Student: Poate vedea resursele și poate actualiza progresul pe o anumită resursă.

În baza de date, utilizatorii sunt persistați în table `AspNetUsers`, rolurile sunt definite în tabela `AspNetRoles` iar legăturile dintre acestea se găsesc în tabela `AspNetUserRoles`.



### 2.3.1 Managementul identității și accesului (IAM)

Pe lângă faptul că, în funcție de rolurile utilizatorului autentificat, vor apărea sau dispărea elemente în bara de navigare. Accesul acestuia este blocat și la nivel de resursă HTTP. În cazul în care utilizatorul reușește să păcălească interfața pentru a afișa meniurile ascunse, acesta va fi blocat de către API și nu va putea avea acces la informații nepermise.

Această limitare la nivel de API se realizează folosind atributul de [Authorize]. Putem vedea acest atribut în cazul Controller-ului de Management al utilizatorilor.

[Authorize(Roles = "Administrator")]

Astfel, doar administratorii pot accesa acțiunile din acest controller.

### 2.3.2 Autentificarea prin tokeni JWT

JSON web token (JWT) este un standard open-source definit de RFC 7519 ce definește un mod compact de a trimite informații de autentificare sub formă de obiecte JSON. [24]

Un token JWT are 3 componente despărțite prin punct (xxxx.yyyy.zzz):

- Header: conține tipul de token și algoritmul de criptare
- Payload: conține informații despre entitate (claims)
- Semnătură: folosită pentru a ne asigura că informațiile nu au fost modificate

Tot în partea de payload vom găsi și rolurile utilizatorului. Folosind aplicația web JWT.io, putem vedea exact ce conține un Token JWT trimis pe un request către API:

HEADER: ALGORITHM & TOKEN TYPE
{ "alg": "RS256", "kid": "Development", "typ": "at+jwt" }
PAYLOAD: DATA
{ "nbf": 1622291339, "exp": 1622294939, "iss": "https://localhost:8904", "aud": "LearningMap.WebAPIAPI", "client_id": "LearningMap.Web", "sub": "795c2d93-f21e-4a16-aa11-692f18f903a9", "auth_time": 1622291335, "idp": "local", "role": "Teacher", "email": "teacher1@localhost", "preferred_username": "teacher1@localhost", "name": "teacher1@localhost", }

Figură 10 - Decodare token JWT



### 2.3.3 Microsoft Identity Platform

Platforma Microsoft Identity reprezintă o suită de tehnologii și standarde ce ne ajută să adăugăm autentificare și autorizare pentru API-urile noastre [22].

Cu ajutorul platformei, putem genera și valida token de autentificare, pentru mai multe scenarii, cum ar fi : SPA-uri, aplicații web server-side, API-uri, aplicații desktop sau mobile. Platforma face parte din categoria produselor IDP (Identity Data Provider) care sunt produse software ce persistă, gestionează și verifică identitățile utilizatorilor.

Un IDP se poate integra și cu alte IDP-uri, am putea spre exemplu să ne integrăm cu Azure AD, Google sau Facebook pentru a le permite utilizatorilor să-și folosească identitățile gestionate de alți furnizori de identitate; astfel, nu ar trebui să mai fim noi responsabili pentru gestionarea datelor personale.

Microsoft Identity ne oferă inclusiv funcționalitatea de a înregistra utilizatori, furnizând ecrane standard pentru înregistrarea unui cont nou. În cazul aplicației noastre, permitem crearea de conturi noi unde asociem utilizatorului rolul de student.

Un alt proiect adesea folosit pentru autentificarea API-urilor este Identity Platform, o platformă gratuită și open-source care se află momentan la versiunea 4 dar care urmează să fie închis în 2021 din considerente comerciale.

#### 2.3.3.1 Microsoft Identity pentru SPA-uri

După cum am menționat în introducerea lucrării, aplicația este de fapt un SPA (Single Page Application) folosind Angular pentru a realiza această implementare. Deși nu suntem limitați la a folosi Angular pentru a construi un SPA, .Net Core oferă suport excelent pentru integrarea cu acest framework.

Folosind metoda extensia `.AddIdentityServerJwt` orice request care necesită autentificare (definită prin attribute) va necesita un Token JWT care va fi validat de către Identity Platform.

Aplicația Angular, va folosi endpoint-ul `_configuration/{clientId}` pentru a citi configurațiile: ce scopuri se permit și care sunt URL-urile pentru redirectare.

Acest mecanism de comunicare al aplicației Front-End cu Identity Platform a fost generat în momentul în care am creat proiectul și am selectat template-ul pentru Angular. În acel moment s-a generat directorul `ClientApp\src\api-authorization` care conține interceptorii din Angular ce se ocupă să adauge tokenul JWT obținut de la Identity Platform pe fiecare request făcut către API.

Folosind Chrome Developer Tools putem observa cum acest Token JWT este adăugat la fiecare request pe header-ul de "authorization".

The screenshot shows the Fiddler Headers tab for a request named "LearningPaths". Under "Request Headers", the "authorization" header is displayed with a long JWT token value.

```

:authority: localhost:8904
:method: GET
:path: /api/LearningPaths
:scheme: https
accept: application/json
accept-encoding: gzip, deflate, br
accept-language: en-US,en;q=0.9,ro;q=0.8
authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IkRldmVsb3BtZW50IiwidHlwIjoiYiIsInN1YiI6ImQ2YjI5NTJiLWFmZAtNDAsMi1hOTY2LTM2NjY4YzKxMDRKOSIsImF1dGhfdGZXJAbG9jYWxob3N0IiwibmFtZSI6ImRldmVsb3BlckBsb2NhbgHvc3QiLCJqdGkiOiI2QzQ4OEfaWx1I10sImFtcIi6WlyJwd2QjXX0.dfxFzoS1S-66fbU1P8g6pwIIy8N71wgnkbUr_F8rZw3eRhsQd10Q4CKkemC1dHfRa1awJ1vSnacqk2ktM0BoUzVRcpqT-X9QosHroACxoBcPMMA_qcQu0Qf

```

Figură 11 - Header de autorizare

Din analiza traficului, folosind programul Fiddler, observăm următoarele requesturi:

The screenshot shows the Fiddler interface with a list of network requests. Request number 78, which is the "connect/token" request, is highlighted with a red box. The details pane shows the JSON response body containing the generated JWT token.

#	Result	Protocol	Host	URL	Content-Type	Comments
52	200	HTTPS	localhost:8904	/Identity/Account/Login?ReturnUrl=...	text/html; charset=UTF-8	Present Login Screen
61	200	HTTPS	localhost:8904	/authentication/login-callback?code=...	text/html; charset=UTF-8	Start Authentication
76	200	HTTPS	localhost:8904	/_configuration/LearningMap.Web	application/json; charset=UTF-8	Get Scopes
77	200	HTTPS	localhost:8904	/.well-known/openid-configuration	application/json; charset=UTF-8	Get Identity Server Config
78	200	HTTPS	localhost:8904	/connect/token	application/json; charset=UTF-8	Get JWT ID Token
79	200	HTTPS	localhost:8904	/connect/userinfo	application/json; charset=UTF-8	Get User Roles

**Filters** Hide 'slack.com' Hide 'Firefox.settings.services.mozilla.com' Hide 'dc.services.visualstudio.com' Hide '/quickpulsebservice.svc/'

**Raw** **JSON** **XML**

name	value
client_id	LearningMap.Web
code	FEAEAA1BB9DB6E862232A37AC0D26B3F9
redirect_uri	https://localhost:8904/authentication/login
code_verifier	374dfa20e9844a7388d0082cad10287af63
grant_type	authorization_code

**JSON**

```

access_token=eyJhbGciOiJSUzI1NiIsImtpZCI6IkRldmVsb3BtZW50IiwidHlwIjoiYiIsInN1YiI6ImQ2YjI5NTJiLWFmZAtNDAsMi1hOTY2LTM2NjY4YzKxMDRKOSIsImF1dGhfdGZXJAbG9jYWxob3N0IiwibmFtZSI6ImRldmVsb3BlckBsb2NhbgHvc3QiLCJqdGkiOiI2QzQ4OEfaWx1I10sImFtcIi6WlyJwd2QjXX0.dfxFzoS1S-66fbU1P8g6pwIIy8N71wgnkbUr_F8rZw3eRhsQd10Q4CKkemC1dHfRa1awJ1vSnacqk2ktM0BoUzVRcpqT-X9QosHroACxoBcPMMA_qcQu0Qf
expires_in=3600
id_token=eyJhbGciOiJSUzI1NiIsImtpZCI6IkRldmVsb3BtZW50IiwidHlwIjoiYiIsInN1YiI6ImQ2YjI5NTJiLWFmZAtNDAsMi1hOTY2LTM2NjY4YzKxMDRKOSIsImF1dGhfdGZXJAbG9jYWxob3N0IiwibmFtZSI6ImRldmVsb3BlckBsb2NhbgHvc3QiLCJqdGkiOiI2QzQ4OEfaWx1I10sImFtcIi6WlyJwd2QjXX0.dfxFzoS1S-66fbU1P8g6pwIIy8N71wgnkbUr_F8rZw3eRhsQd10Q4CKkemC1dHfRa1awJ1vSnacqk2ktM0BoUzVRcpqT-X9QosHroACxoBcPMMA_qcQu0Qf
scope=LearningMap.WebAPI openid profile
token_type=Bearer

```

Figură 12 - Analiză trafic cu Fiddler

Observăm că acest token este generat imediat după ce se face login.

### 2.3.3.2 Extinderea platformei Identity Platform 4

Identity Platform vine la pachet cu o clasă generică numită `IdentityUser`, această clasă conține câmpurile care definesc un utilizator. Pentru a nu îngreuna sistemul și pentru a-i oferi utilizatorului doar ceea ce este stric necesar, această clasă definește doar câteva proprietăți, dintre care doar două sunt obligatorii: `username` și `email` (acesta două fiind diferite).

Mai avem un câmp optional pentru telefon.

Dacă dorim să atașăm mai multe informații, tot ceea ce trebuie să facem este să creăm o clasă nouă care să moștenească clasa de bază `IdentityUser` și să adăugăm câmpurile suplimentare.



Concret, am creat o clasă `LearningMapUser` în care vom adăuga câmpuri suplimentare pentru nume și data nașterii. Folosind conceptul de Middleware din .Net Core, injectăm această clasă în Identity Platform folosind două linii de cod:

```
.AddDefaultIdentity<LearningMapUser>()
.AddApiAuthorization<LearningMapUser, ApplicationDbContext>(options
=> { });
```

Pentru a reflecta aceste câmpuri și în baza de date, va trebui să generăm o migrare, urmând procesul descris în subcapitolul 2.2.2 (Sistemul de migrări pentru versionarea bazei de date).



### 3. Grafuri aplicate în reprezentări vizuale

Grafurile sunt structurile de date ideale pentru a reprezenta relațiile dintre obiecte. Modelarea elementelor dintr-un circuit sau rețea se realizează de obicei cu ajutorul unui graf. Teoria grafurilor s-a dezvoltat în paralele cu algebra, având aplicări în diverse domenii ale matematicii (teoria grupurilor, cercetări operaționale, etc.) dar sunt folosite și în rezolvarea unor probleme economice, tehnice sau informatiche.

#### 3.1 Concepte Generale

Din punct de vedere matematic, un graf  $G$  este un obiect format din două mulțimi: mulțimea nodurilor (vârfurilor) nodată de obicei cu  $V$  și mulțimea muchiilor (arcelor) care conectează vârfurile, notată de obicei cu  $E$  [4].

$$G = (V, E)$$

Ca și componente, într-un graf avem două tipuri de obiecte: noduri și muchii. În cazul digrafurilor, muchiile se mai numesc arce. În cadrul unui arc putem vorbi despre nodul origine și nodul destinație.

Două noduri între care există o muchie, se numesc adiacente.

Dacă o muchie se leagă cu un nod, vom spune că muchia este incidentă aceluia nod (are o conexiune cu acesta). În cazul digrafului, spunem despre un arc că este incident din (pleacă din) nodul  $N_1$  și este incident în (ajunge în) nodul  $N_2$ .

##### 3.1.1 Tipuri de grafuri

Grafurile se pot împărți în mai multe categorii, în funcție de criteriul de clasificare. În cazul în care ne interesează direcția unei muchii și nu putem presupune că ne putem deplasa dintr-un nod în altul în mod bi-direcțional, atunci putem spune că avem de a face cu un graf orientat (sau digraf, prescurtare de la termenul din engleză de Directed Graph), altfel, avem un graf neorientat.

După cum am precizat în introducere, una dintre entitățile de bază ale aplicație Learning Map este traseul educațional. Acest traseu va fi în esență un graf orientat, unde nodurile sunt capitoalele ce fac parte din traseul de învățare.

Am luat în considerare și posibilitatea de a folosi arbori pentru a reprezenta mai bine ierarhiile însă nu m-ar fi ajutat faptul că fiecare nod ar fi trebuit să aibă un singur părinte. Această restricție ar fi făcut traseul educațional să fie destul de rigid și nu mi-ar fi permis să ajung de la un capitol la altul fără a trece prin nodul părinte.

Un graf este conex dacă între oarecare două noduri ale sale există cel puțin un lanț (legătură), pentru grafurile conexe, nu se ține cont de orientarea arcelor.



Numim gradul unui nod, numărul de muchii incidente cu acesta. Un nod care are gradul 1 se va numi terminal deoarece nu mai puteam vizita noduri noi odată ce am ajuns la acesta. În cazul digrafurilor vorbim despre două tipuri de grade:

- gradul interior (sau de intrare) reprezintă numărul de arce care intră în acesta
- gradul exterior (sau de ieșire) reprezintă numărul de arce care ies din acesta

Un graf ponderat este acela în care fiecare muchie are o anumită valoare asociată, astfel putem reprezenta costul unui drum.

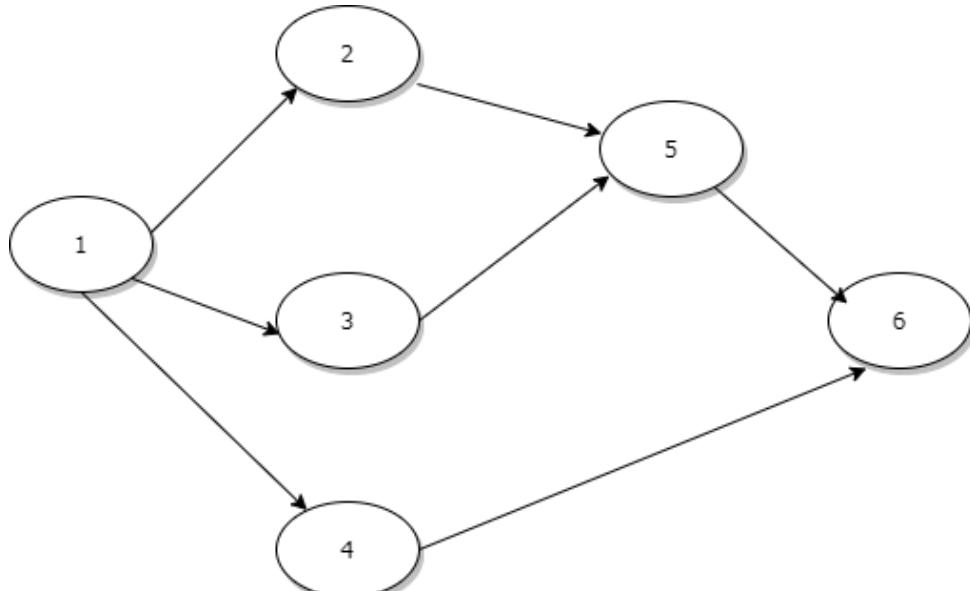
Ordinul grafului este dat de către numărul de noduri; concret în aplicația noastră, ordinul grafului va reprezenta numărul de capitole dintr-un traseu educațional.

### 3.1.2 Reprezentarea unui graf

Pentru a reprezenta un graf, trebuie în primul rând să asociem o cheie fiecărui nod. Din moment ce în aplicația noastră un nod înseamnă un capitol, care este de fapt o entitate din baza de date, vom folosi ID-ul capitolului ca și cheie. Acest ID este și cheia primară din baza de date.

Există două metode modele matematice larg acceptate pentru a reprezenta un graf, acestea sunt: matricea de adiacență și lista de adiacență.

Vom lua ca și exemplu următorul graf orientat unde fiecare nod reprezintă un capitol iar numărul asociat nodului reprezintă ID-ul (cheia) capitolului.



Figură 13 - Exemplu digraf

Observăm, ca și particularitate, că avem o singură direcție și că nu există arce bidirectionale. Am ales această structură deoarece am considerat că din moment ce ai completat un capitol, poți doar să progresezi, nu are sens să te întorci pe același drum.



### 3.1.2.1 Matricea de adiacență

După cum am precizat mai sus, spunem că două noduri sunt adiacente în cazul în care există o legătură directă între acestea.

Un graf cu "n" noduri se poate reprezenta folosind o matrice pătratică cu n linii și n coloane.

Pentru un graf neorientat am completa matricea cu 1 sau 0.

Vom completa 1 în căsuța  $a[i,j]$  dacă nodurile i și j sunt adiacente și 0 în cazul contrar.

În cazul unui graf orientat, va trebui să luăm în considerare și direcția arcelor, astfel că vom completa căsuța  $a[i,j]$  după următoarea regulă:

- 1 doar dacă există un arc dinspre nodul i spre j și
- 0 în rest

În unele cazuri, pot exista mai multe arce cu aceeași direcție între două noduri, mai ales dacă avem un graf ponderat, atunci s-ar completa în matricea de adiacență numărul de arce. În aplicația de față nu am luat în considerare acest scenariu, deoarece ne interesează doar dacă există sau nu o legătură între cele două noduri.

Pentru exemplul de mai sus, ar rezulta următoarea matrice de adiacență:

Sursă/ Destinație	1	2	3	4	5	6
1	0	1	1	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	0	0	1
5	0	0	0	0	0	1
6	0	0	0	0	0	0

Tabel 1 - Exemplu matrice adiacență

Observăm că această reprezentare este foarte ușor de înțeles și intuitivă, însă suntem forțați să folosim mereu o matrice de aceeași dimensiune (în funcție de ordinul grafului). Putem observa că multe căsuțe sunt completate cu zero, practic avem 7 muchii, deci vom completa doar căsuțe cu cifra 1, iar restul de 29 de căsuțe vor fi completate cu zero. Din acest punct de vedere, matricea de adiacență nu este o reprezentare optimă ca și utilizare a memoriei.

### 3.1.2.2 Lista de adiacență

Un graf cu "n" noduri se poate reprezenta ca o listă cu n elemente corespunzătoare fiecărui nod.

Într-un graf neorientat, pentru fiecare nod, vom asocia o listă de noduri adiacente.



În cazul digrafurilor, lucrurile se complică puțin, pentru că vom avea două liste: una pentru arcele care au nodul dat ca și sursă, iar a doua pentru arcele care au nodul dat ca și destinație. Altfel spus, avem o listă a succesorilor unui nod și o listă a predecesorilor unui nod.

Pentru graful din exemplu de mai sus, vom reprezenta lista de adiacență a succesorilor (nodurile care pleacă din-un anumit nod dat)

Nod referință	Noduri adiacente care au acest nod ca și sursă		
1	2	3	4
2	5		
3	5		
4	6		
5	6		
6			

Tabel 2 - Exemplu listă adiacență succesi

Iată acum reprezentarea inversă pentru același exemplu, în care vom nota doar nodurile care au un nod dat ca și succesor.

Nod referință	Noduri adiacente care au acest nod ca destinație		
1			
2	1		
3	1		
4	1		
5	2	3	
6	5	4	

Tabel 3 - Exemplu listă de adiacență predecesori

Alegerea uneia dintre cele două variante depinde de preferințe și de cazul de utilizare. Pentru exemplu nostru practic, ajută mai mult varianta a doua deoarece putem spune că pe prima coloană avem nodul referință apoi avem o listă de noduri care ne pot duce la nodul referință. Practic, avem un capitol urmat de o listă de capitole care ne pot aduce la acest capitol.

Am luat în considerare și folosirea unei vaze de date specializate în stocarea grafurilor însă pentru cazul curent o bază de date SQL este suficientă.



### 3.1.2.3 Persistarea unui digraf în baza de date

O bază de date trebuie să fie rapidă și să ocupe cât mai puțin spațiu, din acest motiv, am eliminat reprezentarea prin matricea de adiacență și am ales varianta reprezentării prin liste de adiacență. Mai exact, varianta a doua, unde salvăm nodurile predecesoare unui anumit nod.

Deoarece folosim o bază de date relațională, trebuie să avem același număr de coloane pentru fiecare entitate, altfel, am aplicat următoarea structură: fiecare pereche de nod destinație-sursă, va fi salvat individual pe câte un rând, astfel, dimensiunea listei va fi mereu 2. Primul element va fi nodul destinație iar al doilea element va fi nodul sursă.

Astfel, pentru exemplul de mai sus, am avea următoarea reprezentare în baza de date.

Nod destinație	Nod sursă
2	1
3	1
4	1
5	2
5	3
6	5
6	4

Tabel 4 - Listă de adiacență normalizată

Observăm că nodul 1 nu are nici un nod sursă, putem presupune că nodul 1 este rădăcină. În cazul aplicației noastre, acest nod va reprezenta capitolul cu elemente introductory, spre exemplu sintaxa limbajului de programare.

## 3.2. Vizualizarea grafică a datelor

Vizualizarea datelor face parte din multimea de instrumente pentru business-intelligence care sunt esențiale pentru o analiză avansată a unui sistem informatic și a datelor prelucrate/furnizate de acesta. Scopul domeniului este să ajute oamenii să înțeleagă toate informațiile sau datele generate în prezent. Prin vizualizarea datelor, informațiile sunt reprezentate în formă grafică, sub forma unui grafic circular sau a altui tip de prezentare vizuală. Jucătorul principal în acest domeniu al reprezentării și vizualizării datelor este Microsoft Power BI.

Se spune că „o imagine valorează cât o mie de cuvinte”. Astăzi, în epoca Big Data, când majoritatea domeniilor sunt bombardate de informații provenite din diverse tipuri de date



și din surse atât on-premise, cât și din Cloud, zicala această veche este mai adevărată ca oricând.

Cert este că a devenit din ce în ce mai dificil să navigăm printre informații pentru a înțelege ce este important și ce nu. Vizualizarea simplifică și accelerează analiza, oferind posibilitatea de a afla dintr-o privire ceea ce ne interesează. Oamenii reacționează mult mai bine la imagini decât la text.

Instrumentele de vizualizare a datelor sunt capabile să se conecteze la surse de date(bazele de date relaționale, API-uri sau chiar fișiere Excel). Aceste date, care pot fi stocate on-premise(servere aflate sub controlul nostru) sau în cloud, sunt preluate și prelucrate pentru analiză. Utilizatorii pot selecta apoi cel mai bun mod de a prezenta datele din numeroase opțiuni. Unele instrumente oferă automat recomandări de afișare, în funcție de tipul de date prezentate.

### 3.2.1 Alegerea unei biblioteci javaScript pentru vizualizarea datelor

Pentru ca utilizatorul să aibă o înțelegere cât mai bună asupra entităților din aplicație și a stării acestora la un moment dat, avem nevoie să reprezentăm datele într-o manieră cât mai eligibilă. Pentru vizualizarea traseului educațional, am ales o structură de graf orientat, unde capitolele reprezintă nodurile grafului.

Deoarece avem de a face cu o aplicație web, am ales o librărie javaScript care să ne ajute să vizualizăm acest graf. Cercetând opțiunile disponibile, am avut de ales între 4 librării: ngx-graph, Cytoscape.js, Mermaid și Highcharts. În tabelul de mai jos, am făcut o comparație între cele patru librării, considerând diverse aspecte ale acestora.

Datele din tabel au fost actualizate în 26 mai 2021.

Nume Librărie	ngx-graph	Cytoscape.js	Mermeid	Highcharts
Tipul de Licență	Open Source (MIT)	Open Source (MIT)	Open Source (MIT)	Comercială sau Non-Profit
Stele GitHub	725	7,700	36,100	10,100
Download-uri săptămânale NPM	31	41,027	92,658	531,252
Dimensiunea totală a librăriei	1.31 MB	4.26 MB	13.1 MB	45.5 MB
Complexitate	Mică	Medie	Mică	Mare
Documentație	Bună	Foarte bună	Bună	Foarte bună
Exemple	Puține	Multe	Puține	Multe
Suport Angular	Nativ	cytoscape-angular	framework agnostic	highcharts-angular

Tabel 5 - Comparație librării JS pentru vizualizarea grafurilor

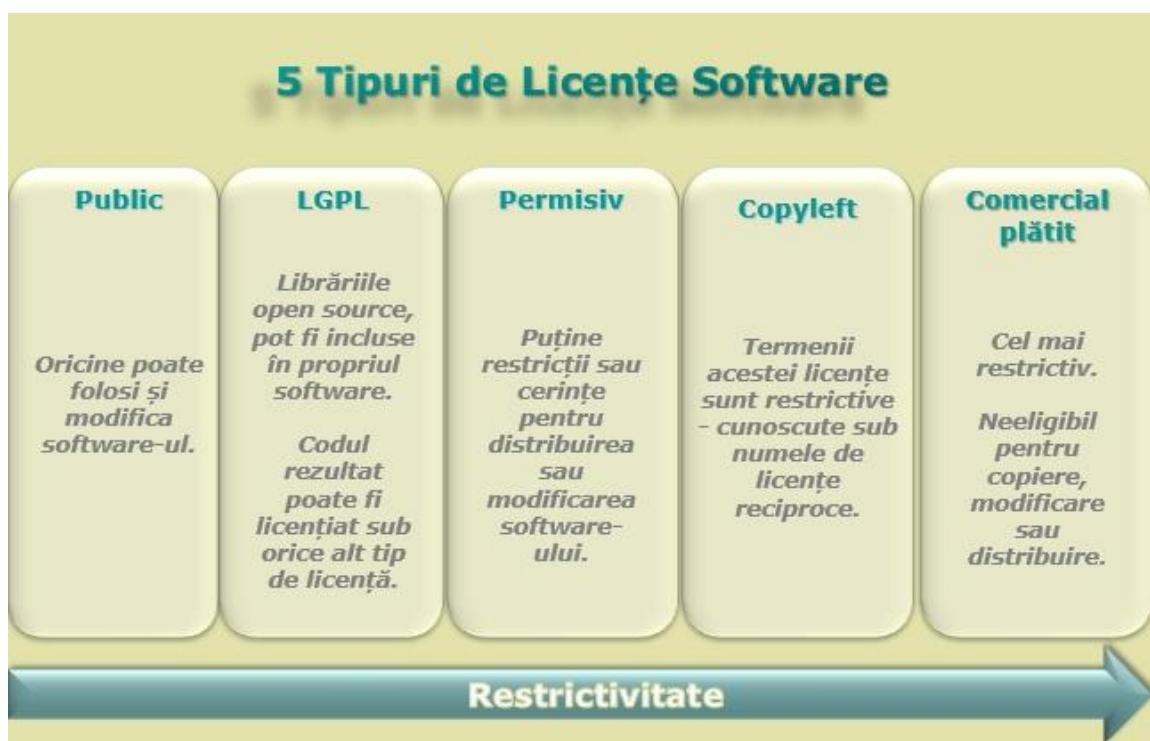


Vom explica pe rând criteriile pe care le-am luat în considerare în luarea deciziei.

### 3.2.1.1 Tipul de Licență

Deoarece avem de afacere cu un proiect didactic, vrem să ne asigurăm că toate librăriile folosite nu intră sub incidența drepturilor de autor. Teoretic, aproape orice aplicație se poate folosi cu scop demonstrativ, dar, pentru siguranța noastră, cel mai bine este să consultăm documentul de licență atașat produsului.

Există cinci categorii [25] principale de licențe software utilizate pentru a acoperi diferite tipuri de aplicații și diverse contracte business. Acestea cuprind un spectru larg de scenarii de licențiere, de la software gratuit (domeniu public) la software comercial plătit (proprietar). Între aceste două extreme, există și trei categorii (GNU/LGPL, permissive și copyleft) care se aplică la diferite forme de proiecte open-source. Nerespectarea termenilor și condițiilor unei licențe open-source poate duce la dezvăluirea secretelor comerciale ale unei companii sau chiar a acțiunilor legale din partea dezvoltatorilor proiectului. Cele cinci categorii de licențe software, sunt următoarele:



Figură 14 - Tipuri de licențe software

#### 1. Public

Când software-ul este definit ca fiind în domeniul public, oricine este liber să utilizeze și să modifice software-ul fără restricții. Aceasta este o licență „permisivă” care permite adoptarea codului în aplicații sau proiecte și reutilizarea software-ului după preferințe.



Din multe motive, companiile trebuie să fie prudente atunci când adoptă un astfel de software de domeniu public în proiecte sau alte aplicații importante:

- Este posibil ca software-ul din domeniul public să nu adere întotdeauna la cele mai bune practici de codare sau să nu fie conform standardelor de software securizat pe care le cere aplicația
- Software-ul care nu se încadrează în termeni specifici de licențiere nu este întotdeauna cod de domeniu public, din acest motiv, înainte de a copia, reutiliza sau distribui, cei care îl folosesc, trebuie să se asigure că software-ul este cu adevărat domeniu public.

## 2. GNU/LGPL – GNU Lesser General Public License (LGPL)

Conform unei licențe LGPL, dezvoltatorii au dreptul de a include librăriile open source în propriul software. Codul rezultat poate fi licențiat sub orice alt tip de licență, chiar și comercial plătit, atunci când proiectele sunt compilate sau legate pentru a include o librărie cu licență GPL.

Avertismentul este că, dacă orice parte a librăriei este copiată în cod sau modificată, termenii licenței LGPL originale, se vor aplica codului dezvoltat care a folosit librăria.

## 3. Permisiv

Acest tip de licență este una dintre cele mai frecvente și populare printre licențele software open-source. Sub o licență permisivă, denumit și „Apache” sau „stil BSD”, există puține restricții sau cerințe pentru distribuirea sau modificarea software-ului. O altă variantă a unei licențe software permisive este licența „MIT” (folosită inclusiv de către Angular și .Net Core).

Variantele din licențele permisive includ diferențe în cerințele pentru păstrarea notificărilor de licență și drepturilor de autor pentru software, precum și modul în care software-ul poate fi utilizat (comercial sau privat), cerințele privind mărcile comerciale și alte stipulații.

## 4. Copyleft

Termenii acestei licențe sunt restrictive – cunoscute sub numele de licențe reciproce. În condițiile unei licențe copyleft, codul licențiat poate fi modificat sau distribuit ca parte a unui proiect software, dacă noul cod este distribuit sub aceeași licență software. Aceasta înseamnă că, dacă codul inclus în produsul software a fost specificat pentru a fi „numai pentru uz personal”, noul produs distribuit trebuie să conțină aceeași desemnare/restricție.

Întrucât software-ul original inclus în noul proiect a permis modificări și distribuire, este posibil să nu fie cea mai bună licență pentru dezvoltatorii de software, deoarece codul



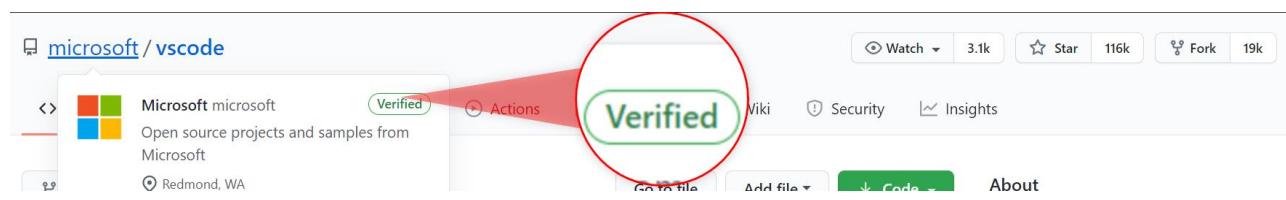
rezultat trebuie să poarte și tipul de licență copyleft, inclusiv disponibilitatea codului sursă.

## 5. Software comercial plătit (proprietar)

Aceste licențe software fac software-ul neeligibil pentru copiere, modificare sau distribuire. Acesta este cel mai restrictiv tip de licență software, protejând dezvoltatorul sau proprietarul de utilizarea neautorizată a software-ului.

### 3.2.1.2 Popularitate (stele GitHub)

GitHub este de departe cea mai populară platformă de distribuire a codului open-source. Dar cum știm dacă un cod disponibil în mod public este suficient de bun? Practic orice utilizator poate publica o bucată de cod pe care alții să o folosească în neștiere. În primul rând, companiile mari au opțiunea să devină parteneri verificăți. Vedem astfel că, utilizatorul GitHub "microsoft" are o insignă specială care atestă autenticitatea.



Figură 15 - O pagină GitHub oficială

Acest concept este împrumutat de la site-urile de socializare, unde, pentru a preveni furtul de identitate, vedetele și persoanele publice îți pot adăuga sigla de "verified" în urma unui proces de acreditare.

Conform site-ului gitstar-ranking.com, cele mai populare companii de pe github sunt: Microsoft (care a și cumpărat platforma), Google, Facebook, apache, alibaba, vuejs, tensorflow, freeCodeCamp.

Pe lângă profilul utilizatorului, ne putem ghida și după cei 3 indicatori de popularitate: numărul de următori, numărul de stele și numărul de clone.

În cazul librăriilor foarte mari, cum este Angular, observăm că aceste numere sunt foarte mari, de ordinul miielor.



Figură 16 - Reputația unui repository

În cazul librăriilor mai mici și mai nișate (cum este cazul celor dezvoltate special pentru un anumit framework: React, Angular, Vue, etc.), în general tot ce trece de 500 de stele este acceptat ca fiind un produs destul de folosit și de stabil. Poate există și clone ale unui repositoriu, un repository care a fost clonat de multe ori și este în dezvoltare activă, motivul pentru care am dorit să facem o clonă este să îmbunătățim acel produs.



### 3.2.1.3 Download-uri săptămânale prin NPM

NPM (Node Package Manager) este cel mai popular manager de pachete pentru javaScript, atât pentru client-side cât și pentru server-site, el fiind managerul de pachete standard pentru Node.js și javaScript în general.

Practic fiecare limbaj are unul sau mai multe managere de pachete, iată câteva exemple:

- JAVA: Maven
- C#: nuGet
- Python: PIP
- javaScript: NPM, Yarn, Bower
- PHP: Composer

Aproape toate framework-urile de javaScript au dependințele configurate folosind NPM.

### 3.2.1.4 Dimensiunea librăriei

În mod evident, cu cât o librărie de javaScript este mai mare, cu atât va dura mai mult să fie descărcată în browser și va impacta performanța.

### 3.2.1.5 Complexitate

Librăriile javaScript pot deveni extrem de complexe, javaScript fiind un limbaj care a devenit foarte popular în ultimii ani. Atunci când alegem o librărie trebuie să găsim un compromis între complexitate și ușurința de folosire. Cele mai complexe libării nu sunt neapărat cele mai potrivite. Highcharts spre exemplu, are o complexitate mult prea mare pentru ce ne dorim noi să reprezentăm.

### 3.2.1.6 Documentație

În cazul librăriilor, documentația este esențială în luarea deciziei, din nefericire, există multe produse software foarte bune dar care au o documentație slabă. Documentația este cu atât mai bună dacă oferă și exemple practice sau demo-uri.

### 3.2.1.7 Suport dedicat pentru Angular

Deși multe librării javaScript se integrează destul de bine, este de preferat să alegem o librărie special făcută pentru framework-ul nostru de bază sau care să se integreze bine cu acesta. Din păcate, unele librării încă mai au dependențe de alte librării mai vechi (cum ar fi jQuery) și putem ajunge să avem conflicte.



Având în vedere comparația din tabela de mai sus, am ales să folosesc **ngx-graph** deoarece se integrează cel mai bine cu framework-ul Angular. Inițial am fost tentat să folosesc Cytoscape.js însă motivul pentru care l-am eliminat a fost dependența de alte librării cum ar fi jQuerry.js.

### 3.2.2 Implementarea folosind ngx-graph

Librăria ngx-graph se instalează folosind utilitarul NPM folosind următoarea comandă

```
npm install @swimlane/ngx-graph --save
```

Următorul pas este să adăugăm pachetul în modulul nostru Angular.

```
@NgModule({
  imports: [NgxGraphModule]
})
```

Că și rezultat, vom avea acces la directiva <ngx-graph> unde putem adăuga următoarele proprietăți:

- **nodes** : reprezintă lista nodurilor din graf (sub formă de JSON)
- **links**: reprezintă muchiile grafului (citite din lista de adiacență)

De asemenea, librăria dispune de metode pentru recentrarea grafului dar și pentru actualizarea modelului.

Că și design, putem modifica forma, culoarea și textul asociat fiecărui nod, precum și a arcelor ce leagă aceste noduri.



## 4. Arhitectura aplicației

Aplicațiile web sunt cu siguranță cel mai popular tip de produs software din ultimii ani. Ca orice produs ingineresc, putem defini arhitectura ca fiind structura de nivel superior al sistemului. Arhitectura unei aplicații descrie structura sistemului dar și cum interacționează componentele sale.

Principalul scop al arhitecturii software este ușurința cu care putem menține aplicația pe viitor. De aceea, există o serie de principii care s-au dovedit eficiente în construirea unei aplicații ușor de menținut; unul dintre aceste fiind separarea aplicației.

În primul rând, aplicația noastră este separată în back-end și front-end. Practic avem două aplicații care comunică între ele. Pe lângă această separare de nivel înalt, avem și alte nivele de separare în cadrul celor două aplicații.

### 4.1 Separarea Preocupărilor – SoC (Separation of Concerns)

Cel mai de bază principiu este cel al separării preocupărilor [26] care recomandă ca un produs software ar trebui separat în bucăți în funcție de sarcinile pe care le îndeplinește.

Din punct de vedere arhitectural, aplicațiile se pot construi ținând cont că va exista mereu o logică de business, o parte de infrastructură și o interfață logică. Un produs ar trebui să fie separat cel puțin pe aceste 3 nivele.

Prin folosirea separării, obținem un cod mai ușor de testat, mai ușor de înțeles și în primul rând, mai ușor de modificat. Teoretic, dacă separarea este suficient de bună, putem oricând înlocui un întreg nivel.

#### 4.1.1 Modelul Client-Server ca și fundament al tehnologiilor web

Internetul a adus schimbări revoluționare în lumea tehnologiei, conectând întreg globul pământesc. Pentru a putea realiza acest lucru, a fost nevoie de implementarea unei arhitecturi client-server. În acest model, un calculator central (numit server) găzduiește, distribuie și controlează majoritatea resurselor și serviciilor folosite de către clienți. Acest model a evoluat atât de mult încât Internetul a devenit un spațiu virtual de lucru, un server putând să răspundă cerințelor mai multor clienți iar clienții putând fi conectați la mai multe servere simultan.

Un exemplu care a devenit consacrat este acela în care clientul, reprezentat de un browser cere resurse de la un server, folosind protocolul HTTP. În modelul tradițional, serverul web se ocupa de generarea paginilor HTML pe care le prezintă mai apoi clientului. Serverul poate întoarce diverse tipuri de conținut (text, imagini, pagini HTML) dar poate pasa



responsabilitatea către client să proceseze aceste informații; de aici și popularitatea limbajului javaScript.

Deoarece browserul poate să execute cod javaScript, un mod de lucru pe care îl vedem tot mai des este acela în care serverul web trimite fișierele inițiale ale aplicației javaScript și scapă de responsabilitatea desenării(randării) interfeței grafice; altfel, serverul execută minimul necesar de efort și îi rămâne doar responsabilitatea de a transmite date într-un mod cât mai puțin costisitor, de obicei JSON.

#### 4.1.2 Tipul de arhitectură MVC și variațiuni ale acestuia

Probabil cel mai cunoscut şablon arhitectural al zilelor noastre, mai ales în contextul aplicațiilor web, MVC (Model-View-Controller), după modelul majorității conceptelor tehnologice; își are originile în mediul academic și datează încă de prin anii '70.

Începând cu anii 2000, tiparul MVC a fost popularizat de mai multe framework-uri web cum ar fi: Spring(JAVA), Django(Python) sau Rails(Ruby). Putem spune că orice limbaj care se respectă, are cel puțin o implementare în stilul MVC sau variațiuni ale acestuia.

În universul Microsoft, anul 2009 a fost marcat de lansarea framework-ului ASP.Net MVC, care între timp a fost rescris în cadrul proiectului .Net Core (vezi subcapitolul 5.1.1 ).

Dacă până acum arhitectura MVC era foarte populară pentru aplicațiile care se executa pe server, mai nou și aplicațiile în javaScript implementează această arhitectură. Mulțumită tehnologiei AJAX, aplicațiile pot sincroniza bucăți din interfață (view) folosind un model care este actualizat de către o logică de business(controller).

#### 4.1.3 Arhitectura stratificată

Cunoscută și sub numele de Arhitectură N-Tier [29] împarte aplicația în mai multe straturi logice sau fizice. Aceste straturi sunt un mod de a separa responsabilitatea și de a gestiona dependențele, fiecare nivel având o responsabilitate specifică. Straturile de nivel superior pot folosi serviciile oferite de straturile inferioare dar nu și invers.

Cel mai des, întâlnim arhitectura în 3 straturi, unde avem [27]:

- Stratul superior de prezentare
- Stratul de mijloc unde întâlnim logica de business
- Stratul operațiunilor asupra bazei de date

Am fi tentați să spunem că și MVC este o arhitectură 3-tier însă diferența este că în MVC datele(modelul) comunică în mod direct cu view-ul, pe când în 3-tier, trebuie să trecem mereu prin stratul de mijloc pentru a ajunge la stratul cel mai de jos al datelor. Practic 3-tier este o arhitectură liniară pe când MVC este o arhitectură triunghiulară (toate cele 3 elemente comunică între ele).



#### 4.1.4 CQRS

CQRS (Command and Query Responsibility Segregation) [28] este un alt şablon arhitectural care separă operaţiunile de citire de cele de scriere pentru a maximiza performanţa. Tradiţional, acelaşi context era folosit atât pentru operaţiunile de scriere, cât şi cele de citire. Dar în practică s-a dovedit că volumul acestor două tipuri de operaţiuni nu este egal. În modelul CQRS, aceste două tipuri de operaţiuni sunt executate în contexte diferite, astfel operaţiunile de citire (queries) nu modifică niciodată datele şi sunt foarte rapide iar operaţiunile de scriere (commands) se pot executa asincron.

În .net, librăria MediatR este cel mai popular mod de a implementa acest şablon.

### 4.2 Arhitectura REST pentru designul unui API

Un API (Application Programming Interface) este un termen oarecum abstract, dar, în termeni simpli, un set de funcţii ce permit altor aplicaţii să manipuleze datele. API-ul nu este reprezentat de către baza de date ci reprezintă mai degrabă o poartă de acces către aceasta.

În cazul API-urilor Web, stilul REST a devenit cel mai adoptat, deşi, vom vedea că nu toate API-urile respectă cu strictete regulile impuse de acesta. Nu este însă singurul mod în care putem construi API-uri web, chiar şi

Nu putem vorbi despre API-uri web fără să definim ce este o resursă. O resursă este similară cu instanţa unui obiect unde avem doar câteva metode publice pe care le putem apela. Un exemplu, de astfel de metode ar fi verbele HTTP (descrise mai jos). Resursele pot fi grupate în colecţii şi pot depinde de alte resurse. O resursă poate fi accesată printr-un endpoint (url) [9].

#### 4.2.1. Constrângerile REST

REST (REpresentational State Transfer) este în primul rând un standard, dar şi un stil arhitectural. Profesorul Roy Fielding a popularizat standardul REST în lucrarea lui de doctorat publicată în anul 2000. [9]

În loc să impună decizii asupra arhitecturii, preferă să definească un set de constrângeri la care sistemul să adere. În felul acesta, detaliile de implementare se pot schimba ulterior, cu condiţia să păstreze avantajele specifice acestei arhitecturi. Pe scurt, arhitectura REST defineşte 5 constrângeri obligatorii şi una optională. Doar dacă respectăm aceste constrângeri putem spune că API-ul nostru este RESTfull, altfel, putem afirma cel mult că avem un API REST-based. Mulți programatori începători fac confuzie între acești termeni şi numesc orice API HTTP un "RESTfull API".



#### 4.2.1.1 Modelul Client-Server în REST

După cum am precizat în arhitectura client-server este una dintre cele mai populare paradigmă în comunicarea dintre două sisteme. Modelul REST folosește acest concept fundamental ca și fundament, pentru a putea separa logica de server de cea specifică clientului. Spunem că API-ul produce niște resurse, iar clientul le consumă.

#### 4.2.1.2 Fără stare (stateless)

Serverul nu se bazează niciodată pe informațiile venite dintr-un request anterior, practic nu există o logică de sincronizare între client și server, clientul este responsabil să furnizeze toate informațiile de care are nevoie serverul în interiorul fiecărui request. Evident, acest lucru implică un trafic mai mare pe rețea însă avantajul este că mai multe servere paralele pot rezolva requestul iar clientul nici măcar nu trebuie să știe acest lucru.

#### 4.2.1.3 Cache

Cache-ingul este o tehnică foarte populară care sporește considerabil performanța aplicațiilor. Serverul îi poate spune clientului pentru cât timp ar trebui să păstreze datele, astfel, clientul știe că nu este nevoie să ceară o actualizare a datelor. Evident, clientul poate să ignore această informație și să ceară datele oricum, dar, de regulă, în aplicațiile web, se va executa un request AJAX doar dacă nu există deja informațiile în cache-ul browserului.

#### 4.2.1.4 Arhitectura stratificată în REST

După cum am precizat în introducerea capitolului, arhitectura stratificată permite adăugarea sau înlocuirea unor nivele arhitecturale, fără a影响a funcționalitățile existente.

#### 4.2.1.5 Interfață Uniformă

Cea din urmă constrângere este de fapt cel mai greu de implementat. Scopul acestei constrângeri, este din nou, să decoupleze clientul de server dar să ofere un contract de comunicare. Există 4 subpuncte ale constrângerii de interfață uniformă:

- Se bazează pe resurse stabile (url-uri care nu se schimbă)
- Se folosesc reprezentări (ex. content-type) cum ar fi JSON, XML, etc
- Există mesaje descriptive (ex. coduri răspuns)
- HATEOAS (Hypermedia as the Engine of Application State) care presupune că resursele sunt însotite de informații de navigare către alte resurse sau sub-resurse



#### 4.2.2. API-uri HTTP ce implementează REST

Deși modelul REST nu necesită neapărat folosirea protocolului HTTP, fiind o arhitectură independentă de protocol, în contextul aplicațiilor web, acesta a devenit standardul preferat. După cum am menționat în capitolul anterior, pentru a putea spune că avem un API RESTFull, trebuie să respectăm toate cele 5 principii.

O altă scară pe care o putem folosi pentru a ne da seama cât de RESTfull este aplicația noastră este Modelul de maturitate Richardson [34].

Acest model identifică 4 nivele de maturitate:

- 0 – Nivelul haotic XML
- 1- Nivelul resurselor dedicate (URL uri specifice fiecărei acțiuni)
- 2- Nivelul HTTP: folosirea verbelor
- 3- Nivelul HATEOS: furnizarea link-urilor pentru navigarea resurselor și pentru executarea acțiunilor suplimentare

Conform acestui model, folosirea protocolului HTTP împreună cu verbele corespunzătoare, ne plasează abia pe poziția 3 din 4.

#### 4.2.3. CRUD folosind verbe HTTP

Protocolul HTTP este indiscutabil, modalitatea preferată de a comunica prin Internet. Clientul și serverul comunică folosind mesaje HTTP. Există două tipuri de mesaje [16]:

- Request – mesaj trimis de către client pentru a declanșa o acțiune
- Răspuns – mesajul trimis clientului de către server

ACEste mesaje nu sunt altceva decât niște text ASCII aranjat pe mai multe linii. Prima linie conține descrierea mesajului. În cazul request-ului, pe această linie găsim verbul http urmat de calea către resursă. Iar în cazul răspunsului, pe prima linie găsim statusul răspunsului (cod + mesaj).

Prin convenție, verbe HTTP specificate în mesajul de request îndeplinesc anumite roluri. Aceste roluri se pot împăra la operațiunile principale ce se pot executa supra unei baze de date : CRUD (Create, Read, Update, Delete).

Operație asupra datelor	Verb HTTP
Create (creare)	POST
Citire (read)	GET
Actualizare(update)	PUT și PATCH(actualizare parțială)
Ștergere (delete)	DELETE

Tabel 6 - Paritatea operațiunilor CRUD cu verbele HTTP



#### 4.2.4. Statusuri de răspuns HTTP

Protocolul HTTP folosește statusuri (coduri numerice) care indică dacă un request a fost îndeplinit cu succes sau nu. Există 5 grupe de coduri de răspuns [16]:

- Răspunsuri informative (100–199)
- Răspunsuri de succes (200–299)
- Răspunsuri de redirectare (300–399)
- Erori la nivelul clientului (400–499)
- Erori la nivelul serverului (500–599)

#### 4.2.5 Documentarea API-ului folosind OpenAPI

Specificațiile OpenAPI definesc un standard care descrie un API REST pentru a putea fi înțeles, fără a oferi acces la codul sursă. Un document de specificație OpenAPI poate fi folosit și de către unelte de generare a codului de client în diverse limbaje de programare. O astfel de unealtă, folosită în proiectul nostru, este NSwag, care generează atât documentația într-un format JSON ce poate fi prezentat într-o interfață web, cât și codul TypeScript care poate fi folosit de către serviciile Angular pentru a consuma resursele API-ului.

### 4.3 Modelul „Clean Architecture” definit de Uncle Bob

În cartea sa, Clean Architecture: A Craftsman’s Guide to Software Structure and Design [6], Robert C. Martin, cunoscut în industrie și ca Uncle Bob, ne prezintă un model de arhitectură pragmatic, numit simplu “Clean Architecture” (arhitectura curată). Acest model a evoluat din arhitectura Onion (ceapă) și din cea hexagonală.

Principiul de bază al acestui arhitecturi, este că nivelele(straturile) superioare de arhitectură depind de nivelele inferioare, iar cele inferioare nu știu nimic despre nivelele superioare. Acest lucru ne permite să schimbăm destul de ușor componentele sistemului și să extindem individual fiecare strat. Această separare ne permite și să scriem mai ușor teste unitare, pentru a oferi un nivel de calitate mai bun.

De asemenea nivelele inferioare sunt independente de versiunea framework-urilor. Nivelul cel mai de jos, al entităților de domeniu, nu ține cont că aplicația folosește EntityFramework sau un alt ORM.

Practic, acest model de arhitectură ne oferă libertatea de a folosi tehnologii diferite pentru fiecare nivel al aplicației.



#### 4.3.1. Concepte de Clean Code și bune practici

Tot Uncle Bob, în cartea sa “Clean Code: A Handbook of Agile Software Craftsmanship” [7] ne prezintă niște principii care s-au dovedit a fi eficiente în a crea un cod care să fie ușor de menținut. Pe lângă principiile SOLID care ar trebui să fie literă de lege pentru orice programator, există și principii legate de design, denumirea variabilelor și metodelor dar și despre cum ar trebui să scriem testele automate.

Uncle Bob descrie un concept foarte interesant, acela de “Code smells”, însemnând practic că “ne miroase” ceva cum că nu avem de a face cu un cod curat, acestea sunt:

- Rigiditate: codul este greu de modificat
- Fragilitate: programul generează erori neprevăzute la cele mai mici schimbări
- Imobilitate: nu se pot refolesi componente sau părți din cod
- Complexitate inutilă
- Repetare inutilă
- Opacitate: codul este greu de înțeles

#### 4.3.2. Implementarea arhitecturii “Clean Code” în .Net

Un programator care a popularizat acest model arhitectural în universul .Net este Jason Taylor, care, în blogul lui, a demonstrat cum se aplică această arhitectură în practică. [34] Jason a prezentat această arhitectură în cadrul conferinței internaționale NDC. El a creat și un template care conține deja structura aplicației și dependențele între proiecte.

Acest template se poate instala folosind CLI-ul .Net Core (descriș în subcapitolul 5.1) folosind următoarea comandă:

```
dotnet new --install Clean.Architecture.Solution.Template
```

Apoi, putem crea o soluție nouă de .Net folosind următoarea comandă

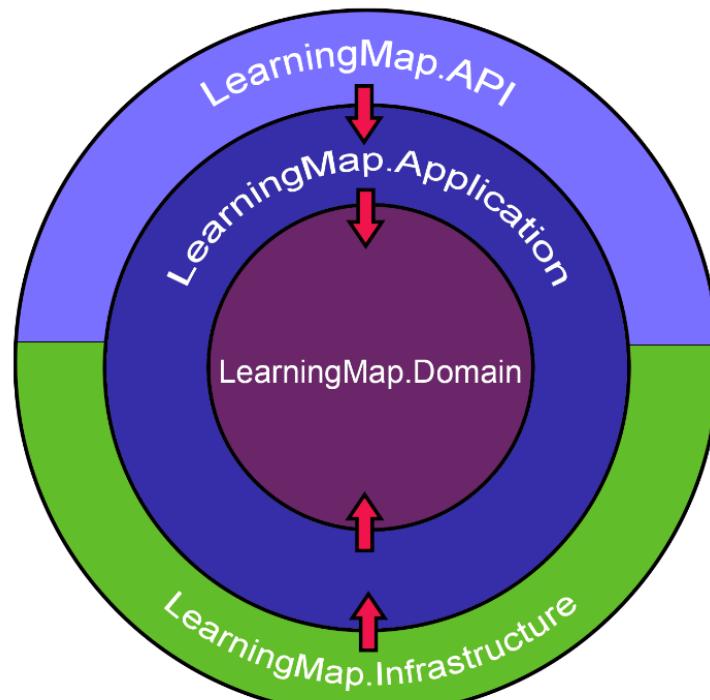
```
dotnet new ca-sln
```

Fiecare nivel este reprezentat de un proiect în Visual Studio.

Pentru a respecta principiul independenței de framework, în nivelul inferior, al entităților, tipul proiectului este .Net Standard (restul folosind .Net 5), ne fiind limitați nici măcar la a folosi .Net Core, acest proiect putând fi refolosit chiar și de o aplicație .Net Framework. Nivelul de infrastructură conține de fapt implementarea specifică pentru Entity Framework. Nivelul superior, de prezentare, conține de fapt WebAPI-ul cu controlerele aferente; logica din controller însă, este încapsulată în proiectul „Application” (acest nivel se mai numește și „Application Core”).



În aplicația noastră, structura proiectelor împreună cu dependențele între acestea arată astfel:



Figură 17 - Arhitectura API-ului

#### 4.4 Securitate

În orice aplicație, securitatea este esențială, cu atât mai mult într-o aplicație disponibilă public pe internet unde comunicarea este de neconceput fără un canal sigur. Un element important al securității este protecția datelor, mai ales când avem de a face cu datele personale ale utilizatorului.

În aplicația noastră, nu avem neapărat date sensibile însă avem date cu caracter personal, cum ar fi: numele propriu, emailul, telefonul și vârsta. Pe lângă acestea, avem și date de interes general, cum ar fi cursurile pe care le urmează un utilizator. Aceste date, ar putea fi folosite în scopuri comerciale pentru a vinde utilizatorului cursuri de pe platforme specifice. Din acest motiv, expunerea acestor date ar avea un impact negativ asupra utilizatorului și, implicit, asupra aplicației. Dacă vrem să monetizăm aceste date în mod intenționat, suntem obligați să informăm utilizatorul despre acest lucru. Însă această aplicație nu are ca și scop valorificarea acestor date și atunci rămâne datoria noastră să implementăm protocoale de securitate și politici de siguranță.

##### 4.4.1. HTTPS

În modelul TCP/IP, derivat din modelul OSI, există 4 nivele, la nivelul superior (nivelul Aplicație) găsim protocolul HTTP [16]. Tehnic vorbind, HTTPS nu este un protocol separat de HTTP ci este o extindere a acestuia.



#### 4.4.1.1 Prezentarea protocolului HTTPS

Protocolul HTTPS extinde protocolul clasic HTTP, încapsulând transferul de date într-un flux securizat SSL/TLS. Nativ, protocolul HTTP transmite datele în clar, ceea ce înseamnă că cineva s-ar putea plasa între client și server și ar putea citi și înțelege aceste date [16]. Standardul de implementare al protocolului HTTPS a fost definit în RFC 2818 în anul 2000 și folosește portul 443, spre deosebire de HTTP care folosește portul 80. Însă nu suntem limitați să folosim doar aceste porturi. Scopul protocolului este să ofere autentificare, criptare și integritatea datelor.

Practic, protocolele SSL și TLS sunt cele care se ocupă de autentificare și criptare. TLS este succesorul lui SSL însă deseori le vedem folosite ca și SSL/TLS. În linii mari, TLS se ocupă de stabilirea legăturii (handshake) între două entități folosind tehnica de autentificare cu două chei: o cheie publică și una privată.

Serverul este cel care detine o cheie publică pe care clientul (browserul) o poate folosi ca să confirme că serverul este într-adevăr cine pretinde a fi (este deținătorul acelui nume). Există autorități globale care semnează aceste certificare ale serverului folosind cheia lor privată, cum ar fi SSL.com.

#### 4.4.1.2 Generarea unui certificat local pentru dezvoltare

Protocolul HTTPS a devenit atât de adoptat încât nu mai avem nici o scuză să lucrăm fără ele, nici nu se mai pune problema să nu avem un certificat de autentificare atunci când aplicația noastră web este publicată pe internet. De aceea și când lucrăm local, trebuie să ne asigurăm că aplicația noastră funcționează bine cu acest protocol.

Pentru că lucrăm local, putem însă să generăm un certificat pe care să-l semnăm tot noi și apoi să-l instalăm pe calculatorul nostru.

Un utilitar excelent pentru a genera certificate este openssl, ce se instalează odată cu programul GIT pe care l-am folosit pentru versionarea codului sursă.

Comanda openssl se poate apela din linia de comandă pentru a genera certificatul împreună cu cheia sa privată.

Pentru a reproduce cu ușurință acest proces de generare și în alte medii, am creat un script de PowerShell care va genera certificatul și îl va instala pentru utilizatorul curent din Windows.

Că și configurare, putem completa detaliiile despre autoritatea care a semnat certificatul (în cazul nostru este propriul calculator), putem seta valabilitatea acestuia (pentru ușurință, am setat această valoare la 9 ani) și pentru ce nume de server s-a generat (în cazul nostru este vorba de localhost).

Fișierul de configurare `learningmap-localhost.cnf` va conține informații despre autoritatea care a emis certificatul de securitate:

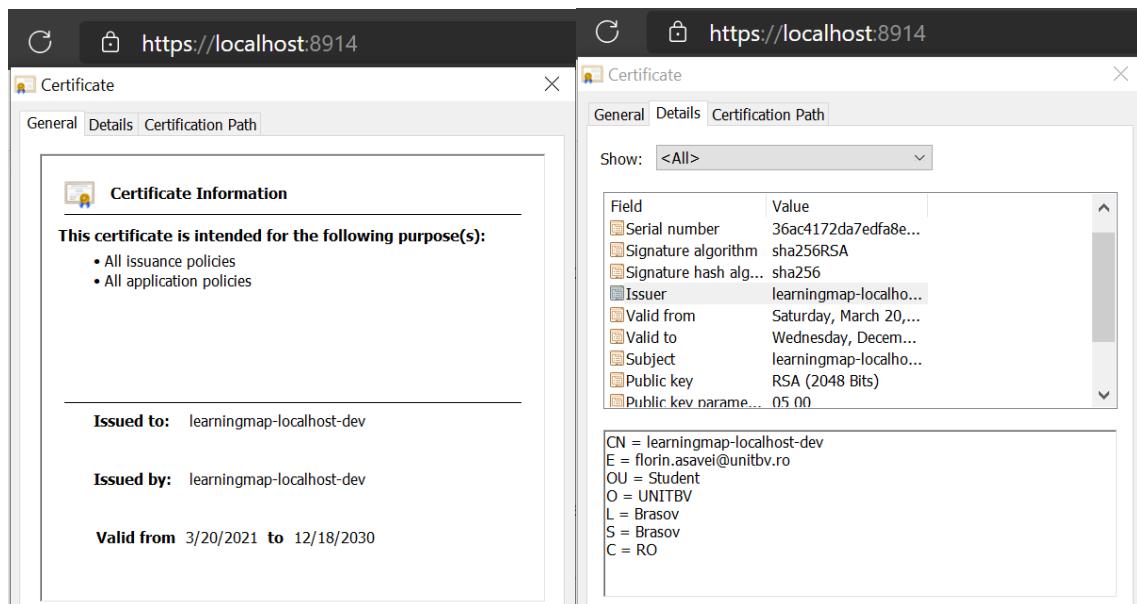
```
[req]
...
[dn]
E = florin.asavei@student.unitbv.ro
O = UNITBV
OU = Student
C = RO
ST = Brasov
L = Brasov
CN = learningmap-localhost-dev
```

Iar cele două comenzi de Powershell folosite sunt următoarele:

```
openssl req -nodes -x509 -sha256 -days 3285 \
-config learningmap-localhost.cnf \
-new -out learningmap-localhost.crt \
-newkey rsa:2048 -keyout learningmap-localhost.key
```

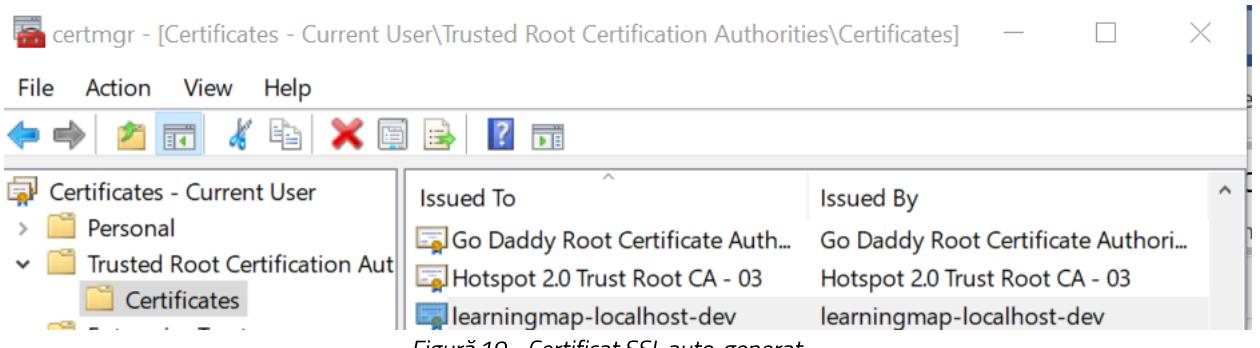
```
Import-Certificate -FilePath .\learningmap-localhost.crt \
-CertStoreLocation cert:\CurrentUser\Root
```

Astfel, atunci când pornim aplicația, browserul nu ne mai avertizează că accesăm un site nesecurizat și putem observa informațiile introduse de noi.



Figură 18 - Certificat SSL în browser

Acest lucru este posibil deoarece am instalat local acest certificat, acceptând validitatea acestuia.



Figură 19 – Certificat SSL auto-generat

O alternativă ar fi fost să schimbăm setările browser-ului (Chrome) ca să accepte certificate nesigure (auto semnate) însă ne-am fi expus în cazul în care folosim browserul și pentru a accesa alte site-uri. Ar fi trebuit să fim atenți să activăm și să dezactivăm această setare după fiecare rulare locală a aplicației. Este vorba despre chrome://flags/#allow-insecure-localhost

Evident, acest certificat nu ar funcționa dacă am publica site-ul pe internet, ar trebui ca toți utilizatorii să-ți instaleze acest certificat local.

#### 4.4.1.3 Obținerea unui certificat gratuit

După cum am văzut mai sus, e din ce în ce mai greu să ignorăm elementele de securitate, pur și simplu nu mai avem voie să folosim protocolul HTTP nesecurizat, browserele vor bloca aceste site-uri din oficiu și vor încerca să acceseze varianta HTTPS, care nu e altceva decât HTTP + SSL/TLS.

Din fericire, putem obține certificate SSL/TLS gratuit. Un astfel de furnizor de certificate este LetsEncrypt. Certificatele generate de ei sunt valabile 90 de zile însă putem să le reînnoim folosind un proces automatizat.

Dacă folosim un serviciu de Cloud, cum ar fi Azure, acest proces este extrem de simplu [37]. Practic, în Azure avem din oficiu un certificat SSL gratuit.

#### 4.4.2. OpenId Connect și OAuth2

După cum am prezentat în subcapitolul 2.3 (Managementul identităților și al utilizatorilor), protocolele care stau la baza acestor implementări de autentificare și al accesului sunt OpenId Connect și OAuth2. În acest subcapitol vom intra puțin mai în detaliu în specificațiile acestor protocole.

##### 4.4.2.1. Autentificare versus Autorizare, concepte generale

De multe ori, programatorii începători fac confuzie între autentificare și autorizare, acestea fiind două concepte care se leagă unul de altul dar servesc scopuri diferite.



Autentificarea se referă la faptul că un utilizator este cine afirmă el că este. Un exemplu clasic aici este autentificarea clasică folosind username și parolă. Făcând o analogie cu domeniul hotelier, atunci când un client vrea să intre într-un hotel, el trebuie să-și prezinte buletinul la recepție pentru a dovedi că el este într-adevăr persoana care a făcut rezervarea. Autorizarea se referă la ceea ce poate un utilizator autentificat deja să facă sau nu, mai exact ce drepturi are acesta. Revenind la analogia cu hotelul, după ce clientul prezintă un document de autentificare (buletin, pașaport, etc), acestuia i se oferă o cheie de acces cu care va putea deschide doar ușa camerei unde este cazat, nu și alte uși.

Acești 2 termeni: autentificare și autorizare, se confundă deoarece în unele protocoale cum ar fi OAuth2, doar pentru că aveam o cheie de acces, se consideră că deja suntem autenticați. Era ca și cum un om care a obținut o cheie de acces, nu mai trebuia să prezinte buletinul niciodată, pentru că se presupunea că nu poți obține chei de acces fără buletin. Dar dacă totuși cheia de acces este furată? Sau dacă cineva a reușit cumva să treacă de recepție pretinzând că este altcineva?

#### 4.4.2.2. Standardul modern OpenId Connect bazat pe OAuth 2.0

OpenId Connect (OIDC) este un protocol ce adaugă partea de autentificare peste partea de autorizare oferită de OAuth 2.0. În timp ce OAuth 2.0 dictează regulile de acces la o anumită resursă, OIDC se ocupă de partea de autentificare [36].

Protocolul implică existența unui server de autentificare (în cazul nostru este vorba despre Microsoft Identity) care se ocupă de întreg procesul de autentificare folosind anumite flow-uri.

Unul dintre scopurile OIDC este să ne ofere același login pentru site-uri multiple.

#### 4.4.2.3. Flow-uri de autentificare

După cum am văzut în subcapitolul (2.3.4.1 Microsoft Identity pentru SPA-uri), aplicația Angular se autentifică cu Microsoft Identity Platform printr-un request HTTP de tip GET către "https://[url-backend]/Identity/Account/Login" adăugând o serie de parametrii (query params), printre care:

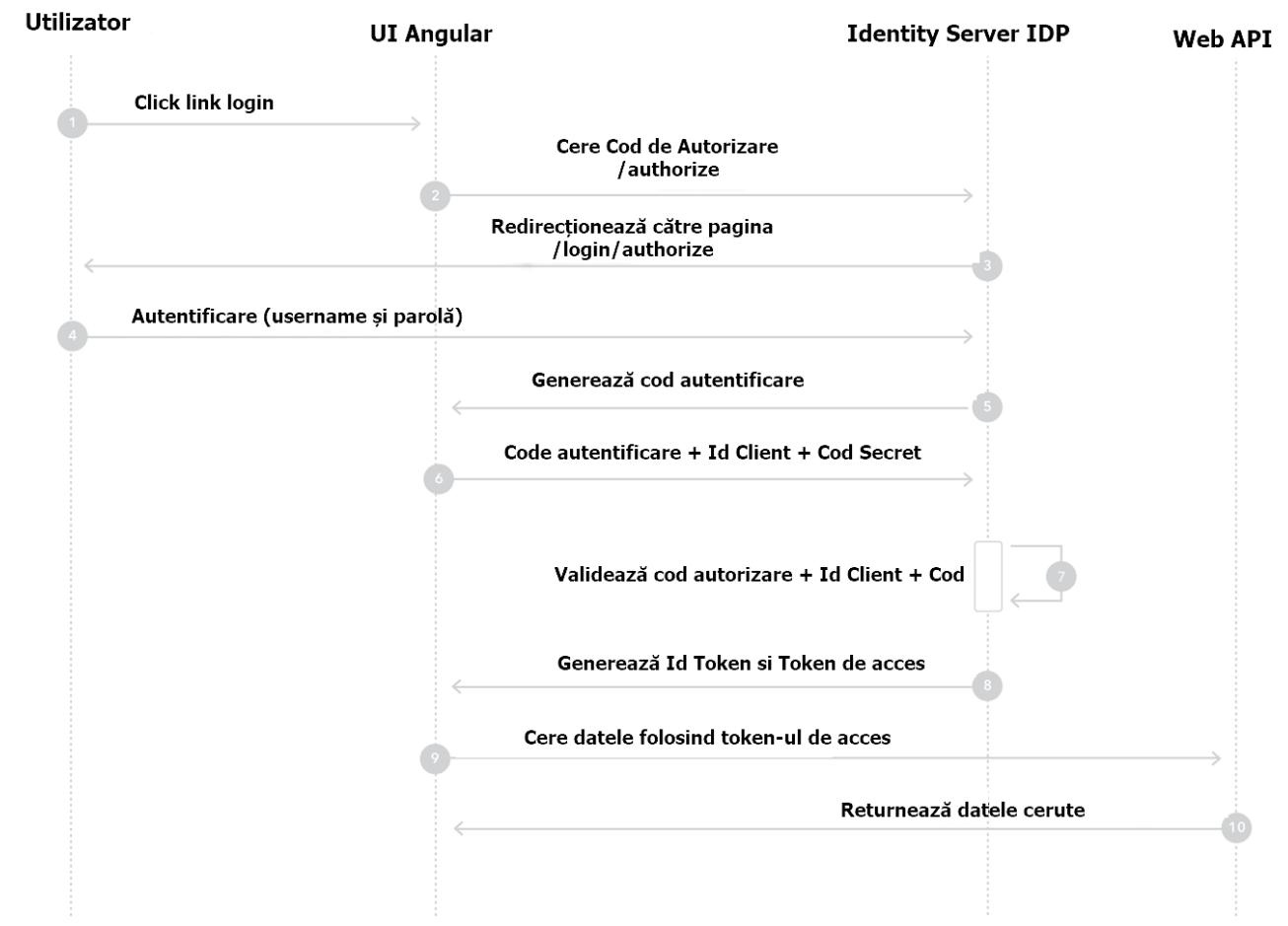
- `response_type = code`
- `scope = openid`

Conform standardului rfc6749 [31], această combinație pornește flow-ul de autentificare numit Authorization Code. Acest flow se folosește pentru a adăuga autentificare și autorizare în majoritatea aplicațiilor web. Acest flow permite obținerea unui token de acces JWT(descriș în subcapitolul 2.3.1.3) în cel mai sigur mod posibil. În linii mari, în urma procesului de autentificare, se obține un token de autentificare ce este mai apoi schimbat cu un token de autorizare. Operațiunea de generare și validare a acestor două tipuri de



tokeni este realizată de către un IDP (Identity Data Provider) care în cazul nostru este Microsoft Identity (prezentat în subcapitolul 2.3.2).

Următoarea schemă explică în detaliu cum funcționează Authorization Code Flow [31]



Figură 20 - Flow autentificare și autorizare



## 5. Tehnologii folosite

Toate tehnologiile folosite au licențe software open-source permisive de tip MIT.

### 5.1 .Net 5

Lansată în Noiembrie 2020, sub denumirea de .Net 5 este ultima și cea mai semnificativă și așteptată versiune a platformei .Net oferită de Microsoft. Tehnic vorbind, este o continuare a proiectului .Net Core, care a însemnat o rescriere completă a platformei deja consacrate .Net Framework care se află momentan la versiunea 4.8 și urmează să intre în menenanță.

Pentru a evita confuziile de denumire, Microsoft au decis că vor sări de la versiunea .Net Core 3.1 direct la .Net 5, pentru a evita confuziile între .Net Framework și .Net Core. Practic, de acum înainte vom avea un singur .Net.

Însă .Net 5 nu ne aduce doar o denumire nouă ci unifică toate componentele framework-ului într-o singură platformă. Fie că dorim să dezvoltăm aplicații web, mobile, desktop sau IoT, vom folosi aceeași platformă de bază, peste care vom adăuga pachete. Deși pare a fi succesorul lui .Net 4, .Net 5 este de fapt derivat din .Net Core iar vechiul .Net se numește acum .Net Framework. [11]

#### 5.1.1 Ecosistemul .Net

Microsoft a început să dezvolte Framework-ul .Net la sfârșitul anilor '90, prima versiune de .Net apărând în februarie 2003. Această versiune a introdus și limbajul de programare C#, pe lângă alte limbi cum ar fi F# și Visual Basic [3].

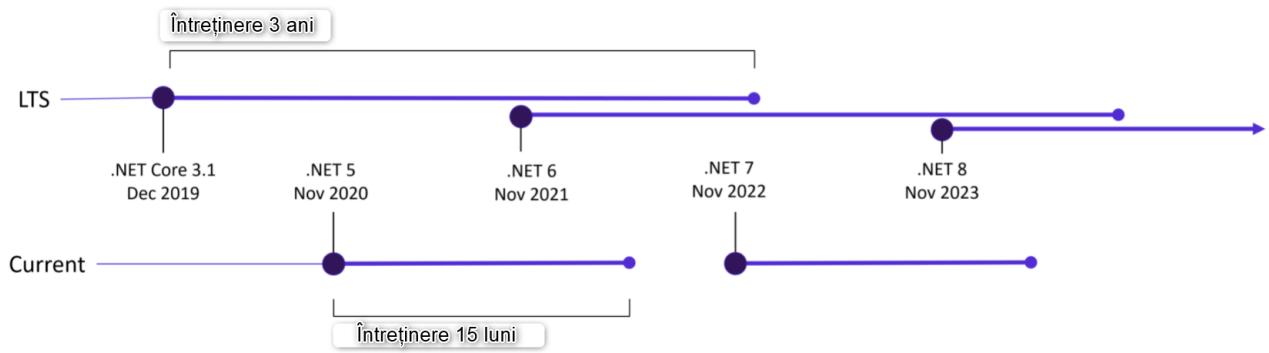
Limbajul C# a fost inventat de către danezul Anders Hejlsberg, cel care a fost de asemenea și autorul original al limbajului Turbo Pascal și arhitect șef al limbajului Delphie [10]. Andreas este în continuare arhitectul principal al limbajului dar se ocupă și de dezvoltarea altor limbi în cadrul Microsoft, în principal TypeScript, un limbaj pe care l-am folosit în aplicația de front-end, în cadrul framework-ului Angular.

C# se află într-o evoluție continuă, momentan aflându-se la versiunea 9 (lansată în Septembrie 2020) odată cu .Net 5 și se află în topul celor mai folosite limbi de programare (deși nu este singurul limbaj suportat de .Net, amintim și F# sau VB.Net).

Ca și tipuri de versiuni, în funcție de suportul oferit, avem două mari categorii

- Versiunea curentă pentru care se oferă suport până când apare următoarea versiune și 3 luni după
- Versiunea LTS (Long Term Support) pentru care se oferă suport pentru minim 3 ani sau minim un an după ce apare următoarea versiune LTS

Conform planurilor publicate de Microsoft, avem următoarea planificare:



Figură 21 - Planificarea versiunilor .Net

Pentru lucrarea în cauză, am fi putut folosi .Net Core 3.1 pentru a beneficia de suportul Microsoft până în 2022. Doar pentru că o versiune ieșe din suport activ, nu înseamnă că nu o mai putem folosi, ci doar că ieșe din perioada de întreținere activă.

Motivul pentru care am ales versiunea curentă în locul unei versiuni LTS a fost acela de a experimenta cu ultima versiune.

Pentru a nu face confuzie de termeni, când spunem .Net Core ne referim la runtime-ul .Net (similar cu JAVA VM) iar când spunem ASP.Net Core ne referim la framework-ul web.

### 5.1.2. ASP.Net Core WebAPI

Conform statisticilor din 2020 făcute de cei de la StackOverflow [30], ca și framework-uri de dezvoltare pentru aplicații web, imediat după primele 3 poziții ocupate de framework-urile javaScript jQuery, React.js și Angular, urmează tehnologiile celor de la Microsoft: ASP.Net și ASP.Net Core. Nu ar trebui să ne surprindă faptul că tehnologiile mai vechi încă ocupă un loc fruntaș în listă, având în vedere costul de refactorizare al unei aplicații existente.

Dacă vrem să construim un API Web, soluția pare evidentă, ASP.Net(framework) în cazul aplicațiilor vechi și ASP.Net Core dacă pornim un proiect nou. De fapt, până și site-ul StackOverflow este construit cu ASP.Net.

Proiectul ASP.Net Core este versiunea open-source al lui ASP.Net, lansată în 2016, cu scopul de a funcționa și pe alte sisteme de operare în afară de Windows.

Dacă până acum, în .Net aveam proiecte diferite în funcție de tipul de aplicație pe care dorim să-o construim: MCV folosind Razor Pages sau Web API; în .Net Core aceste proiecte sunt unite într-un singur tip de proiect (aplicații web).

Astfel, dacă dorim să folosim partea de Web API, trebuie pur și simplu să injectăm motorul de randare Razor. De fapt, la bază, toate proiectele de .Net Core sunt aplicații tip consolă. O aplicație web nu este altceva decât o consolă care atunci când se lansează pornește și un server web.

În .Net Core acest server web intern se numește Kestrel [11].



### 5.1.3 EntityFramework Core

După cum am menționat în subcapitolul 2.2 (Faza de implementare a bazei de date), am folosit EntityFramework (prescurtat EF) pentru proiectarea bazei de date și în general pentru toate operațiunile ce țin de date. EF a fost introdus în ecosistemul .Net încă din 2008 fiind ORM-ul standard al platformei .Net [11].

În acest proiect, am folosit noua versiune, EF Core, lansată odată cu platforma .Net Core. Deși, conform celor de la Microsoft EF 6 suportă și .Net Core, această versiune nu mai este menținută în mod activ și deci se recomandă folosirea versiunii Core, aceasta din urmă având o performanță mai bună [38].

Urmând tendința Microsoft de a face totul open-source, nu ar trebui să ne surprindă că și acest ORM(object-relational mapper) de la Microsoft este tot open-source.

EF Core suportă mai multe tipuri de baze de date : SQL Server, SQLite, PostgreSQL, MySQL, Oracle și multe alte. Această funcționalitate este furnizată prin plugin-uri, practic vom instala un pachet NuGet corespunzător bazei noastre de date.

În aplicația noastră, am instalat pachetul Microsoft.EntityFrameworkCore.SqlServer pentru a comunica cu SQL server 2019 sau Azure SQL (care în spate este tot 2019).

## 5.2. Angular

Framework-urile bazate pe javaScript au explodat în ultimii ani. Cei mai mari 3 jucători din această categorie sunt React, Angular și Vue. Deși încă se mai folosește bătrânelul jQuery, acesta pierde rapid din popularitate din cauza problemelor de performanță cauzate de manipularea intensă a DOM-ului (Document Object Model).

Adevărata bătălie se dă între Angular (creat de Google inițial în 2010 și rescris complet în 2016) și React (creat de Facebook în 2013) [35]. Ambele framework-uri sunt open-source. Diferența majoră între acestea fiind următoarea: Angular vine la pachet cu mai multe sisteme cum ar fi : rezolvarea rutelor, injectarea dependențelor, menținerea stării obiectelor, etc; pe când React se bazează pe alte librării suplimentare pentru a îndeplini aceste sarcini. Altfel spus, Angular vine la pachet cu tot ce avem nevoie, pe când în React este responsabilitatea noastră să alegem librăriile externe potrivite.

Un alt aspect este ușurința cu care putem învăța un framework, fiind mai simplu, React se bucură de o popularitate sporită, fiind mai ușor de integrat în aplicații existente.

Avantajul principal al lui Angular e că folosește o arhitectură bazată pe componente reutilizabile ce pot evoluă independent, permitând o dezvoltare modulară. [15]



### 5.2.1. Ecosistemul javaScript/TypeScript

JavaScript este un limbaj de programare/scriptare interpretat cu tipuri dinamice, orientat pe obiect care suportă multiple paradigme de programare.

Prima versiune a limbajului (numită inițial Mocha) a fost creată de către Brandon Eich [35] în 1995 și a fost integrată în browserul Netscape. Ulterior, în 1997, limbajul a fost standardizat sub denumirea de ECMAScript, prefixul java impus de cei de la Sun Microsystems nefiind decât o strategie eșuată de marketing care în zilele noastre creează multă confuzie. Versiunea ECMAScript4 (2008) a fost de fapt cea care a revoluționat aplicațiile web, fiind un efort comun al celor de la Yahoo, Google și Microsoft.

Principalul avantaj al limbajului este că se poate executa nativ în browser, deși, mai nou, mulțumită limbajului Node.js, putem folosi JavaScript și pe sever. Un alt avantaj este că limbajul nu necesită compilare, fiind un limbaj interpretat. Însă acest lucru este și cea mai mare slăbiciune a sa, neavând compilare, nu putem avea erori de compilare (cum ar fi conversii de tipuri de date) și ne putem trezi cu aceste erori direct în aplicația lansată în producție.

Din nou, cei de la Microsoft salvează situația în 2012, tot sub coordonarea lui Anders Hejlsberg [41], creând limbajul TypeScript care este de fapt un super-set peste JavaScript care oferă tipuri de date statice, ca în majoritatea limbajelor de programare. Practic codul TypeScript este compilat în cod JavaScript ce poate fi mai apoi înțeles de browser.

Toate cele 3 framework-uri principale de JavaScript (Angular, React, Vue) folosesc TypeScript. În aplicația LearningMap se folosește versiunea 4 a limbajului TypeScript.

### 5.2.2. Versiunea curentă a framework-ului Angular

Începând cu 11 Mai 2021, Angular se află la versiunea stabilă (LTS) 11. Tendința este să se lanseze o nouă versiune o dată la șase luni, după care această versiunea să rămână în Long-Term Support (LTS) timp de un an. Practic, Angular 10 va ieși din ciclul de suport în Noiembrie 2021 [15].

Versiune	Statusul dezvoltării	Data publicării	Încetarea dezvoltării	Încetarea suportului LTS
➤ 12	Activ	12 Mai 2021	12 Noi 2021	12 Noi 2022
➤ 11	LTS	11 Noi 2020	11 Mai 2021	11 Noi 2022
➤ 10	LTS	24 Iun 2020	24 Dec 2020	24 Dec 2021
➤ 9	LTS	06 Feb 2020	06 Aug 2020	06 Aug 2021

Tabel 7 - Versiuni Angular [15]



### 5.2.3. Angular CLI

Angular ne oferă o aplicație utilitară, numită Angular CLI care poate fi folosită pentru a interacționa cu framework-ul din linia de comandă. Această aplicație este folosită inclusiv pentru a crea o aplicație nouă Angular. Acest utilitar se instalează la nivelul global al sistemului de operare, folosind NPM.

#### 5.2.3.1 Crearea unei componente noi folosind Angular CLI

Din rădăcina directorului corespunzător proiectului de Angular (acolo unde găsim fișierul package.json) vom lansa consola PowerShell și vom executa următoarea comandă:

```
ng generate component <nume-componenta>
```

Se recomandă ca versiunea de CLI să fie aceeași cu a framework-ului Angular.

## 5.3. Microsoft SQL Server

SQL Server este un sistem de gestionare a bazelor de date relaționale (RDBMS) dezvoltat și comercializat de Microsoft. Ca server de baze de date, funcția principală a SQL Server este stocarea și preluarea datelor utilizate de către alte aplicații (nu neapărat din aceeași familie de tehnologii Microsoft). [32]

Microsoft SQL Server este una dintre cele trei tehnologii de bază de date lider de piață, împreună cu baza de date Oracle și DB2 IBM. La fel ca alte software-uri RDBMS, Microsoft SQL Server este construită pe baza SQL, un limbaj de programare standardizat pe care administratorii de baze de date (DBA) și alți profesioniști IT îl folosesc pentru a gestiona bazele de date și pentru a interoga datele pe care le conțin. [35]

Pentru a interacționa cu baza de date se poate realiza folosind interfața grafică oferită de SQL Server Management Studio (SSMS). Putem folosi tot SSMS pentru a accesa și o baza de date publicată în cloud.

### 5.3.1. Avantajele bazelor de date relaționale

O bază de date relațională este structurată, ceea ce înseamnă că datele sunt organizate în tabele. De multe ori, datele din aceste tabele au relații între ele sau dependențe. O bază de date non-relațională este bazată pe documente, ceea ce înseamnă că toate informațiile sunt stocate într-o listă nestructurată, de exemplu, într-o bază nonSQL, se pot salva adresele poștale ale unui utilizator, deoarece, un utilizator poate avea mai multe adrese cu structuri diferite și nu am vrea să limităm schema aceasta. [40]



### 5.3.2. Microsoft SQL Server 2019

SQL Server 2019, a fost lansat pe 4 Noiembrie, 2019, și se bazează pe versiunile anterioare pentru a dezvolta SQL Server ca o platformă care oferă opțiuni de limbaje de dezvoltare, tipuri de date, medii locale sau cloud și sisteme de operare. [39]

SQL Server 2019 permite utilizatorilor să se alăture containerelor SQL Server, HDFS și Spark folosind o nouă caracteristică Big Data Cluster. SQL Server 2019 introduce, de asemenea, compilări de indexuri de coloane, reconstrucții și mascare de date statice. Recuperarea accelerată a datelor este, de asemenea, nouă, care efectuează și anulează o fază de refacere în cel mai vechi număr de secvență a jurnalului de pagină. [39]

## 5.4. Microsoft Azure

Microsoft este unul dintre cei mai mari jucători pe piața de cloud. Puțin în urmă față de Amazon, care au fost pionieri în acest domeniu însă destul de competitivi și probabil cei mai inovatori. Nu aș vrea să sună ca și cum sunt cel mai mare fan Microsoft, însă având în vedere că am ales să folosesc platforma .Net pentru dezvoltarea aplicației, soluțiile de cloud de la Azure mi s-au părut cele mai potrivite, deși Azure nu ne limitează la a folosi doar .Net, sistemul lor este complet agnostic, putând rula cu ușurință și sisteme LINUX sau JAVA. O alternativă foarte bună este platforma Amazon AWS, aceștia fiind pionerii platformelor cloud.

### 5.4.1. Concepte generale

Există multe definiții ambiguë ale acestui termen folosit de multe ori greșit. În termeni generali, Cloud-ul se referă la o rețea globală de servere menite să îndeplinească anumite funcționalități [18]. Dar cloud-ul nu este o entitate fizică (deși în spatele lui sunt tot componente fizice) ci se referă mai degrabă la rețeaua virtuală de servere ce îl alcătuiesc pentru a crea un singur ecosistem.

Atunci când vrem să publicăm o aplicație undeva, în special o aplicație web, avem două categorii mari de platforme:

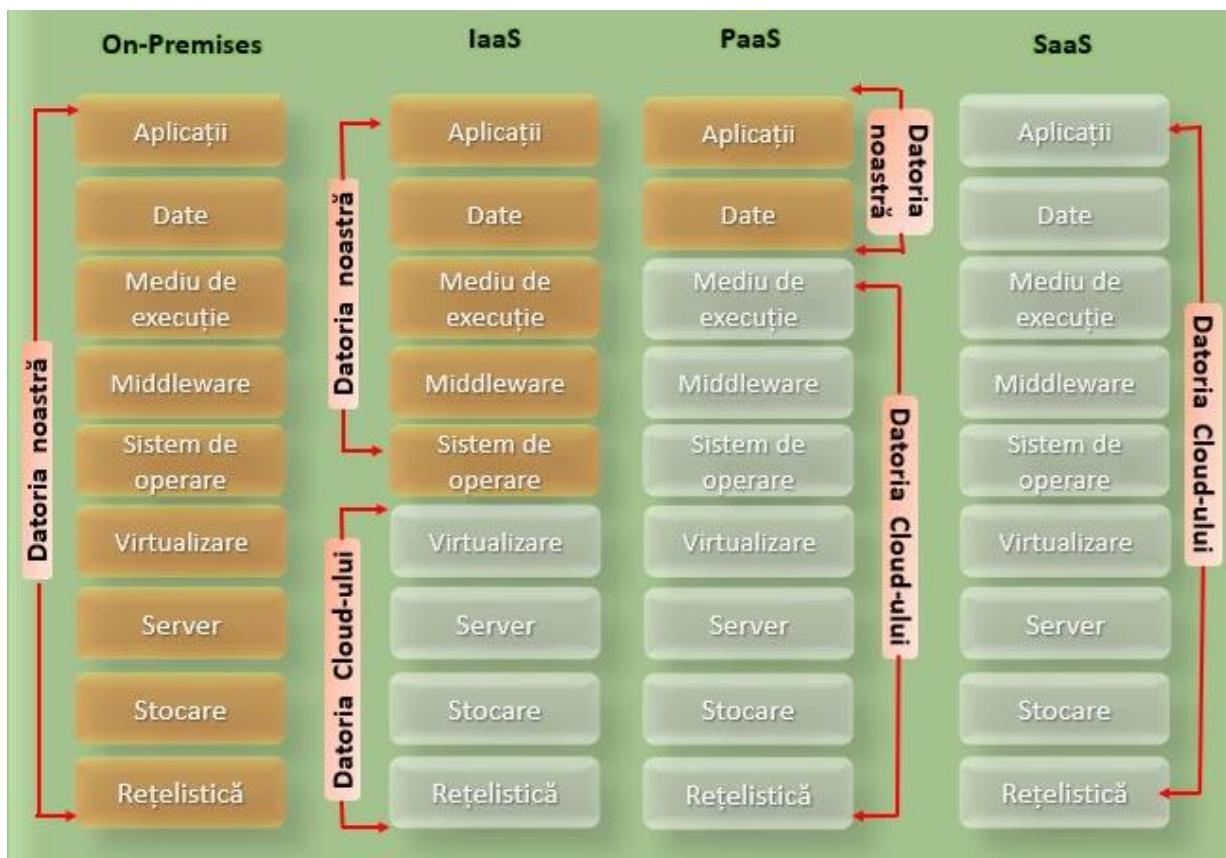
- On Premises – modelul clasic unde avem un server fizic gestionat de noi (chiar dacă acest server este calculatorul nostru personal)
- Cloud – tendința modernă despre care vom vorbi în continuare.

Există două mari categorii de cloud: cloud privat (folosit intern de o anumită organizație) și cloud public (ce poate fi folosit de către toată lumea) așa cum sunt Amazon, Azure și Google Cloud. Dar mai există și conceptul de cloud hibrid unde se combină aceste două tipuri.

## 5.4.2 IaaS, PaaS și SaaS

În funcție de împărțirea responsabilităților între utilizatorul platformei cloud și furnizorul de servicii cloud, majoritatea serviciilor cloud se încadrează în una dintre aceste 3 categorii [18] :

- IaaS = Infrastructure as a Service
- PaaS = Platform as a Service
- SaaS = Software as a Service



Figură 22 - Comparație servicii Cloud

### 5.4.1.2. De ce Azure?

Cloud-ul Azure oferă peste 200 de servicii și produse diferite, de la servicii de computație, până la servicii de stocare sau inteligență artificială. Dintre toți furnizorii de cloud, Microsoft implementează cel mai bine conceptul de Cloud Hibrid. Azure oferă cel mai mare număr de regiuni și zone de disponibilitate(peste 60) dintre toți furnizorii de cloud [19].

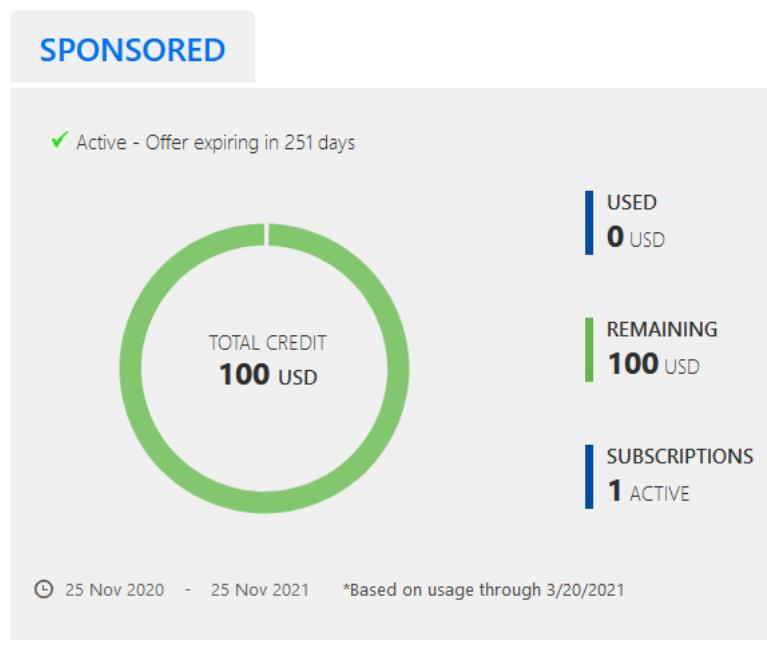
De asemenea, Azure are cea mai bună capacitate de a scala automat, putând ajusta performanța în funcție de cerințele aplicației, pentru a obține un cost cât mai mic.



#### 5.4.1.2. Azure pentru studenți.

Microsoft în general este o companie care investește foarte mult în educație. Unul dintre beneficiile oferite de Microsoft este accesul gratuit la platforma Azure pentru studenți. Folosind email-ul instituțional de student, ne putem înregistra în acest program folosind adresa <https://azure.microsoft.com/en-us/free/students/>

Abonamentul are o balanță de 100 de dolari care pot fi consumați în decurs de un an de zile.



Figură 23 – Sponsorizare Azure pentru studenți

O alternativă ar fi să ne creăm un cont de "trial" care vine cu suma de 200 de dolari însă este valabil doar o lună de zile, un timp prea scurt pentru realizarea unei aplicații mai complexe.

Pe lângă această sumă, există multe servicii care sunt complet gratuite (Azure Active Directory) sau care au o limită lunară ce nu se taxează (free grant) cum ar fi Funcțiile Serverless; care se pot folosi practice gratuit pentru aplicații care nu au un număr mare de utilizatori,

#### 5.4.2. Servicii oferite de Azure

Pe site-ul [azurecharts.com](http://azurecharts.com), putem vedea toate cele peste 200 de servicii oferite de Azure dar și zonele în care acestea sunt disponibile. Este datoria noastră să verificăm dacă serviciul pe care vrem să-l folosim este disponibil în zona de disponibilitate aleasă.

Zona de disponibilitate se alege în funcție de locația de unde este accesată aplicația cel mai des, pentru a reduce costurile. Pentru această aplicație am ales zona Europei de Vest.



Serviciile Azure sunt grupate în unități logice numite "Resource Groups". O resursă nu poate exista în afara unui astfel de grup, putem avea însă servicii din zone diferite și chiar din subșcripții diferite. În aplicația noastră avem un singur Resource Group, în cazul în care am fi avut mai mulți clienti sau mai multe medii de dezvoltare, am fi avut câte un grup pentru fiecare.

Aplicația LearningMap folosește următoarele servicii Azure:

Showing 1 to 5 of 5 records. <input type="checkbox"/> Show hidden types ⓘ		No grouping ▾	
Name ↑↓	Type ↑↓	Location ↑↓	
Server baze de date	learning-map	SQL server	West Europe
Aplicație web	learningmap-api-dev1	App Service	West Europe
Diagnostic și Loguri	learningmap-api-dev1	Application Insights	West Europe
Bază de date SQL Server	learningmap-dev1 (learning-map/learningmap-dev1)	SQL database	West Europe
Container aplicații web	LearningMap-FarmDev1	App Service plan	West Europe

Figură 24 - Resurse Azure folosite

#### 5.4.2.1. Servicii de Identitate și Securitate

Identitatea și securitatea în Cloud este un subiect foarte important, mai ales în contextul noii legi pentru protecția datelor GDPR.

Azure dispune de peste 10 astfel de servicii, oferind protecție inclusiv împotriva atacurilor DDOS (Denial-of-service).

##### 5.4.2.1.1 Azure Active Directory

Microsoft oferă acest serviciu gratuit, nu vom consuma credite din subșcripție pentru a folosi Azure AD. De fapt Microsoft Identity Platform se bazează pe Azure AD. Acest serviciu se poate folosi pentru a ne păstra baza de date a utilizatorilor. Însă Azure AD este mai mult decât o bază de date a utilizatorilor, este un serviciu complet de management al identității. Platforma Microsoft Identity pe care am folosit-o pentru păstrarea utilizatorilor în propria bază de date este de fapt derivată din Azure Active Directory.

##### 5.4.2.1.2 Azure Key Vault

Așa cum am văzut atunci când am configurat baza de date, avem nevoie să definim un string ce conține informații de autentificare. Similar, am avut nevoie de anumiți identificatori pentru a ne integra cu Azure Active Directory. Se pune problema evidentă a securității, unde definim astfel de informații și cum le păstrăm în siguranță. Key Vault poate fi folosit pentru a păstra parole, stringuri de conexiune, certificate de securitate și multe altele.



#### Connection strings

Connection strings are encrypted at rest and transmitted over an encrypted channel.

		<a href="#">New connection string</a>	<a href="#">Hide values</a>	<a href="#">Advanced edit</a>
		<input type="text" value="Filter connection strings"/>		
Name	Value			Source
DefaultConnection	<a href="#"> Server=tcp:learning-map.database.windows.net,1433;Initial Catalog=learningmap-dev1;Persist Security Info=False;User ID=galileo;Password=Invata10;</a>			App Service Config

Figură 25 - Stringul de conexiune SQL Server

#### 5.4.2.2 Servicii de computație

Acestea sunt de fapt cele mai importante componente ale platformelor Cloud. Azure oferă servicii de computație din toate categoriile : IaaS (mașini virtuale), PaaS(App Service) dar și un concept nou numit FaaS (function as a service) în care se implementează conceptul de serverless. De asemenea, Azure suportă și managementul containerelor Docker.

##### 5.4.2.2.1 Azure App Services

În cazul în care dezvoltam aplicații web, vom avea nevoie cu siguranță de un server web. Una dintre cel mai populare soluții oferite de Azure este "App Service". Aceste servicii se pot scala foarte simplu atât orizontal(mărind memoria și puterea procesorului) cât și vertical (mărind numărul de instanțe).

#### Consola KUDU

KUDU este motorul din spatele unui AppService în Azure ce se ocupă de publicarea aplicației noastră. Prin KUDU putem vedea procesele ce rulează pe acel AppService dar și structura de fișiere de pe disc.

Folosind această utilitară, putem vizualiza executabilul care initializează aplicația web.

The screenshot shows the Kudu interface for a web application named 'LearningMap'. The top navigation bar includes links for 'Kudu', 'Environment', 'Debug console', 'Process explorer', 'Tools', and 'Site extensions'. On the right, there is a user profile icon and the email address 'florin.asavei@student.unitbv.ro'. Below the navigation, the path '... / wwwroot' is shown with a '+' button and '87 items'. There are icons for upload, download, and refresh. A table lists the files in the wwwroot directory:

	LearningMap.Infrastructure.pdb	5/16/2021, 3:44:39 PM	37 KB
	LearningMap.WebAPI.deps.json	5/16/2021, 7:10:28 PM	321 KB
	LearningMap.WebAPI.dll	5/16/2021, 7:10:26 PM	59 KB
	<b>LearningMap.WebAPI.exe</b>	5/16/2021, 7:10:26 PM	98 KB
	LearningMap.WebAPI.pdb	5/16/2021, 7:10:26 PM	34 KB
	LearningMap.WebAPI.runtimeconfig.json	5/16/2021, 6:54:03 PM	1 KB
	LearningMap.WebAPI.Views.dll	5/16/2021, 7:10:27 PM	21 KB
	LearningMap.WebAPI.Views.pdb	5/16/2021, 7:10:27 PM	23 KB

Figură 26 - Platforma Kudu

Tot aici, putem vedea fișierele sursă ale aplicației Angular care se ocupă de interfața web.

... / wwwroot + | 88 items |

Name	Modified	Size
frontend-angular	5/16/2021, 7:12:30 PM	
obj	5/16/2021, 7:11:24 PM	
wwwroot	5/16/2021, 6:54:44 PM	
appsettings.Development.json	4/23/2021, 10:54:41 PM	1 KB
appsettings.json	4/23/2021, 10:54:41 PM	1 KB
appsettings.Production.json	5/16/2021, 7:11:04 PM	1 KB
AutoMapper.dll	10/16/2020, 3:36:54 PM	280 KB
AutoMapper.Extensions.ExpressionMapping.dll	2/10/2021, 5:50:54 PM	98 KB
AutoMapper.Extensions.Microsoft.DependencyInjection.dll	2/5/2021, 2:28:34 PM	13 KB

```
PS C:\home\site\wwwroot> New-Item -Path . -Name "testfile1.txt" -ItemType "file" -Value "This is a text string."
New-Item -Path . -Name "testfile1.txt" -ItemType "file" -Value "This is a text string."
```

Figură 27 - Structura fișierelor pe server

Serverul web este configurat să rezolve această cale (relativă la executabilul Learning.Map.WebAPI.exe) folosind conceptul de Middleware din .Net Core. Mai exact, următoarea bucată de cod se ocupă de această configurație (leagă interfață grafică a lui Angular de API-ul de backend).

```
app.UseSpa(spa =>
{
    spa.Options.SourcePath = "frontend-angular/ClientApp/";
});
```

### 5.2.2.3 Servicii de date

Azure oferă atât servicii de stocare(fisiere) cât și servicii de baze de date. Există numeroase tipuri de baze de date suportate, atât relaționale cât și non-relaționale, pentru toate cele 3 categorii de servicii. Pentru a restrânge această categorisire, ne vom referi doar la posibilitățile oferite pentru bazele de date SQL Server:

- Soluția IaaS: SQL server într-o mașină virtuală
- Soluția PaaS: Azure SQL

#### 5.2.2.3.1. Azure SQL Server

În spate, Azure SQL folosește de fapt ultima versiunea de Microsoft SQL Server (momentan SQL 2019).

În funcție de nevoile noastre și de cât de mult vrem să deținem controlul, putem alege să ne gestionăm singuri serverul de vaze de date sau să lăsăm această sarcină pe seama furnizorului de servicii Cloud.



În Azure există 3 categorii de servicii SQL [33] :

- SQL Server Database – presupune existența unui singur server de baze de date, având o singură bază de date.
- Elastic Pool – grupează mai multe baze de date care folosesc același server.
- Instanță de SQL server – în cazul în care vrem să avem mai mult control de administrare a serverului, ideală pentru scenariul în care avem deja o aplicație configurață să folosească o bază de date on-premises.

În aplicația noastră am ales folosirea unei singure baze de date pentru a avea un cost cât mai mic și mai predictibil. Opțiunea Elastic Pool având un cost prea mare pentru aplicații mici.

### 5.2.2.3. Servicii de productivitate în dezvoltare

După cum am văzut în subcapitolul 1.3 (Designul aplicației), productivitatea este un element esențial în dezvoltarea unui produs software. Ne dorim ca din momentul în care scriem codul până când acesta ajunge să se compileze și să rezulte într-o funcționalitate reală, să existe cât mai puțină intervenție manuală.

#### 5.2.2.3.1. Azure DevOPS

Azure DevOPS este o unealtă de productivitate care încorporează mai multe componente:

- Azure Boards – pentru managementul proiectului
- Azure Pipelines – pentru operațiuni de DevOps
- Azure Repos – pentru versionarea codului (GIT).

Toate aceste servicii se integrează foarte ușor. Atunci când vrem să publicăm aplicația în cloud, folosind Pipelines, putem specifica ce repository de GIT să se folosească și în ce Resource Group să se facă publicarea aplicației web.



## 6. Instrucțiuni de utilizare ale aplicației Learning Map

Aplicația LearningMap dispune de 3 tipuri de utilizatori, fiecare având propriul set de meniuri vizibile:

- Administrator: Manage Users, API (Swagger)
- Teacher: Manage Learning Paths, Map
- Student: Learn, Map progress

Toți utilizatorii vor vedea meniurile Home și Login/Logout.

### 6.1 Managementul utilizatorilor

Din oficiu, în aplicație va exista un utilizator cu rolul de "Administrator" (proces descris în subcapitolul 2.2.3 - Popularea bazei cu date inițiale prin metoda de Seed), având următoarele credențiale:

Username: **administrator@localhost** | Parola: **Administrator1!**

Un utilizator cu rolul de "Administrator" va avea acces la meniul "Manage Users" unde va vedea toți utilizatorii aplicației împreună cu rolurile acestora.

De aici, Administratorul poate adăuga, șterge sau modifica utilizatori.

Utilizatorii vor fi grupați într-un meniu de tip acordeon, în funcție de rol. Cei ce au mai multe roluri, vor apărea în fiecare grup de roluri.

Teachers :			
Username	Role(s)		
developer@localhost	Teacher	Administrator	Student
teacher1@localhost	Teacher		

Figură 28 - Management utilizatorilor

Un utilizator din exterior, atunci când se va înregistra, va primi rolul de "Student". Utilizatorii pot vedea rolul propriu, făcând hover peste numele lor din meniul din dreapta-sus. Dacă vor da click pe numele de utilizator vor ajunge la pagina de management al identității de unde își pot schimba informațiile contului (inclusiv parola) sau își pot șterge contul (respectând cerințele GDPR).



## 6.2. Managementul resurselor (trasee, capitole, resurse)

Utilizatorii cu rolul "Teacher" pot vedea meniul "Manage Learning Paths" de unde pot adăuga trasee educaționale, ce conțin capitole, care la rândul lor conțin resurse educaționale.

Adăugarea unei entități se face folosind butonul +

Pentru modificarea unei entități, avem meniul de detalii care se deschide din iconița cu 3 puncte (...). Tot din această componentă de detalii putem șterge entitatea. Ștergerea unei entități părinte, va șterge și copiii (ștergerea unui capitol va șterge și resursele educaționale).

Pentru capitole și resurse putem defini o ordine și o dificultate.

La nivelul cel mai de jos, al resurselor, putem defini tipul resursei (Carte, Articol, Video sau Curs Interactiv) și putem specifica un timp inițial estimativ. O resursă va fi de fapt un URL către un articol/carte/video.

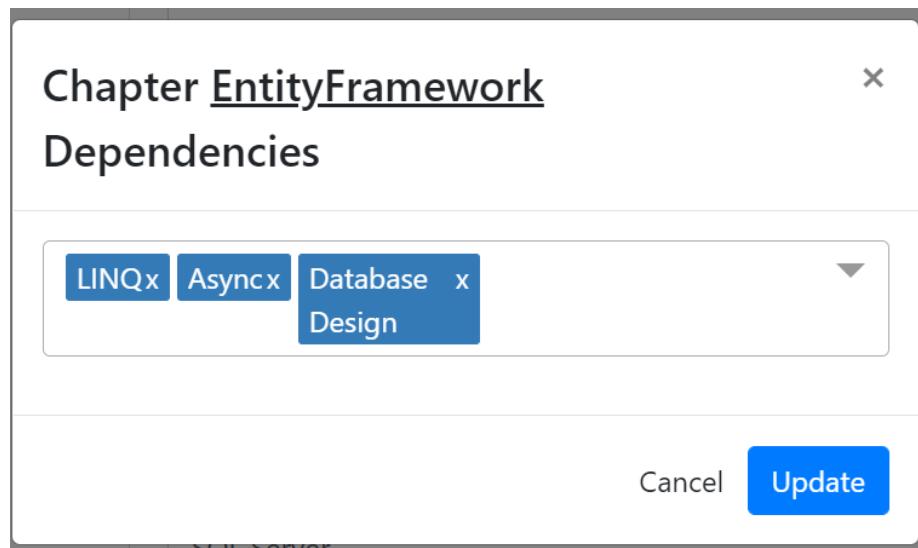
The screenshot shows a form for creating a learning resource. The fields are:

- Name: C# Statements
- Order: 1
- Required Level: Mandatory
- Resource Type: Article
- Estimated Duration (hours & minutes): 03 : 30 (with up and down arrows for adjustment)
- URL: <https://docs.microsoft.com/en-us/dotnet/csharp/programming>

Figură 29 - Exemplu resursă educațională



La nivelul capitolelor se pot defini legături între acestea (dependențe). Acest meniu de configurare se poate deschide folosind iconita



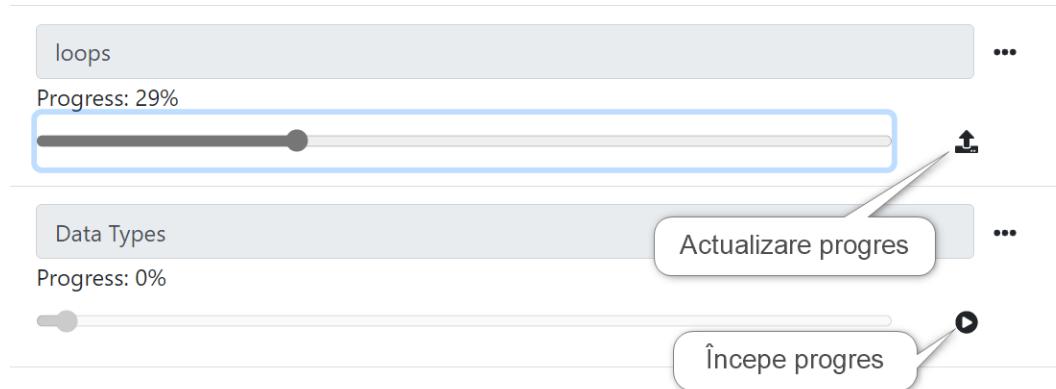
Figură 30 - Exemplu dependențe capitulo

### 6.3. Logarea progresului de către student

Din meniul Learn, studentul poate vedea toate traseele, capitoalele și resursele educaționale, în structura definită de profesor, însă va avea acces doar de citire.

Pentru a începe lucrul pe o anumită resursă, va apăsa butonul play.

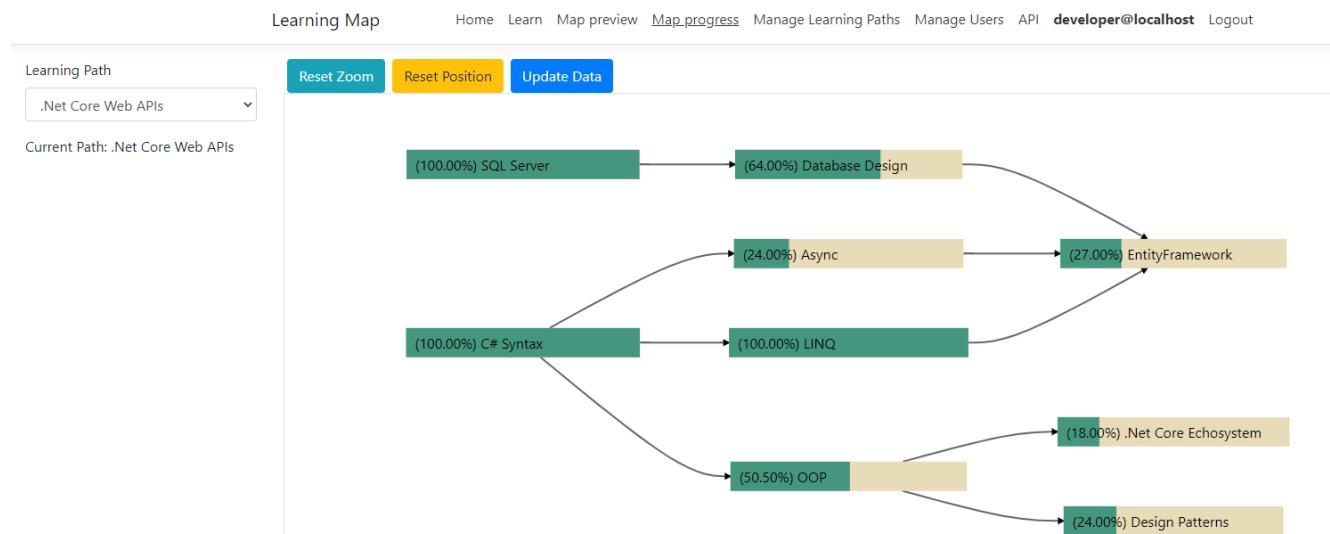
Apoi, folosind un control de tip slider va selecta progresul. La final, pentru actualizarea progresului, va apăsa butonul de upload.



Figură 31 - Control actualizarea progresului

### 6.4. Vizualizarea datelor

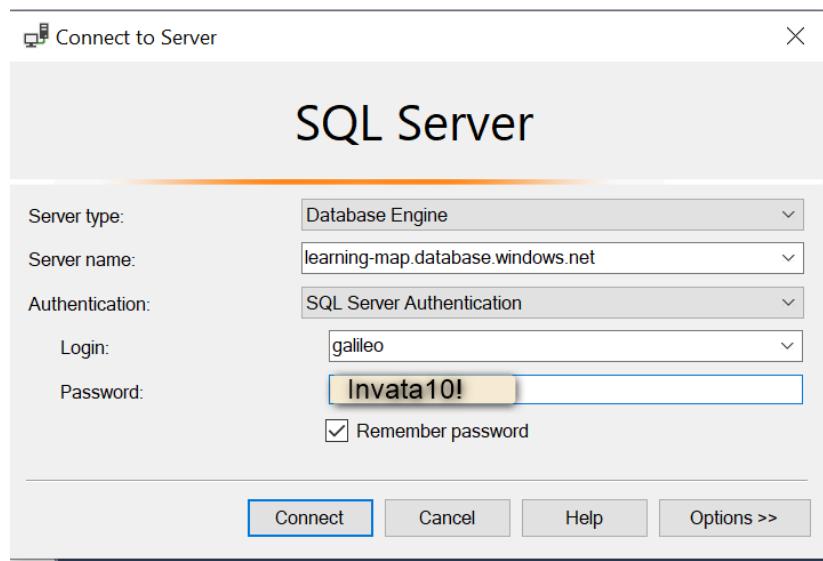
Partea de vizualizare a datelor este disponibilă atât pentru profesor cât și pentru student. În cazul studentului, vom vedea progresul pentru fiecare capitol în parte.



Figură 32 - Vizualizarea datelor

## 6.5. Instrucțiuni de depanare

Pentru a ne conecta la baza de date din Cloud-ul Azure putem folosi SQL Server Management Studio (SSMS) unde vom introduce numele serverului de baze de date, utilizatorul administrator și parola asociată



Figură 33 - Accesare SQL Azure

Desigur, putem folosi și o bază de date locală de tip SQL Server 2019 pentru a face economie, tot ce trebuie să facem este să modificăm string-ul de conectare.



## 7. Concluzii și perspective de viitor

După cum am văzut, există platforme și tehnologii moderne ce se pot folosi pentru implementarea unei aplicații web, urmând o arhitectură curată, performantă și sigură.

### 7.1. Concluzii

Provocarea principală a fost găsirea unei librării JavaScript care să se integreze bine cu framework-ul ales. Integrarea librăriei ngx-graph reprezintă contribuția personală cea mai importantă în acest proiect.

Am văzut de asemenea, utilitatea structurilor de date de tip graf în reprezentările vizuale. Mulțumită beneficiilor pentru studenți oferite de Azure, am putut publica aplicația într-un ecosistem modern de tip cloud, ținând cont de bunele practici arhitecturale. Un alt aspect important a fost implementarea bunelor practici de management al proiectului, versionare și DevOPS, permitând o dezvoltare rapidă și sigură.

### 7.2 Posibilități de extindere

Dezvoltarea unei aplicații, nu se termină niciodată, mereu putem face îmbunătățiri sau refactorizări.

Putem lua în considerare următoarele viitoare îmbunătățiri:

- Permiterea actualizării parțiale a datelor folosind PATCH.
- Implementarea standardului Microsoft OData pentru a îmbunătăți constrângerile REST.
- Separarea aplicației de Angular într-un Azure App Service separat pentru a permite scalarea independentă a celor două aplicații (API .Net și UI Angular).
- Introducerea unui modul de rapoarte.
- Permiterea creării de conturi cu rolul de profesor și din exterior(nu doar de către utilizatorii cu rol de administrator).
- Actualizarea versiunilor de .Net (la versiunea 6 în Noiembrie 2021) și Angular (la versiunea 11).
- Sporirea numărului de teste unitare pentru creșterea calității și stabilității.
- Crearea mai multor ecosisteme (dezvoltare, testare, producție) pentru îmbunătățirea ciclului de viață al produsului.



## Listă de figuri

Figură 1 – Exemplu platformă e-learning.....	7
Figură 2 – Exemplu traseu educațional.....	8
Figură 3 – User Story în Azure DevOps .....	14
Figură 4 - Creare branch.....	16
Figură 5 - GitHub Flow .....	17
Figură 6 - Pipeline Azure DevOps.....	19
Figură 7 - schema bazei de date.....	22
Figură 8 - Diagrama de desfășurare.....	23
Figură 9 - Diagrama cazurilor de utilizare.....	24
Figură 10 - Decodare token JWT.....	28
Figură 11 - Header de autorizare .....	30
Figură 12 - Analiză trafic cu Fiddler.....	30
Figură 13 - Exemplu digraf.....	33
Figură 14 - Tipuri de licențe software.....	38
Figură 15 - O pagină GitHub oficială.....	40
Figură 16 - Reputația unui repository.....	40
Figură 17 - Arhitectura API-ului .....	50
Figură 18 - Certificat SSL în browser.....	52
Figură 19 - Certificat SSL auto-generat .....	53
Figură 20 - Flow autentificare și autorizare.....	55
Figură 21 - Planificarea versiunilor .Net.....	57
Figură 22 - Comparație servicii Cloud .....	62
Figură 23 - Sponsorizare Azure pentru studenți.....	63
Figură 24 - Resurse Azure folosite .....	64
Figură 25 - Stringul de conexiune SQL Server .....	65
Figură 26 - Platforma Kudu .....	65
Figură 27 - Structura fișierelor pe server.....	66
Figură 28 - Management utilizatorilor.....	68
Figură 29 - Exemplu resursă educațională.....	69
Figură 30 - Exemplu dependențe capitole.....	70
Figură 31 - Control actualizarea progresului.....	70
Figură 32 - Vizualizarea datelor.....	71
Figură 33 - Accesare SQL Azure .....	71



## **Listă de tabele**

Tabel 1 - Exemplu matrice adiacență.....	34
Tabel 2 - Exemplu listă adiacență succesori.....	35
Tabel 3 - Exemplu listă de adiacență predecesori.....	35
Tabel 4 - Listă de adiacență normalizată.....	36
Tabel 5 - Comparație librării JS pentru vizualizarea grafurilor.....	37
Tabel 6 - Paritatea operațiunilor CRUD cu verbele HTTP.....	47
Tabel 7 - Versiuni Angular [15].....	59



## Bibliografie și Webografie

- [1] Dorin Bocu și Răzvan Bocu, **Modelare Obiect Orientată cu UML**, Editura Albastră, Cluj-Napoca, 2006
- [2] Bocu Dorin, **Inițiere în ingineria sistemelor soft**, Editura Albastră, Cluj-Napoca, 2002
- [3] Lucian Sasu, **Medii Vizuale de programare**, Curs pentru învățământ la distanță, Brașov, 2014
- [4] Valeriu Iorga, **Colecții de date**, Editura Albastră, Cluj-Napoca, 2009
- [5] Scott Chacon și Ben Straub, **Pro Git**, Editura Apress, ediția 2, 2021 (disponibilă gratuit la <https://git-scm.com/book/en/v2>)
- [6] Robert C. Martin, **Clean Architecture: A Craftsman's Guide to Software Structure and Design**, Editura Pearson, USA, 2017
- [7] Robert C. Martin, **Clean Code: A Handbook of Agile Software Craftsmanship**, editura Pearson, 2008
- [8] Eric Ries, **The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses**, Editura Currency, USA, 2011
- [9] Roy Thomas Fielding, **Architectural Styles and the Design of Network-based Software Architectures**, Universitatea Irvine California, 2000
- [10] Valerio De Sanctis, **ASP.NET Core 5 and Angular: Full-stack web development with .NET 5 and Angular 11**, ediția 4, Editura Packt, SUA, 2021
- [11] Mark J. Price, **C# 9 and .NET 5 – Modern Cross-Platform Development**, ediția 5, Editura Packt, SUA, 2021
- [12] Manifestul pentru dezvoltarea agilă, <https://agilemanifesto.org/iso/ro/manifesto.html>
- [13] 12 Principles to Material Design, <https://xd.adobe.com/ideas/principles/app-design/new-to-material-design-12-principles-you-need-to-know/>
- [14] Documentație Azure Pipelines, <https://docs.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops>
- [15] Documentație Angular, <https://angular.io/docs>
- [16] MDN, The HTTP Protocol, <https://developer.mozilla.org/en-US/docs/Web/HTTP>
- [17] Agile Methodology: What is Agile Software Development Model?, <https://www.guru99.com/agile-scrum-extreme-testing.html>
- [18] What is the cloud? , <https://azure.microsoft.com/en-us/overview/what-is-the-cloud/>
- [19] Azure vs. AWS, <https://azure.microsoft.com/en-us/overview/azure-vs-aws/>
- [20] Entity Data Model, <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/entity-data-model>



- [21] Code-First vs Model-First vs Database-First: Pros and Cons, <https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/>
- [22] What is the Microsoft identity platform?, <https://docs.microsoft.com/en-us/azure/active-directory/develop/v2-overview>
- [23] Role-based authorization in ASP.NET Core, <https://docs.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-5.0>
- [24] Standardul RFC 7519 (JWT), <https://datatracker.ietf.org/doc/html/rfc7519>
- [25] 5 Types of Software Licenses You Need to Know About, <https://snyk.io/learn/what-is-a-software-license/>
- [26] Architectural principles, <https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/architectural-principles>
- [27] Three-Tier Architecture, <https://www.ibm.com/cloud/learn/three-tier-architecture>
- [28] CQRS pattern, <https://docs.microsoft.com/en-us/azure/architecture/patterns/cqrs>
- [29] N-tier architecture style, <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier>
- [30] StackOverflow Survey 2020, <https://insights.stackoverflow.com/survey/2020>
- [31] Standardul RFC 6749 (OAuth2), <https://datatracker.ietf.org/doc/html/rfc6749#section-4.1>
- [32] SQL Server Tutorial, <https://www.sqlservertutorial.net>
- [33] Azure SQL Database, <https://docs.microsoft.com/en-us/learn/modules/explore-relational-data-offerings/4-azure-sql-database>
- [34] Clean Architecture with .NET Core: Getting Started, <https://jason-taylor.dev/clean-architecture-getting-started/>
- [35] The History of JavaScript, <https://www.springboard.com/blog/data-science/history-of-javascript/>
- [36] OpenID Connect Protocol, <https://auth0.com/docs/protocols/openid-connect-protocol>
- [37] Add a TLS/SSL certificate in Azure App Service, <https://docs.microsoft.com/en-us/azure/app-service/configure-ssl-certificate>
- [38] Compare EF Core & EF6, <https://docs.microsoft.com/en-us/ef/efcore-and-ef6/>
- [39] What's new in SQL Server 2019, <https://docs.microsoft.com/en-us/sql/sql-server/what-s-new-in-sql-server-ver15?view=sql-server-ver15>
- [40] Relational vs. Non-Relational Database: Pros & Cons, <https://aloa.co/blog/relational-vs-non-relational-database-pros-cons>
- [41] Wikipedia despre Anders Hejlsberg, [https://en.wikipedia.org/wiki/Anders\\_Hejlsberg](https://en.wikipedia.org/wiki/Anders_Hejlsberg)