

MINISTERUL EDUCAȚIEI



UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

APLICAȚIE WEB DE GESTIUNE A FIȘELOR DE DISCIPLINĂ

LUCRARE DE LICENȚĂ

Absolvent: **Eugeniu COJOCARU**

Coordonator **S.l.dr.ing. Călin CENAN**
științific:

2022

Cuprins

Capitolul 1	Introducere - Contextul proiectului	1
1.1	Contextul temei	1
1.2	Structura proiectului	2
Capitolul 2	Obiectivele Proiectului	3
2.1	Prezentarea temei in detaliu	3
2.2	Rezumat obiective	4
Capitolul 3	Studiu Bibliografic	5
3.1	Back-end	5
3.1.1	Limbaaj de programare	5
3.1.2	Framework	6
3.1.3	API	6
3.2	Front-end	7
3.2.1	Limbaaj de programare	7
3.2.2	Framework	8
3.3	Baza de date	11
3.4	Arhitectura	11
3.5	Securitate	13
3.5.1	Protocol de comunicare	13
3.5.2	Algoritm de criptare	14
3.5.3	Autorizare pe baza JWT Token	14
Capitolul 4	Analiză și Fundamentare Teoretică	15
4.1	Arhitectura	15
4.2	Back-end	15
4.2.1	Arhitectura	15
4.2.2	Baza de date	16
4.3	Front-end	18
4.3.1	Arhitectura	18
4.4	Securitate	19
4.5	Fișa disciplinei	21
Capitolul 5	Proiectare de Detaliu și Implementare	22
5.1	Arhitectura detaliată a aplicației	22
5.1.1	Baza de date	22
5.1.2	Back-end	27
5.1.3	Front-end	37
5.2	Funcționalități	43
5.2.1	Adăugarea și gestiunea ierarhiei de instituții, facultăți, departamente și domenii de studiu din cadrul universității	43
5.2.2	Adăugarea și gestiunea cadrelor didactice	43
5.2.3	Adaugrea și gestionarea materiilor	44
5.2.4	Adăugarea și gestionarea fișelor de disciplină	44
5.2.5	Vizualizare versiuni anterioare ale fișelor de disciplină	49

5.2.6	Export fișe de disciplină în format PDF	49
5.2.7	Audit de securitate pentru toate operațiunile de mai sus	49
Capitolul 6 Testare și Validare		51
6.1	Back-end	51
6.2	Font-end	52
Capitolul 7 Manual de Instalare și Utilizare		53
7.1	Instalare	53
7.1.1	Resurse hardware	53
7.1.2	Resurse software	53
7.1.3	Back-end	53
7.1.4	Front-end	54
7.2	Manual de utilizare	54
Capitolul 8 Concluzii		62
8.1	Rezultate obținute	62
8.2	Descriere a posibilelor dezvoltări și îmbunătățiri ulterioare	62
Bibliografie		64
Anexa A Exemplu fisa de disciplina		65

Capitolul 1. Introducere - Contextul proiectului

1.1. Contextul temei

Motivația principală a lucrării este adusă de necesitatea umană de a simplifica procesul numit munca, prin dezvoltarea de unelte. O unealtă reprezintă o un lucru fizic sau o metodă de a ușura o sarcină, prin reducerea dificultății acesteia sau prin reducerea timpului de execuție, preferabil prin reducerea amândurora. Încă din primele zile ale omenirii s-au dezvoltat unelte, primitive, dar care ajutau oamenii să îndeplinească unele obiective care erau necesare supraviețuirii: arcuri și cuțite la vântoare, cuțite și containere pentru adunatul diverselor fructe și îmbrăcăminte pentru a proteja omul de frig sau vreme rea. Omul și-a dat seama că folosind unelte șansa lui de supraviețuire crește dramatic. Acest lucru l-a împins să dezvolte noi și noi metode de a ușura evenimentele de zi cu zi. Această evoluție continuă, această dorință de a petrece mai puțin timp/a lucra mai eficient pentru lucrurile necesare, cu scopul de a investi apoi timpul rămas în lucrurile care generează plăcere, a împins omenirea din spate până când de la uneltele primitive făurite din piatră și bete s-au creat rețele de comunicații, calculatoare performante și noi metode de a ușura munca.

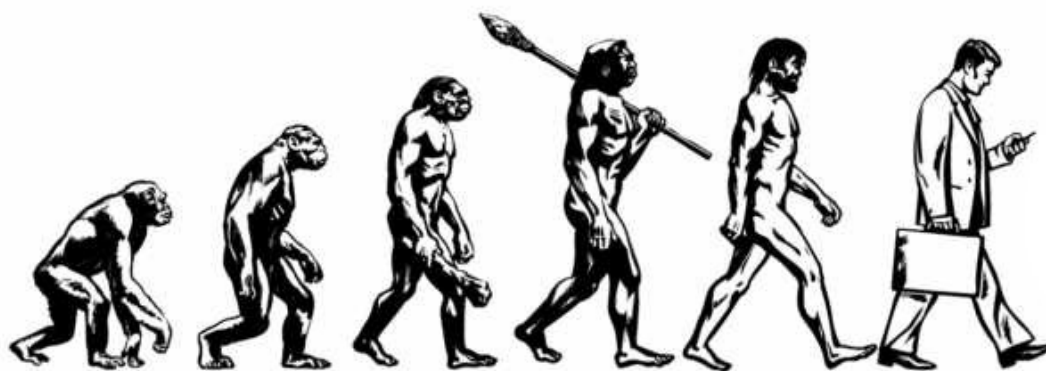


Figura 1.1: Evoluția omenirii și a uneltelor până în era modernă ¹

În această eră digitalizată, calculatoarele, microprocesoarele sunt unelte extrem de utile pentru rasa umană. Astfel majoritatea job-urilor care foloseau pix și hârtie au ajuns să folosească mouse și tastatură. Aplicațiile de management au ca scop reducerea timpului necesar finalizării unei sarcini, centralizarea datelor, reducerea riscului erorilor cauzate de oameni(de exemplu vărsarea unei cafele pe teancul de documente care trebuie să fie

¹<https://www.jobshopleanmanufacturing.com/cavemen-dealing-with-difficult-people.html>

predate în acea zi), reducerea accesului personalului neautorizat, împărțirea drepturilor de acces a angajaților la informațiile companiei și multe altele.

1.2. Structura proiectului

Proiectul este structurat în 8 capitole, fiecare explicând o parte a procesului urmat de mine pentru dezvoltarea aplicației de gestiune a fișelor de disciplină. Cele 8 capitole sunt:

- Introducere - contextul proiectului: în acest capitol se explică într-un mod generalizat tematica proiectului și beneficiile care reies în urmă evoluției temei alese
- Obiectivele proiectului: capitolul acesta explică viziunea mea asupra temei și oferă o listă de obiective pe care doresc să le îndeplinesc la finalizarea lucrării
- Studiu bibliografic: acest capitol este dedicat expunerii resurselor: cărți, articole, site-uri web și documentații utilizate pentru dezvoltarea aplicației
- Analiză și fundamentare teoretică: capitol în care creionez o imagine de ansamblu a aplicației, dar cu mai multe referințe pentru partea teoretică
- Proiectare de detaliu și implementarea: acest capitol este dedicat pentru oferirea detaliilor de implementare, atât pentru partea de arhitectură, cât și pentru partea de dezvoltare propriu-zisă a bazei de date, WebAPI-ului și aplicației React
- Testare și validare: în acest capitol vorbesc despre metodele prin care am testat funcționalitatea aplicației și uneltele folosite pentru testare
- Manual de instalare și utilizare: capitol rezervat pentru instrucțiuni de dezvoltare ulterioară și de utilizare a aplicației în condiții optime
- Concluzii: capitol în care îmi exprim părerea despre proiectul final și expun metode de îmbunătățire ulterioară

Capitolul 2. Obiectivele Proiectului

2.1. Prezentarea temei în detaliu

Lucrarea mea de licență se încadrează în categoria aplicațiilor de management. Într-o instituție de dimensiuni mari cum este și Universitatea Tehnică din Cluj-Napoca există multe informații care trebuie procesate. Acest proces de procesare a datelor nu mai poate să fie realizat manual, din cauza imensității de informații care trebuie citite, interpretate și apoi modificate. Toată această gestiune utilizează mult timp și forță de muncă, ceea ce nu mai este rentabil în zilele secolului 21 când totul este digitalizat.

Aplicația propusă de mine are scopul de a minimiza timpul necesar redactării, modificării sau vizualizării versiunilor trecute ale unei fișe de disciplină. Acest lucru presupune centralizarea informațiilor și proceselor (operații care, curent, sunt realizate manual) într-o aplicație digitalizată, care poate fi accesată oricând și oriunde există o conexiune la internet. Aplicația oferă utilizatorilor (profesori sau ajutoare) o interfață grafică ușor de înțeles, ușor de învățat și ușor de utilizat. Aplicația nu este perfectă, însă cu ajutorul personalului care ar utiliza aplicația, se pot dezvolta mult mai multe funcționalități și metode de a utiliza mai ușor aplicația. Asemenea oricărei aplicații care iese prima dată pe piață, aplicația mea pentru gestiunea fișelor de disciplină nu este perfectă, însă este dezvoltată în așa fel încât să permită modificări viitoare și adăugări de noi funcționalități.

O fișă de disciplină a Universității Tehnice din Cluj-Napoca, reprezintă o colecție de secțiuni (10 secțiuni) care păstrează informații legate de materia pentru care a fost creată. De aceea, o aplicație de gestiune poate reduce enorm efortul personalului care se ocupă de redactarea, modificarea și actualizarea fișelor de disciplină. Aplicația a fost special dezvoltată pentru a satisface toate nevoile unei persoane atunci când se ocupă de fișele de disciplină (multe dintre input-uri sunt de formă autocomplete sau combobox). Cu toate acestea nu toți utilizatorii pot avea acces la toate funcționalitățile aplicației deoarece este în firea omenirii să greșească, și să încerce să își îndeplinească scopurile proprii abuzând de puterea care le-a fost oferită. De aceea, aplicația dispune de un mijloc de împărțire a drepturilor în funcție de nevoile utilizatorului. Administratorii oferă sau iau drepturi astfel încât utilizatorul poate accesa sau pierde accesul spre anumite funcționalități disponibile în aplicație.

Deoarece universitatea deține mai multe facultăți și se împrășteie pe mai multe domenii de studiu, aplicația se mulează pe aceste nevoi, astfel că funcționalitatea nu este strict focalizată doar pe Facultatea de Automatică și Calculatoare. Operațiile de gestiune pot fi utilizate de toate cadrele didactice indiferent de facultatea de care aparțin. Persoanele trebuie doar să primească un cont și să li se dea drepturi. Un mare avantaj al aplicației este generarea automată a fișei de disciplină în format PDF la apăsarea unui buton.

2.2. Rezumat obiective

Principalul obiectiv al proiectului este dezvoltarea unei aplicații web care funcționează pe arhitectură Client-Server împreună cu un server de backend. Acest sistem trebuie să îndeplinească următoarele funcționalități:

- cerințe funcționale
 - conectare securizată în aplicație
 - autorizare pe baza de JWT Token pentru folosirea funcționalităților
 - role management atât în aplicația web cât și în WebAPI
 - vizualizare, adăugare, modificare, ștergere elemente din arhitectură instituțională: instituții, facultăți, departamente și domenii de studiu
 - vizualizare, adăugare, modificare, ștergere utilizatori și cadre didactice
 - vizualizare, adăugare, modificare discipline
 - vizualizare, adăugare, modificare, ștergere fișe de disciplină
 - vizualizare istoric al fișelor de disciplină
 - vizualizare audit de securitate
 - export în format PDF a fișelor de disciplină
- cerințe nonfuncționale
 - flexibilitate: aplicația se adaptează ușor nevoilor în continuă schimbare a clienților
 - documentation: aplicația are documentația autogenerată de Swagger UI [1]
 - securitate: informațiile oferite de utilizator sunt verificate atât pe front-end cât și pe back-end, conectarea la front-end se face doar prin JWT Token-ul generat de back-end, iar funcționalitățile aplicației de back-end pot fi accesate doar de persoanele autorizate
 - eficiență: aplicația este construită în așa fel încât să folosească minimumul necesar de resurse
 - reutilizabilitate: aplicația de front-end folosește componente definite generic \Leftrightarrow aceeași componentă, dar cu funcționalitate schimbată datorită propusurilor care intră în componentă

Capitolul 3. Studiu Bibliografic

Pentru realizarea proiectului de licență a fost nevoie de documentare intensă pe mai multe domenii care țin atât de programare (pe back-end cât și pe front-end), de proiectare, de implementare cât și de securizarea API-ului și protecția datelor cu caracter personal:

- Back-end:
 - limbaj de programare: C#
 - framework: ASP.NET Core
 - API: Rest API
- Front-end:
 - limbaj de programare: TypeScript
 - framework: React
- Bază de date: PostgreSQL
- Arhitectură(design pattern): Model-View-Controller pentru partea de back-end, Client-Server pentru aplicația întreaga
- Securitate
 - Protocol de comunicare: HTTPS
 - Algoritm de criptare: SHA384
 - Autorizare pe bază de jeton(token): JWT Token

Documentarea a fost realizată atât pe baza cărților care sunt prezentate în secțiunea de bibliografie cât și din documentațiile oficiale ale diverselor framework-uri și librării folosite.

3.1. Back-end

Luând în considerare experiența profesională acumulată în urma cursurile și laboratoarelor pe care le-am parcurs de-a lungul celor 4 ani de facultate am decis să utilizez limbajul C# pentru programarea back-end-ului lucrării de licență deoarece în opinia mea este superior limbajului Java, având actualizări mai frecvente și o structura mai prietenoasă a librăriilor. Structura arhitecturală a unui proiect de tipul ASP.NET Core se mulează pe principiile MVC(care au fost studiate și aprofundate la facultate) ceea ce mi-a oferit un mediu familiar.

3.1.1. Limbaj de programare

Limbajul de programare ales este C#, un limbaj dezvoltat și întreținut de compania Microsoft care permite utilizatorilor să își construiască aplicațiile dorite(personale sau enterprise) pe toate tipurile de platforme. Utilizarea limbajului este foarte diversificată, putem menționa aplicații web, aplicații desktop, aplicații mobile chiar și jocuri. Eu am utilizat limbajul pentru a-mi construi web API-ul pentru back-end.

C# este un limbaj de tip OOP care mi-a permis structurarea într-un mod eficient și elegant a pachetelor de date(namespace-urilor în aplicațiile .Net). Limbajul de nivel

înalt vine la pachet cu o mulțime de librării care ajută programatorul să codeze mai repede, mai clar și mai eficient, totodată respectând principii de cod curat(clean code) [2] .

3.1.2. Framework

Microsoft oferă gratis framework-ul ASP.NET Core, un framework open-source care stă la baza creerii de aplicații pe orice tip de platformă. Pentru implementarea aplicației mele am folosit cea mai nouă versiune a framework-ului: ASP.NET Core 6.0 lansată în data de 8 noiembrie 2021.

Pe lângă framework-ul propriu-zis, Microsoft oferă și o documentație vastă a acestui framework [3]. Faptul că framework-ul este open-source permite comunității să vină cu reproșuri, observații sau chiar îmbunătățiri. Datorită comunității mari de programatori .Net este foarte ușor să gestionezi erorile peste care dai în cadrul dezvoltării aplicației. Tot datorită faptului că framework-ul este open-source există o mare diversitate de librării compatibile cu ASP.NET Core, care ajută mult programatorul în timpul dezvoltării.

3.1.3. API

Pentru implementarea API-ului am ales arhitectură REST care este un ansamblu de constrângeri arhitecturale. Deoarece aplicația nu necesită paradigme de comunicare complicate, comunicarea se poate realiza folosind doar apeluri HTTP(GET, POST, PUT, DELETE), o arhitectură REST API îndeplinește toate cerințele de funcționare. Luând în considerare dezvoltarea ulterioară a aplicației un REST API permite comunicarea cu orice tip de client, fie el web, mobile sau desktop, ceea ce reduce dramatic problemele viitoare de compatibilitate cu clienți noi apăruiți.

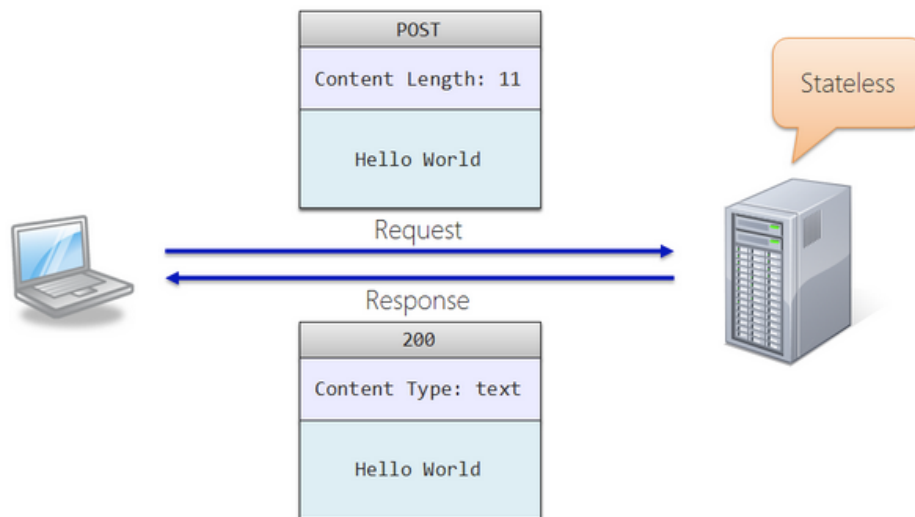


Figura 3.1: Comunicare cu un REST API ¹

Un API poate fi considerat REST dacă îndeplinește următoarele condiții:

- arhitectura trebuie să fie una de tip client-server

¹<https://www.pluralsight.com/blog/tutorials/representational-state-transfer-tips>

- comunicarea se face pe baza requesturilor HTTP
- informații legate de client nu sunt stocate între request-uri
- requesturile nu sunt interconectate
- API-ul este structurat pe straturi(layers) pentru a nu expune involuntar date confidențiale despre client
- mesajele trebuie să contină suficiente informații pentru ca userul să știe ce acțiune să abordeze ca răspuns

3.2. Front-end

Luând în considerare experiența redusă de lucru cu partea de front-end a aplicațiilor web, dar luând în calcul pasiunea mea pentru front-end consider că am făcut alegerea bună pentru selectarea tehnologiilor. Deoarece am dorit să învăț lucruri practice în timp ce îmi realizez lucrarea de licență, am ales framework-ul React, la care am adăugat mai multe librării pentru ușurarea dezvoltării acesteia. Limbajul de programare l-am ales deoarece oferă o mai bună înțelegere a codului. Concomitent limbajul de programare Typescript este recomandat pentru aplicații de lungă durată, pe care ar putea lucra mai mulți programatori datorită codului explicit și numărului redus de bug-uri care ar putea apărea în producție.

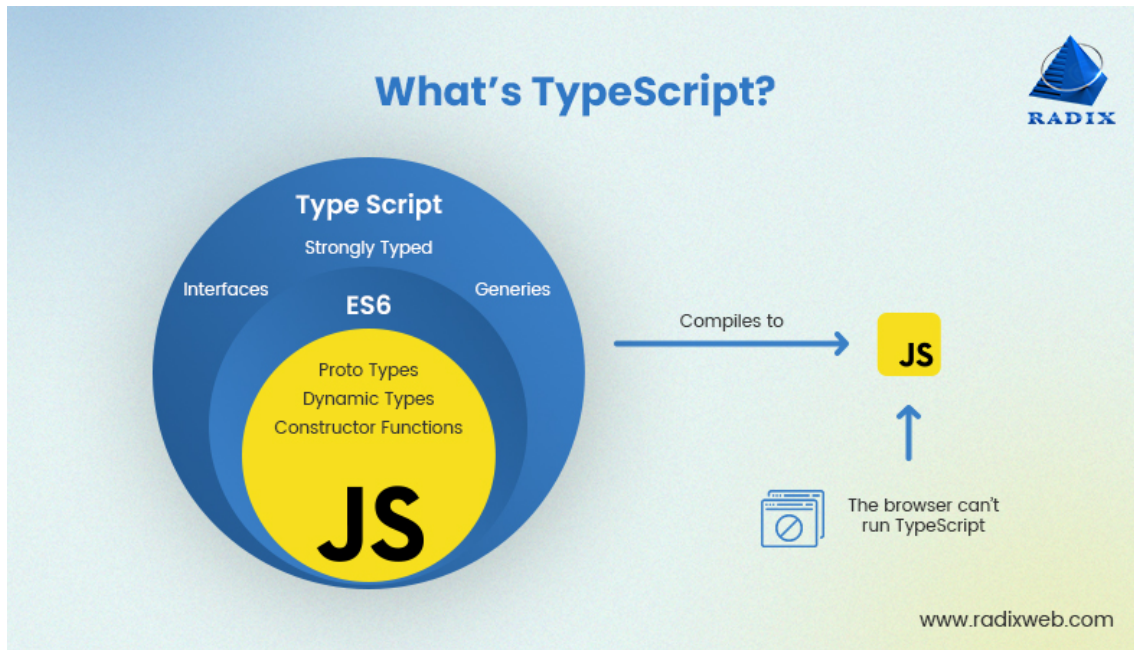
3.2.1. Limbaj de programare

Typescript [4] este un limbaj de programare dezvoltat și întreținut tot de către compania Microsoft, care este folosit pentru crearea paginilor web interactive. Acesta este un wrapper peste limbajul Javascript pur care aduce în cod conceptele de clase, tipuri, interfețe și garantează o mai mare siguranță a codului care se scrie. În mare parte un programator de front-end care utilizează Typescript recurge la cunoștințele precedente de Javascript deoarece se folosește aceeași abordare în scrierea codului.

Acest limbaj de programare avertizează programatorul din timp asupra diverselor erori care pot apărea(ex. field-urile unui obiect nu corespund cu field-urile așteptate de funcție), pentru a nu trimite în "producție" un cod care ar urma să provoace neplăceri utilizatorilor prin funcționarea necorespunzătoare. Javascript-ul nu ar semnaliza nici un semnal de alarmă, totul ar merge mai ușor și mai repede, însă ar putea exista mici neconcordanțe care s-ar transforma în bug-uri atunci când este trimis codul spre utilizator și folosit în rutină zilnică de acesta.

Un utilizator neexperimentat sau care abia începe programarea web, ar putea avea probleme la scrierea codului de Typescript, de aceea se recomandă începătorilor să învețe Javascript prima dată. Însă odată ce se capătă suficientă experiență, Typescript-ul este o unealtă extrem de utilă atât pentru programatorul care a început proiectul, cât și pentru programatorii care urmează să continue proiectul.

Typescript-ul comparativ cu Javascript-ul este un limbaj de programare compilat nu interpretat, ceea ce înseamnă că erorile pot fi găsite și rezolvate înainte de execuția codului.

Figura 3.2: Ce e Typescript-ul? ¹

După cum scrie și în figura de mai sus, clientul/consumatorul(browser-ul) nu știe să interpreteze cod Typescript, de aceea codul Typescript este compilat în cod Javascript.

3.2.2. Framework

React este o librărie open-source oferită de Meta și menținută de Meta și comunitate pentru crearea interfețelor interactive web. Datorită comunității mari librăria este într-o continuă evoluție cu update-uri periodice. React se bazează pe principiul de încapsulare a elementelor în componente, care au stări proprii și știu să își modifice comportamentul în funcție de starea lor internă.

Componentele de tip React pot fi văzute asemenea unor funcții care primesc niște parametri de intrare și returnează o resursă care poate fi tot de tipul funcție sau componentă HTML. Datorită acestui lucru, programatorii pot scrie o componeta cu un comportament general care poate fi reutilizată în mai multe părți ale aplicației prin schimbarea parametrilor de intrare. Un exemplu poate fi văzut în figura 3.3.

Datorită comunității mari și interesului ridicat pentru această librărie s-a dezvoltat un întreg ecosistem de librării externe(3rd party library) care se folosesc de funcționalitățile de baza în React și definesc metode mai ușoare de a realiza o funționalitate. Exemple de 3rd party libraries pe care le-am folosit în cadrul dezvoltării aplicației web sunt:

- Redux
- Axios
- Js-cookie
- React router dom
- Material UI
- Styled components

¹<https://radixweb.com/blog/typescript-vs-javascript>

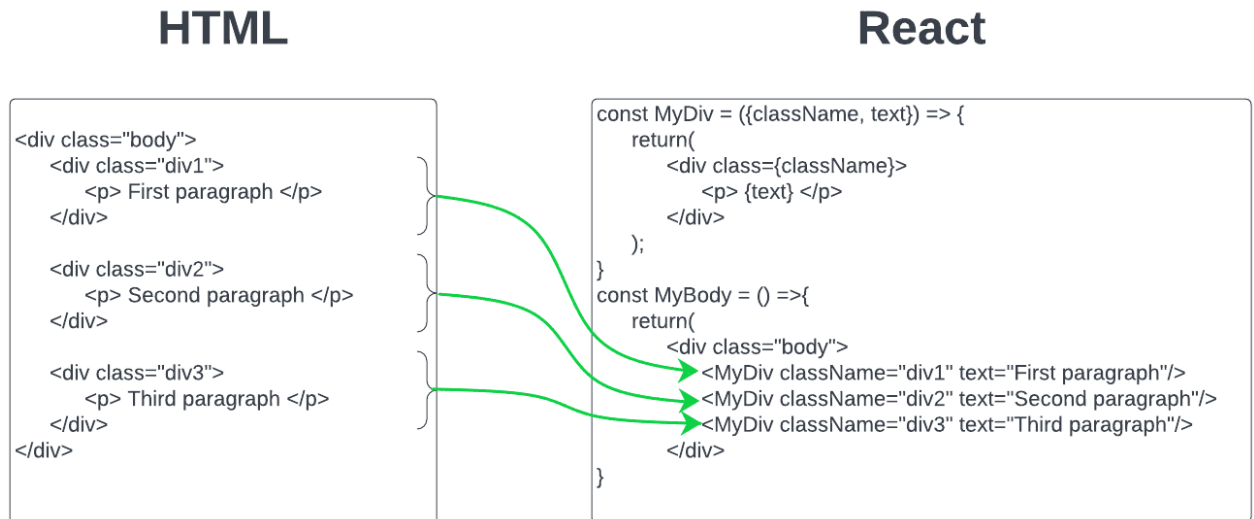


Figura 3.3: Definirea de componente și reutilizarea lor în cadrul altor componente React

Redux

Redux [5] este o librărie de Javascript care are rolul de a centraliza și gestiona starea aplicației. Această librărie funcționează foarte bine împreună cu React deoarece acționează ca o bază de date care învește aplicația de React. Acest lucru este benefic pentru aplicație deoarece orice modificare a stării aplicației poate fi interpretată și gestionată corespunzător fluxului de lucru.

Redux Toolkit este un pachet dezvoltat tot de comunitatea care lucrează pentru librăria Redux, care a devenit abordarea recomandată de dezvoltator. Redux Toolkit ușurează muncă programatorului printr-o arhitectură structurată mai bine, în multe funcții care îndeplinesc doar un singur lucru ușor de înțeles.

Axios

Axios [6] este un pachet pentru node.js bazat pe noțiunea de promisiuni(Promise) din Javascript care este folosită pentru a manipula request-urile de date de tipul HTTP. Axios poate fi folosit atât pe un server de node.js unde se translateaza la request-uri native de HTML, cât și pe browsere, request-urile translatându-se în request-uri de tipul XMLHttpRequests.

Js-cookie

Js-cookie [7] este un pachet de Javascript open-source, menținut de comunitate, care face posibilă utilizarea Browser cookie-urilor mai ușor, folosind mai puțin cod, dar cu tot aceeași funcționalitate.

React router

React router [8] este un pachet de Javascript care face posibilă routarea dinamică în cadrul aplicației web.

Rutarea dinamică este un proces în care routerul poate să trimită date mai departe bazându-se pe starea actuală a aplicației

Material UI

Material UI [9] este un pachet de componente predefinite pentru aplicații web React. Obiectele sunt gata pregătite pentru producție. Dezvoltatorii Material UI folosesc componentele de baza HTML: div-uri, paragrafe, input-uri, etc.. împreună cu Javascript/Typescript pentru a crea componente mai inteligente și sofisticate pregătite pentru a fi folosite de orice programator în cadrul dezvoltării unei aplicații web personale sau chiar și de entreprise.

Material UI se ocupă și de stilizarea componentelor, dar oferă utilizatorului posibilitatea să își creeze o tema personalizată folosind funcții predefinite în librărie. Tema se folosește de Javascript pentru a genera un fișier .css care să preia proprietățile CSS specificate în tema.



Figura 3.4: Avantajele scrierii codului CSS în format Javascript 1

Styled components

Styled components [10] este o librărie care combină funcționalitatea CSS-ului cu utilitatea Javascript-ului. Această librărie permite scrierea de cod CSS într-un fișier Javascript sau Typescript .js/.ts în componente definite de programator cu aceeași sintaxă utilizată în fișierele .css. Acest lucru duce la schimbări dinamice ale componentelor care

¹<https://dev.to/rleija/5-reasons-to-go-with-css-in-js-for-your-next-application-43m>

își pot modifica stilul în funcția de starea aplicației.

Componentele definite cu ajutorul librăriei sunt niște wrapper-e formate deasupra unu-i container primitiv HTML: div, span, sau chiar deasupra unei alte componente React, care aplică proprietățile CSS definite în declararea acestora.

```
// Greet.js
import React from 'react';
import styled from 'styled-components';
import { primaryTextColor } from './constants';

const Title = styled.h1`
  font-size: 1.5em;
  text-align: center;
  color: ${primaryTextColor};
`;

const Greet = () => <Title>Hi there, I'm a styled component!</Title>
```

Figura 3.5: Exemplu de componentă definită cu ajutorul styled components 1

3.3. Baza de date

PostgreSQL [11] este o bază de date open-source, relațională și gratis. Acest sistem de baze de date are peste 30 de ani de dezvoltare activă, cu nenumărate update-uri și noi funcționalități în dezvoltare. Ultima actualizare a fost făcută în dată de 19 mai 2022 ceea ce redă importanța, cererea și utilizarea crescută a acestui tip de baze de date. Datorită comunității mari și interesului despre acest produs, există numeroase extensii care fac posibilă utilizarea unor funcționalități non-existente în pachetul de baza al librăriei, de exemplu: bazele de date PostgreSQL nu au funcționalitatea de tabele temporale (temporal tables), dar folosind o extensie creată de comunitate, această funcționalitate poate fi adăugată bazei de date.

3.4. Arhitectura

Arhitectura (design pattern-ul) pe care am ales să o folosesc pentru dezvoltarea aplicației de licență este Client-Server. Am ales această arhitectură, deoarece aplicația constituie un server centralizat care se ocupă de procesarea datelor și o aplicație grafică care permite unui utilizator fără cunoștințe în domeniul informaticii să utilizeze la maxim puterea serverului și funcționalitățile sale. Luând în considerare informațiile prezentate mai sus clientul este reprezentat de browsere web care permit rularea interfeței grafice realizate cu ajutorul librăriei React. Serverul realizat pe baza framework-ului ASP.NET Core 6 așteaptă cereri din partea clienților și apoi răspunde cererilor. Datorită faptului că cererile adresate serverului sunt procesate cu ajutorul thread-urilor, serverul este scalabil și permite accesul mai multor utilizatori simultan fără a pierde din performanță.

Arhitectura folosită pe partea de back-end este Model View Controller. Am ales această arhitectură deoarece permite structurarea unei aplicații complete în module mai

mici de o complexitate mai redusă ceea ce duce la soluționarea probleme într-un timp mult mai mic. În plus, împărțirea în module oferă posibilitatea mai multor echipe de programatori să lucreze separat la funcționalități diferite fără să influențeze într-un mod negativ activitatea celorlalte echipe. Ultimul fapt precizat nu este neapărat util pentru cazul meu, deoarece există doar o singură echipă formată dintr-o singură persoană care lucrează la proiect, dar arhitectura MVC asigură funcționarea în parametri optimi și pe viitor, dacă se alege continuarea proiectului.

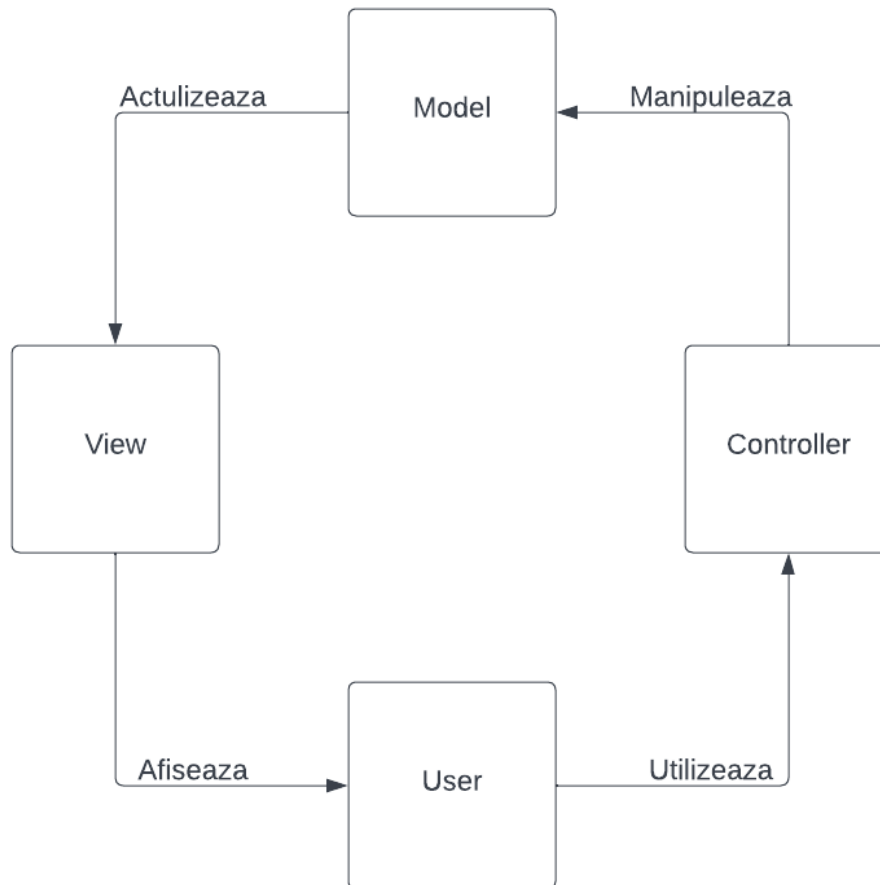


Figura 3.6: Arhitectura model view controller

Componentele arhitecturii MVC sunt:

- **Modelul:** componentă care încapsulează datele aplicației, date care reprezintă entități folosite atât pentru inițial generarea bazei de date, cât și mai apoi pentru manipularea acestora
- **Controller:** componentă care procesează cererile utilizatorilor, prin comunicare cu componentă Model și apoi prin trimiterea datelor utile spre componenta View
- **View:** componentă responsabilă de interpretarea cererilor clientului și apelarea serviciilor componente de Controller pentru a primi datele necesare satisfacerii nevoilor clientului

3.5. Securitate

Asigurarea unui mediu sigur pentru confidențialitatea și intimitatea datelor utilizatorului a fost și încă este un domeniu complex și în plină dezvoltare. Cu cât evoluează tehnologia, cu atât există mai multe ocazii de dezvoltare ale unor aplicații dăunătoare intimității utilizatorilor. Aceste aplicații pot fură, distribui și chiar ține sub asediu date importante ale unor companii/ persoane importante care ar putea avea un impact puternic asupra statutului financiar sau asupra reputației companiei/ persoanei. În aceste circumstanțe am ales să adaug un strat de securitate aplicației de Back-end.

3.5.1. Protocol de comunicare

HTTP este un protocol la nivel de aplicație creat pentru a satisface comunicarea între web client și web server. HTTP-ul este contruit pe baza clasicei arhitecturi client-server(un client deschide o conexiune și face o cerere, iar serverul răspunde) și este un protocol fără stare(stateless), ceea ce înseamnă că nu salvează date între request-urile utilizatorilor. În ciuda faptului că funcționează și satisface nevoile de performanță, acest protocol nu este sigur, datele utilizatorilor fiind transmise la server în format clar(plain text) ceea ce poate furniza metode nelimitate de exploatare a breșelor de securitate ale serverului vizat.

Cu ideea îmbunătățirii protocolului a fost creată o nouă versiune: HTTPS, care se aliniază la toate funcționalitățile protocolului HTTP, dar datele sunt transmise criptat în timpul comunicării.

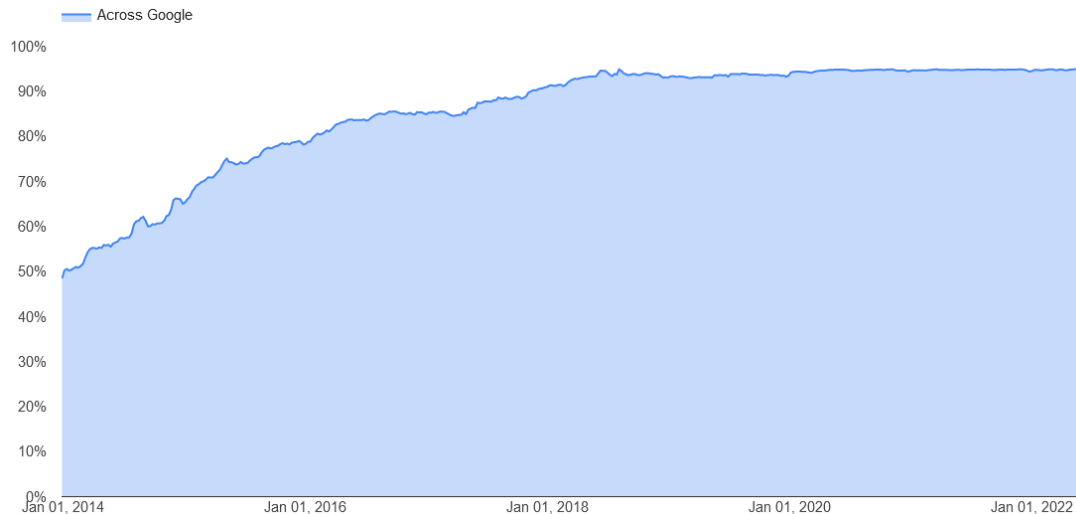


Figura 3.7: Creșterea treptată a adaptării paginilor web la protocolul HTTPS ¹

Datele sunt criptate folosind algoritmi de encriptare asimetrică, ceea ce înseamnă că sunt generate chei publice și chei private, cheia publică poate fi decriptată doar de cheia privată, iar cheia privată poate fi decriptată doar cu ajutorul cheii publice.

¹<https://transparencyreport.google.com/https/overview?hl=en>

3.5.2. Algoritm de criptare

Pentru a asigura siguranță datelor utilizatorului, în special pentru parole, folosesc algoritmul SHA384. Acest algoritm garantează rezultate unice care nu mai pot fi vizualizate în clar(plain text). Orice mică modificare a parolei poate genera rezultate neașteptate în parola criptată ceea ce oferă confidențialitate ridicată datelor utilizatorului. Algoritmul este deja implementat în cadrul librăriei System.Security.Cryptography ceea ce face utilizarea acestuia foarte ușoară.

3.5.3. Autorizare pe baza JWT Token

JWT Token este un obiect standardizat internațional în cadrul internetului care reprezintă consimțământul userului pentru a utiliza un API. Acest token este format din 3 părți: header, payload și verify signature. În payload se vor regăsi date despre utilizatorul care încearcă să accese API-ul.

Capitolul 4. Analiză și Fundamentare Teoretică

Scopul acestui capitol este de a explica principiile funcționale ale aplicației implementate. Aici voi descrie soluția propusă dintr-un punct de vedere teoretic.

4.1. Arhitectura

Aplicația este structurată în 3 straturi: front-end, back-end și bază de date. Utilizatorul creează cereri din aplicația de front-end, care apoi trimite respectivele cereri către server unde sunt procesate. După o verificare aprigă se stabilește dacă utilizatorul are drepturile necesare executării operației cerute și în urma autorizării back-end-ul comunică cu baza de date. După ce s-a primit răspunsul bazei de date și s-au făcut operațiile necesare îndeplinirii cererii clientului, back-end-ul trimite un răspuns care este interpretat de către front-end, astfel interfața se actualizează în funcție de răspunsul primit.

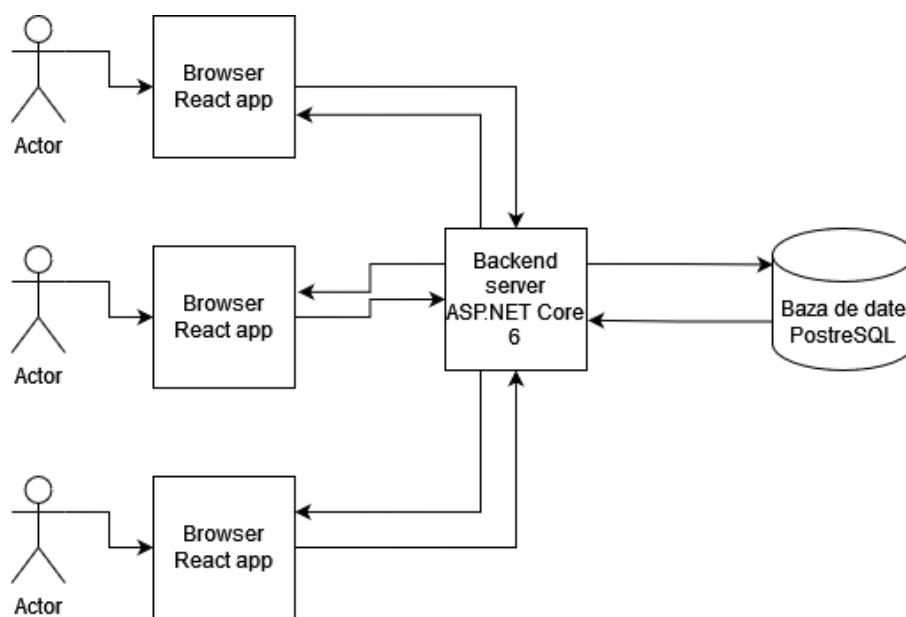


Figura 4.1: Diagrama conceptuală și componentele aplicației

4.2. Back-end

4.2.1. Arhitectura

Back-end-ul aplicației, care a fost scris cu ajutorul framework-ului ASP.NET Core 6, se mulează pe arhitectură Model View Controller. Aplicația este împărțită în 3 straturi (layers): model, view și controller, fiecare având o sarcină unică de îndeplinit.

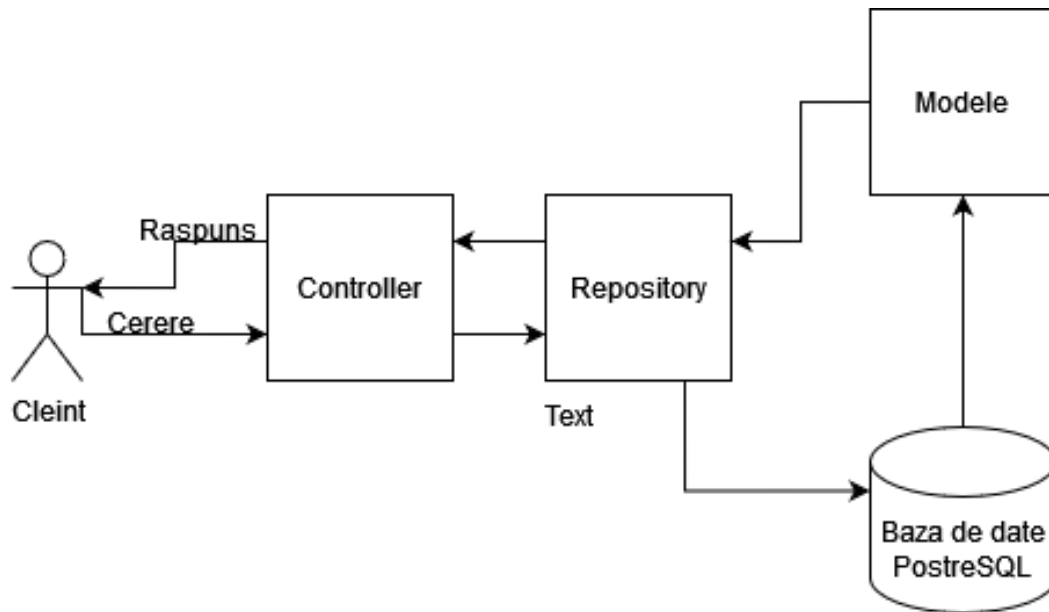


Figura 4.2: Diagrama conceptuală a aplicației de back-end

4.2.2. Baza de date

Model

Modelul este partea declarativă a tipurilor de date care se vor folosi pentru a îndeplini funcționalitățile necesare rulării în parametri optimi a aplicației. Am urmat tehnica de programare code-first a framework-ului ASP.NET Core 6, ceea ce m-a obligat să definesc modelul pentru a se putea genera mai apoi tabelele bazei de date și relațiile dintre acestea după modelul de clase.

Framework-ul ASP.NET Core 6 pune la dispoziție un feature denumit DbContext care acționează ca un lipici puternic ce pune laolaltă partea de baze de date cu partea de model și îți permite, în calitate de programator, să controlezi cum se vor lega conexiunile între tabele ce urmează să fie generate pe baza claselor definite în model. Pentru a beneficia de toată puterea și utilitatea framework-ului, gestionarea claselor și în general a layer-ului Model se va face cu ajutorul Entity Framework Core. Entity Framework Core permite programatorului să lucreze cu baza de date folosind obiectele deja definite în Model.

Controller

Controller-ul este layer-ul care procesează cererile adresate serverului de către client. Layer-ul de controller este format din mai multe seturi de acțiuni, grupate în funcție de tipul acțiunii și context, pentru a favoriza principiile de clean-code în scrierea controller-ului, principiile de securitate în autorizarea diferitelor tipuri de utilizatori la diferite tipuri de acțiuni sau îndeplinirea diferitelor grupări de condiții.

Respectând standardul impus de comunitate pentru începerea unor proiecte de ASP.NET Core 6, layer-ul de controllere ar trebui să fie situat într-un folder, aplasat la nivel cu rădăcina sistemului de directoare numit Controllers. Un controller trebuie să moștenească clasa ControllerBase și să aibă setată o ruta pe care clientul să o acceseze

atunci cand dorește să efectueze acțiuni. Rutele ar trebui să fie distincte în rândul celorlalte controllere pentru a nu există probleme de rutare, care ar putea împiedică fluxul de lucru al clienților.

View

Deoarece aplicația dezvoltată de mine folosește o aplicație web făcută pe baza framework-ului React, funcționalitățile de server-side-rendering(procesarea unei cereri și trimiterea unui fișier html drept răspuns) puse la dispoziție de către ASP.NET Core 6 pentru a realiza partea de View a API-ului nu sunt folosite.

Deoarece ASP.NET Core 6 este un produs pentru care se depune interes și multe ore de muncă, s-a dezvoltat un framework care sare în ajutorul ASP.NET Core 6 pentru gestionarea bazelor de date. Entity framework este un framework open-source dezvoltat de către compania Microsoft pentru a servi drept un Object-Relation Mapping framework¹ (pe scurt ORM).

Entity framework servește în cadrul layer-ului pentru acces la date(Data Access Layer) și oferă o imagine exactă a bazei de date. Entity framework poate fi configurat în 3 moduri:

- Definirea codului prima dată(code first approach)
Această metodă presupune crearea manuală a claselor aplicației prima dată, se recomandă folosirea acestei metode atunci când se lucrează cu o baza de date proaspăt făcută. Odată create clasele Entity framework pune la dispoziție un sistem de migrări, care în urmă unei comenzi scrise în linia de comandă(există mai multe metode, dar cea mai simplă este: add-migration [migration-name]) generează un fișier de migrare care are 2 funcții: Up și Down. În urma executării comenzii: update-database se execută funcția Up, astfel se creează un nou context al bazei de date unde sunt definite toate modificările: adăugări de table, coloane, editări de nume, tipuri de date, ștergere de tabele sau coloane.
- Definirea bazei de date prima dată(database first approach)
Această metodă funcționează mai bine atunci când baza de date există deja și se dorește actualizarea ei. În primul rând este nevoie de o baza de date existentă și relații de dependență deja stabilite. Odată ce s-a actualizat baza de date cu noile tipuri/tabele/coloane se rulează o comandă în linia de comandă care generează sau actualizează clasele modificate în urma modificării bazei de date.
- Definirea modelului prima dată(model first approach)
Această metodă se bazează pe o unealtă preintegrată în Visual Studio: Visual Studio Entity Data Model Wizard. Folosind acea unealtă programatorul poate defini entități și relațiile dintre ele apoi poate genera baza de date corespunzătoare lor.

Metoda aleasă de mine pentru dezvoltarea bazei de date este: code first approach. Entity framework dispune de un sistem de migrări care monitorizează continuu baza de date. La adăugarea primei migrări s-a creat un tabel __EFMigrationsHistory care atunci când se adaugă o nouă migrare și se actualizează baza de date, se actualizează și el cu denumirea noii migrări și versiunea curentă a produsului.

¹ORM = tehnică de conversie a datelor unei baze de date în obiecte(folosite de limbaje orientate pe obiect), ceea ce rezultă într-o baza de date "virtuală" accesabilă prin obiecte

	MigrationId [PK] character varying (150)	ProductVersion character varying (32)
1	20220404101004_InitialMigration	6.0.3
2	20220501124720_licenta_1	6.0.3
3	20220523201740_licenta_teacher	6.0.3

Figura 4.3: Tabelul __EFMigrationsHistory

4.3. Front-end

4.3.1. Arhitectura

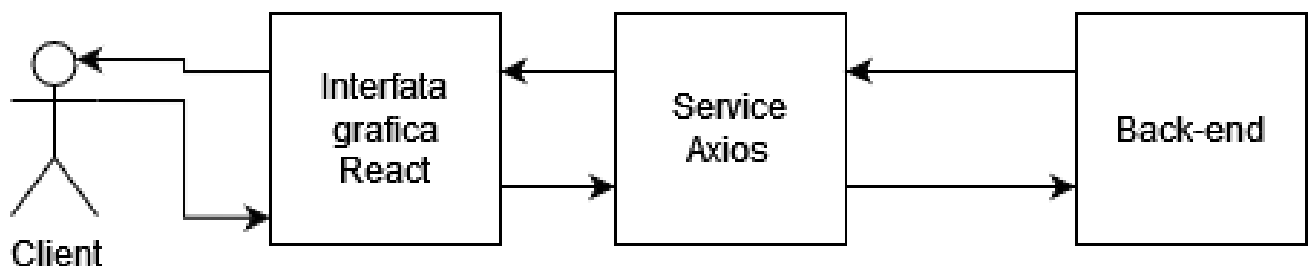


Figura 4.4: Diagrama conceptuală a aplicației de Front-end

În această secțiune voi prezenta la nivel teoretic arhitectura pe care am folosit-o pentru aplicația de front-end. Arhitectura propusă de mine este formată din 4 straturi (layers), fiecare împărțit în mai multe secțiuni. Layer-ele pe care le-am propus sunt:

- common
- components
- data-access
- pages

În următoarele paragrafe voi explica de ce am ales această împărțire și ce beneficii aduce. Explicațiile le voi începe în ordinea transferului de date de la user la aplicația de back-end.

În layer-ul de pages se află componentele React responsabile cu randarea paginilor atribuite fiecărei ruta declarată în router. Împărțirea se face pe directoare, astfel încât fiecare componeta are un fișier main, în care este declarată componenta și funcționalitatea sa, și un fișier de stiluri unde sunt declarate containere cu ajutorul librăriei styled-components. În directorul fiecare pagini sunt declarate și componentele scrise și funcționale doar pentru pagina respectivă.

În layer-ul de components sunt declarate și implementate toate componentele reutilizabile din cadrul aplicației. Acest layer este separat de toate celelalte deoarece îl putem vedea ca un schelet pe care se adaugă mușchi, organe etc. Componentele din layer-ul de components sunt doar niște carapace goale care nu știu să facă nimic singure. Ele trebuie instantiate în componentele din layer-ul de pages unde își primesc funcționalitatea prin parapetree de intrare (props-uri).

În layer-ul de common se află obiecte reutilizabile cum ar fi: string-uri predefinite pentru rute, stiluri, resurse etc.. . Acest layer este accesat din pages sau data-acces pentru a lua elemente cu valori predefinite sau funcții care pot fi reutilizate: un exemplu ar fi o funcție care parsează o dată UTF într-un format anume, prestabilit de programator, deoarece nu se vrea adăugarea acestei funcții în toate pachetele, poate fi declarată o singură dată și apoi reutilizată.

În layer-ul de data-access se regasesc funcțiile necesare comunicării cu back-end-ul, tipurile de date predefinite și layer-ul de redux. Funcțiile necesare comunicării cu aplicația de back-end se folosesc atât de layer-ul common cât și de pachetul de tipuri, care este definit la același nivel de director. În pachetul cu tipuri sunt definite interfețele necesare rulării în parametri optimi a typescript-ului. Layer-ul de Redux: store, am hotărât să-l definesc la același nivel cu service-ul și tipurile din motive de conveniență a importurilor și din motive de asigurarea a viitorului(future proofing) când se va muta partea de service în layer-ul de Redux.

4.4. Securitate

Securitatea în aplicația dezvoltată de mine este asigurată atât pe partea de front-end cât și pe partea de back-end. În primul rând se folosește protocolul de comunicare HTTPS pentru a lega conexiuni securizate între back-end și front-end. Datorită acestei conexiuni securizate utilizatorul nu mai trebuie să își facă griji pentru eventuale scurgeri de informații, precum: parole, sau date biometrice, sau date cu caracter personal. Astfel dacă cererea este procesată folosind HTTP și există o persoană rău intenționată care fură și inspectează pachetele de date, poate să vadă atât credențialele pentru login cât și adresa către care este îndreptată cererea, însă dacă se folosește protocolul de comunicare HTTPS, persoană rău intenționată va vedea doar o serie de litere și cifre care nu pot fi interpretate de către creierul uman.

La fel de important ca și folosirea unui protocol de comunicare securizat este și testarea și validarea input-ului userului. Nu toți userii sunt persoane bine intenționate care doresc doar să folosească aplicația pentru a-și îndeplini scopurile personale, există și persoane care încearcă să creeze disconfort persoanei/firmei care prestează servicii prin intermediul mijloacelor de business online precum site-uri web, prin furare, distrugere sau ținere sub asediu a datelor importante sau confidențiale. Astfel că toate informațiile folosite într-o cerere adresată serverului trebuie să fie validate pentru a nu se introduce abuzuri destinate breșelor de securitate ale serverului. Pentru a produce o cerere validă și nevătămătoare sunt folosite procedee de validare a input-urilor ¹.

Validarea inputurilor este un proces obligatoriu pentru buna funcționare a aplicației și trebuie îndeplinit atât de aplicația de front-end cât și de aplicația de back-end. Înainte de trimiterea datelor spre server, aplicația web trebuie să confirme faptul că utilizatorul a introdus date corecte în field-urile necesare. Doar după verificarea datelor aplicația web poate să trimită cererea către aplicația de back-end, însă în cazul în care nu s-au atins standardele de validare a input-urilor, cerere nu se trimite, iar utilizatorul este atenționat

¹input = spațiu în cadrul aplicației web destinat inserării de date din parte clientului care urmează să fie trimise și procesate de server, sau datele introduse de utilizator adresate serverului

prin diverse metode de faptul că datele introduse de el nu respectă cerințele de validare a input-urilor.

În urmă obținerii gradului de încredere în datele adăugate de utilizator, o cerere adresată server-ului care se ocupă de administrarea operațiilor de back-end este emisă. La rândul său serverul trebuie să verifice autenticitatea și validitatea datelor primite. Însă înainte de validarea respectivelor date, serverul trebuie să stabilească dacă utilizatorul are dreptul de a accesa respectiva funcționalitate. Această operație de autorizare poate fi bazată pe un JWT Token emis în urmă autentificării utilizatorului la server. Acel JWT Token poate să conțină informații despre utilizator și în urma verificării respectivelor informații se poate asigura dacă utilizatorul are permisiunile necesare efectuării operației solicitate. În cazul în care datele ținute în Token nu sunt valide sau Tokenul a expirat, utilizatorul este înștiințat de acest lucru prin afișarea unui mesaj de către aplicația de front-end pe interfață grafică, prin diverse metode.

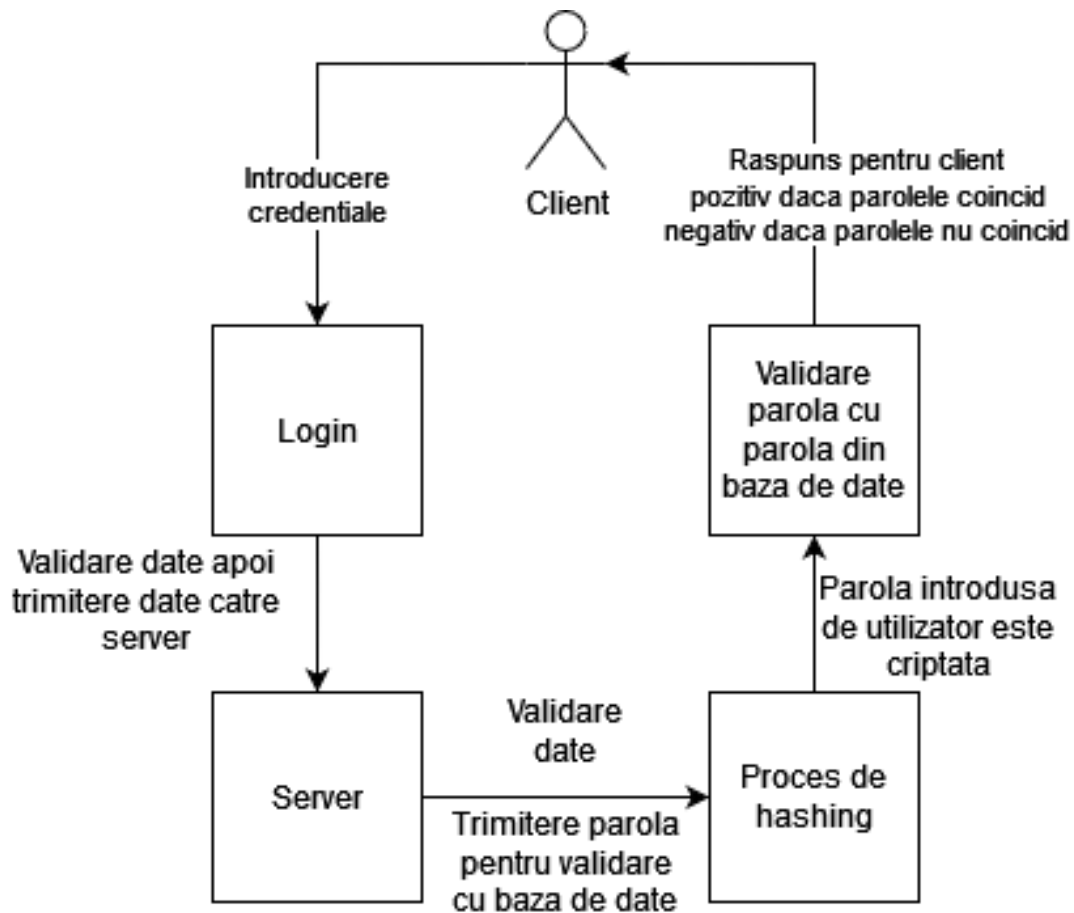


Figura 4.5: Procesul de validare a parolei

Pentru menținerea datelor utilizatorului în deplină siguranță, utilizez un algoritm de criptare care ține parolele în baza de date într-un format de hash ¹. Algoritmul folosit pentru criptarea datelor este SHA384 ². La o cere de logare, deoarece parolele sunt ținute în baza de date în format de hash, pentru a verifica autenticitatea parolei, parola primită de la utilizator este trecută prin algoritmul SHA384, iar dacă rezultatele coincid, atunci parolele sunt identice. Este bine de menționat faptul că orice modificare în interiorul

parolei introduse de utilizator provoacă schimbări drastice la nivelul hash-ului rezultat.

4.5. Fișa disciplinei

Fișa disciplinei A reprezintă un fișier PDF care conține informațiile necesare identificării și înțelegerii compoziției unei discipline. Studentul poate să citească fișa de disciplină pentru a culege informații referitoare la programă, înainte de a începe cursurile. O fișa de disciplină este un fișier complex alcătuit din 10 secțiuni, fiecare conținând informații unice. Secțiunile unei fișe de disciplină sunt:

- Date despre program
- Date despre disciplină
- Timpul total estimat
- Precondiții (acolo unde este cazul)
- Condiții (acolo unde este cazul)
- Competențele specifice acumulate
- Obiectivele disciplinei
- Conținuturi
- Coroborarea conținuturilor disciplinei cu așteptările reprezentanților comunității epistemice, asociațiilor profesionale și angajatorilor reprezentativi din domeniul aferent programului
- Evaluare

¹hash = procesul prin care un text este transformat în urma executării unui algoritm într-un alt text ilizibil

²SHA384 = algoritm standard de criptare a datelor uni-direcțional \Leftrightarrow nu poate fi refăcut textul în urma procesului de hashing

Capitolul 5. Proiectare de Detaliu și Implementare

Scopul acestui capitol este de a documenta aplicația dezvoltată în așa fel încât dezvoltarea și întreținerea ulterioară să fie posibile. Se pot identifica funcțiile principale ale aplicației.

5.1. Arhitectura detaliată a aplicației

Aplicația folosește arhitectura layers(straturi) deoarece funcționalitățile principale sunt separate în straturi care îndeplinesc o funcționalitate unică. Cele 3 straturi sunt: aplicația de React(front-end), web API-ul scris în ASP.NET Core 6(back-end) și baza de date(PostgreSQL). Se poate spune că aplicația este dezvoltată după modelul arhitectural Client-Server și arhitectura sistemului respectă modelul Model-View-Controller(MVC).

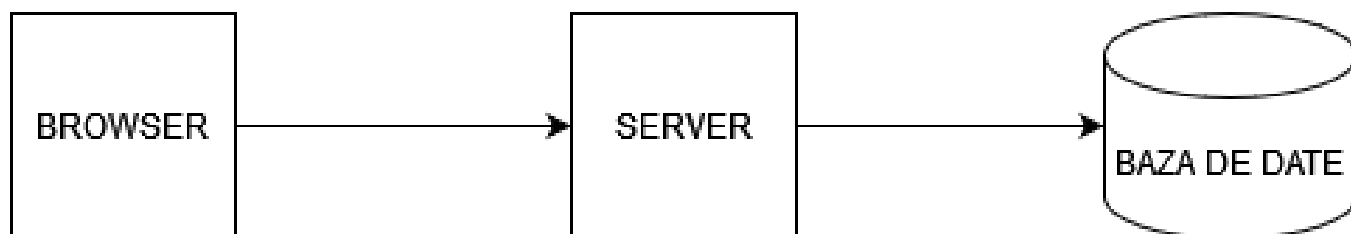


Figura 5.1: Schema bloc a sistemului

5.1.1. Baza de date

Baza de date utilizată de mine este una de tipul PostgreSQL. Baza de date a fost generată în urmă tehnicii de programare code-first din ASP.NET Core 6. Baza de date are 23 de tabele care salvează datele necesare efectuării operațiilor solicitate de utilizator. Există o tabelă generată de ASP.NET Core 6 care saveaza migrările responsabile de modificările aduse bazei de date și anume: `__EFMigrationsHistory`. Tabele pot fi încadrate în 2 tipuri: primare și secundare, primare fiind tabelele care păstrează date necesare funcționalității principale a aplicației, și cele secundare care păstrează date necesare tabelor principale. În cele ce urmează o să prezint schema bazei de date, dar deoarece nu încap pe o singură pagină, voi prezenta schema pe bucăți.

Tabele secundare

Tabelele secundare sunt tabele care păstrează date care pot fi ușor schimbate în fișa de disciplină, de exemplu: instituțiile, facultățile, departamentele și domeniile de studiu. Am ales abordarea creerii unor tabele separate pentru acestea pentru a reduce riscul de erori în completarea informațiilor unei fișe de disciplină. Dacă ofer utilizatorului doar opțiuni existente, acesta nu poate să adauge informații care ar putea crea neconcordanțe în cadrul aplicației.

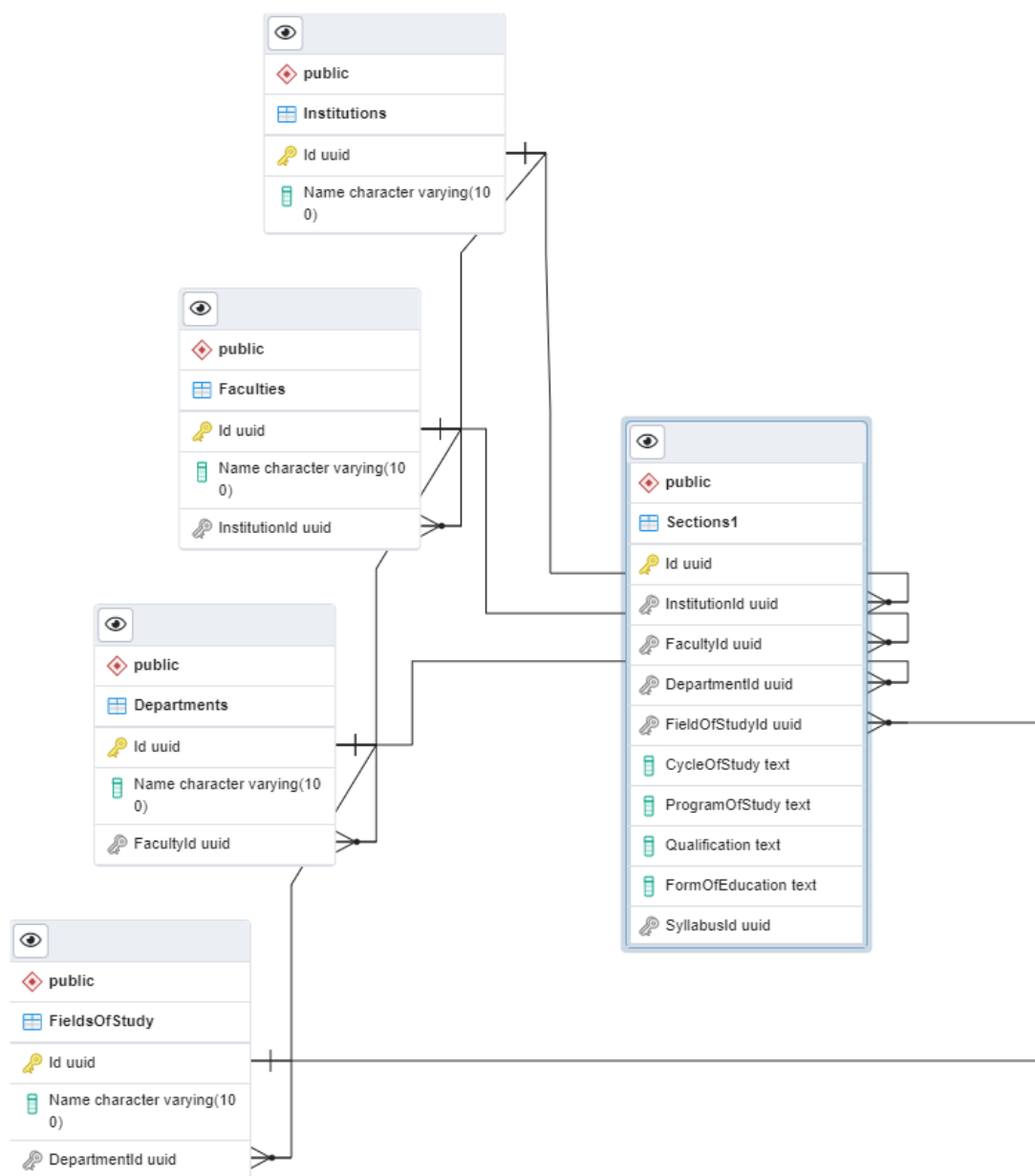


Figura 5.2: Schema bazei de date - tabele secundare

Tabelele Institutions, Faculties, Departments și FieldsOfStudy sunt construite în cascadă, ceea ce înseamnă că o inserție în tabela copil nu poate să existe fără o entitate părinte. Această construcție în cascadă facilitează operația de ștergere, deoarece atunci când se șterge, spre exemplu o instituție, tot arborele care pornește cu rădăcina în entitatea instituție care trebuie ștearsă, va fi de asemenea șters în mod automat. Relațiile definite între tabelele părinte și copil sunt relații de one-to-many, un părinte poate să aibă mai mulți copii, iar copiii la rândul lor pot avea din nou mai mulți copii. După cum am spus și în titlul secțiunii, aceste tabele sunt secundare, deoarece facilitează creerii unui obiect de tipul Section1. Elementele din tabela Sections1 au în componența lor chei străine determinate de id-urile tabelelor secundare și au și id-ul fișei de disciplină din care fac

parte.

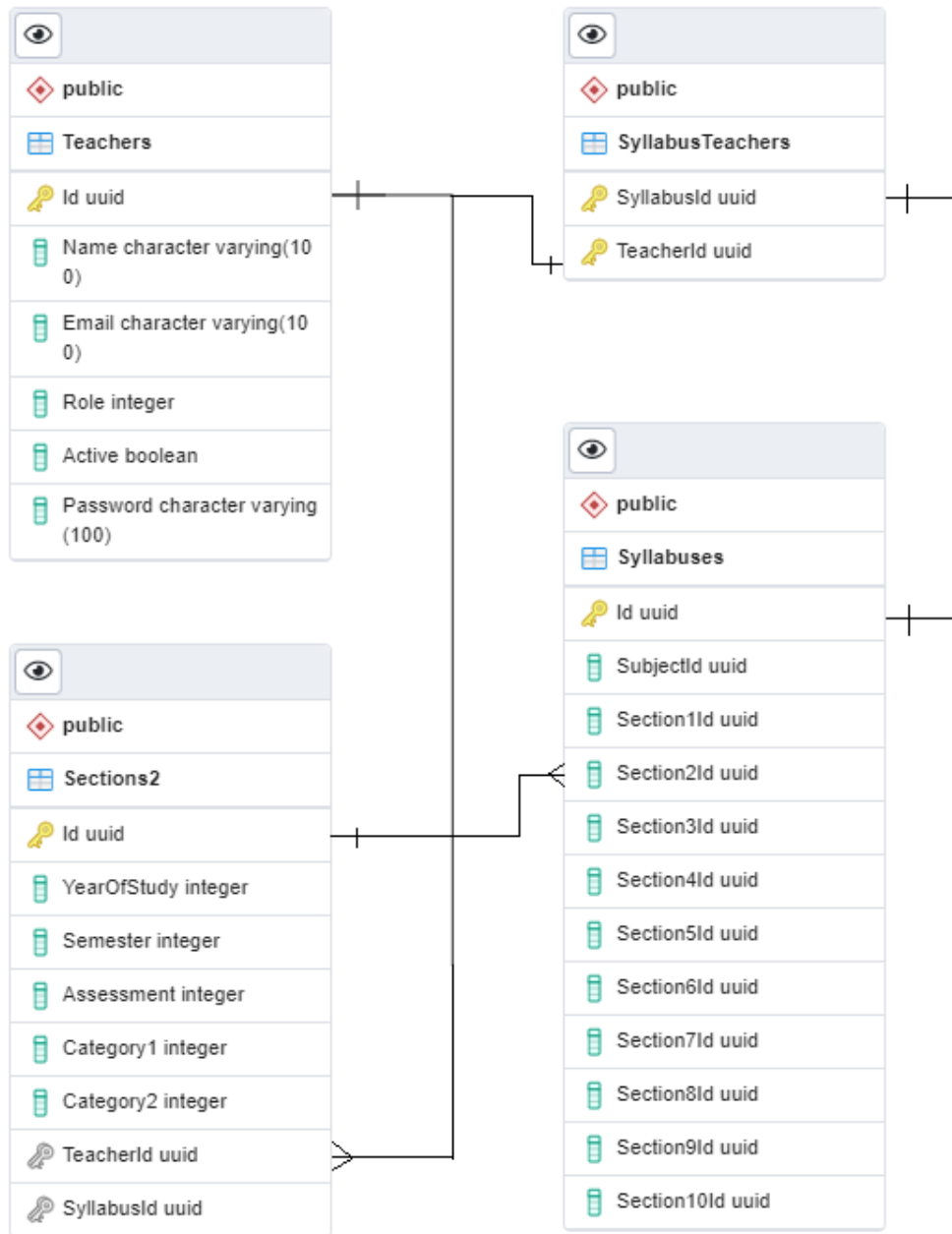


Figura 5.3: Schema bazei de date - secțiunea 2

Tabela Teachers împreună cu tabela SyllabusTeachers și tablea Syllabuses întrețin o relație many-to-many, deoarece în secțiunea 2 a unei fișe de disciplină trebuie aleși profesorii care vor ține laboratoare/seminare/laboratoare de proiect. Am ales să implementez relația many-to-many între profesori și fișa de disciplină din motive de funcționalitate pentru funcționalitatea de vizualizare a versiunilor anterioare. Tabela SyllabusTeacher are legături realizate prin chei străine cu tabela Teachers și tabela Syllabus. Cele 2 chei străine sunt setate să acționeze asemenea unei chei primare.

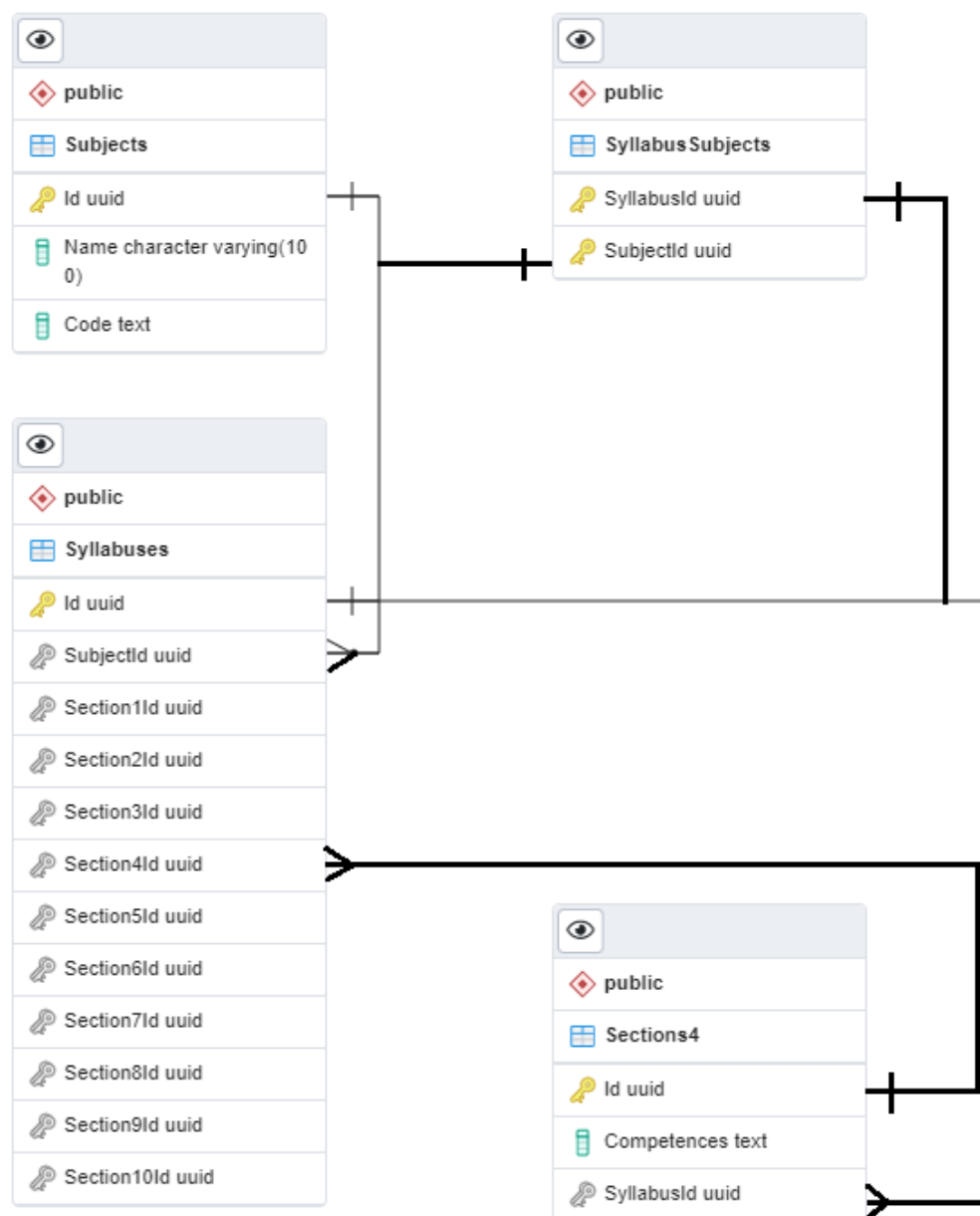


Figura 5.4: Schema bazei de date - secțiunea 4

Tabela Subjects împreună cu tabela SyllabusSubjects și tabela Syllabuses întrețin o relație many-to-many, deoarece în secțiunea 4 a unei fișe de disciplină trebuie alese materiile care ar trebui știute pentru a putea înțelege și acapara informații ușor despre materia curentă. Am ales să implementez relația many-to-many între materii și fișa de disciplină din motive de funcționalitate pentru funcționalitatea de vizualizare a versiunilor anterioare. Tabela SyllabusSubjects are legături realizate prin chei străine cu tabela Subjects și tabela Syllabus. Cele 2 chei străine sunt setate să acționeze asemenea unei chei primare. Deoarece o materie poate avea mai multe fișe de disciplină (una singură care este activă curent) există o relație de one-to-many între tabela Subjects și tabela Syllabuses.

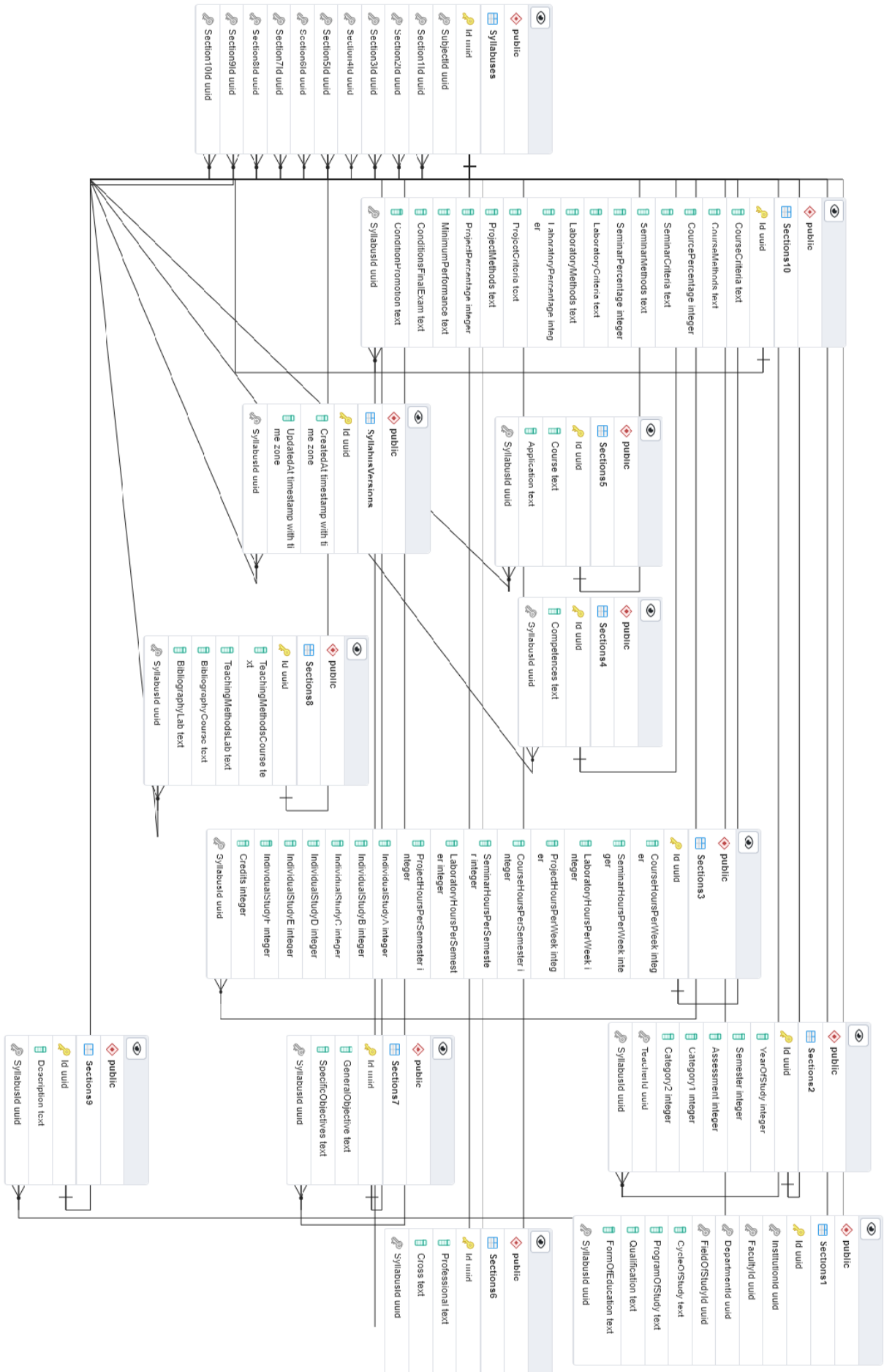


Figura 5.5: Schema bazei de date - fișa de disciplină și legăturile cu celelalte secțiuni

Tabele principale

În schemă de mai sus prezint tabela Syllabus și toate tabelele care stau la baza construcției acesteia. Am decis să folosesc această abordare datorită faptului că aplicația are și funcționalitate de vizualizare a istoricului fișelor de disciplină. Pentru a face o mare economie de spațiu, atunci când se actualizează o fișă de disciplină, serverul de back-end creează doar o nouă înregistrare în tabela Syllabuses și o nouă înregistrare pentru secțiunea/secțiunile actualizate, astfel se refolosesc secțiunile neactualizate și se face economie de spațiu. Această modularizare a secțiunilor îmbunătățește spațiul necesar de memorie și ajută atât la partea de dezvoltare cât și la partea de debugging.

Între tabela Syllabuses și tabelele destinate secțiunilor, există legături de tipul one-to-one, care se împlinesc prin intermediul cheilor străine(FK = foreign key) din fiecare tabela. Tabela SyllabusVersions face posibilă vizualizarea istoricului fișelor de diciplina.

În alte tipuri de baze de date există tabele temporale ¹ care fac posibilă gestionarea vesiunilor mai ușor, însă în PostreSql nu există astfel de tabele. Există o extensie, dar eu am hotărât să îmi codez versiunea mea de tabele temporale, astfel că atunci când se inserează o înregistrare nouă în tabela Syllabus, se va insera și o nouă înregistrare în tabela SyllabusVersion și câmpul CreatedAt va ține dată și ora curentă.

Fiecare tabela reprezintă o secțiune a fișei de disciplină cu informațiile necesare, însă unele secțiuni necesitau și tabele intermediare de stocare, precum secțiunea 2 și secțiunea 4 care au fost prezentate mai sus. Secțiunea 8 necesită de asemenea o tabela copil cu care se leagă o relație de one-to-many, astfel că o intrare în tabela Sections8 poate să aibă mai multe intrări în tabela Section8Elements. Tabela Section8Elements a fost necesară datorita faptului că datele pot fi citite mai ușor decât dacă s-ar fi salvat că un string și s-ar fi parsat mai departe. O intrare în această tabela are 4 proprietati: nume(care reprezintă numele cursului sau laboratorului), timpul alocat cursului sau laboratorului, observatii despre curs sau laborator și o valoare booleana care ne spune dacă intrarea respectivă este adresată subsecțiunii 8.1 Curs sau subsecțiunii 8.2 Aplicații a fișei de disciplină.

5.1.2. Back-end

Serverul de back-end este format din 8 pachete conectate între ele, dintre care 7 pachete sunt create de către mine și un pachet vine predefinit de la framework: Migrations. Pachetele aplicației de back-end sunt:

- Controllers: format din clasele care se ocupă de gestionarea cererilor clienților și fac disponibile funcționalitățile expunând spre exterior end-point-uri
- Services: format din clasele care ocupă de comunicarea cu baza de date. Pachetul este format la rândul lui din mai multe alte pachete separate pe funcționalități
- Entities: format din clasele care contruiesc baza de date. Sunt obiectele de baza ale aplicației
- Models: format din clasele cu care se formează stratul de abstractizare(DTO-uri) ¹

¹Temporal tables = tabele care ajută la versionarea informațiilor stocate în baza de date, reținând automat dată și ora la care a fost inserată informația, respectiv și dată și ora la care a fost modificată

- Profile: format din clasele care servesc librăria AutoMapper [12]
- Utils: format din clase care îndeplinesc diferite funcționalități necesare în aplicație
- DbContext: are o singură clasă care ajută la crearea bazei de date
- Migrations: format din fișiere care conțin modificările aduse bazei de date

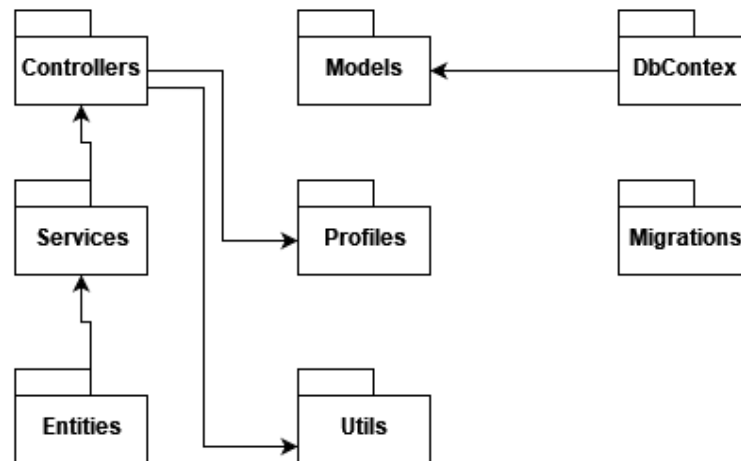


Figura 5.6: Diagrama de pachete a WebAPI-ului

Entities

În cadrul pachetului Entities, luând în considerare scopul aplicației, se poate spune că există 2 tipuri de entități și anume:

- Entități cu scop de ajutor
 - Constants: sunt salvate obiecte de tip Enum necesare în aplicație
 - Institution: date reprezentative unei instituții
 - Faculty: date reprezentative unei facultăți
 - Department: date reprezentative unui departament
 - FieldOfStudy: date reprezentative unei domenii de studiu
 - Section8Element: informații ajutatoare pentru secțiunea 8 a fișei de disciplină
 - SyllabusSubject: date necesare creerii unui tabel pentru legătură many-to-many
 - SyllabusTeacher: date necesare creerii unui tabel pentru legătură many-to-many
- Entități principale
 - Teacher: date reprezentative unui profesor
 - Subject: date reprezentative unei materii
 - Audit: date reprezentative unei înregistrări de audit
 - Syllabus: entitate care ține cheile primare ale entităților necesare pentru a ține informații despre secțiuni

¹DTO = Data Transfer Object, obiect care transportă informații între operațiile executate de server

- Section1: date reprezentative secțiunii 1 a fișei de disciplină
- Section2: date reprezentative secțiunii 2 a fișei de disciplină
- Section3: date reprezentative secțiunii 3 a fișei de disciplină
- Section4: date reprezentative secțiunii 4 a fișei de disciplină
- Section5: date reprezentative secțiunii 5 a fișei de disciplină
- Section6: date reprezentative secțiunii 6 a fișei de disciplină
- Section7: date reprezentative secțiunii 7 a fișei de disciplină
- Section8: date reprezentative secțiunii 8 a fișei de disciplină
- Section9: date reprezentative secțiunii 9 a fișei de disciplină
- Section10: date reprezentative secțiunii 10 a fișei de disciplină

Mergând pe abordarea code-first a framework-ului Entity Framework [13], fiecare entitate reprezintă o tabelă în baza de date. Deoarece în noile versiuni de C# există posibilitatea să îți declari field-urile clasei drept proprietăți, am mers pe această abordare, ceea ce ușurează utilizarea claselor.

Models

Pachetul models este format din alte 6 pachete mai bine delimitate. Modelele reprezintă tipurile de date utilizate în comunicarea dintre WebAPI și utilizator. Aceste pachete sunt necesare pentru a adăuga un strat de abstractizare între datele salvate în baza de date și datele trimise către utilizator. Nu este o abordare bună dacă se trimit datele direct din baza de date către utilizator deoarece s-ar putea să conțină și alte field-uri care sunt confidențiale, sau care ar putea încetini viteza request-ului. DTO-urile(data transfer objects) creează un strat de protecție, deoarece operațiile cerute de utilizator se efectuează mai întâi pe elemente abstracte apoi dacă au trecut toate verificările și nu există probleme, se mapează dto-urile la entități ale bazei de date, care mai apoi se salvează.

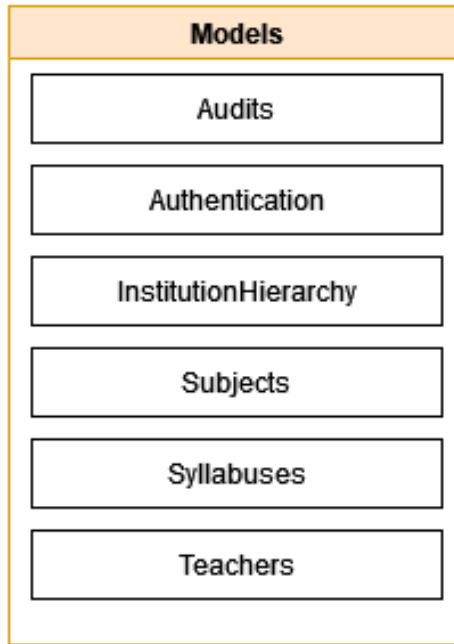


Figura 5.7: Pachet Models

Am ales să împart acest pachet în pachete mai mici pentru a putea gestiona mai bine clasele care formează DTO-urile. Fiecare pachet are clase care sunt folosite în cadrul unui controller, sau a mai multor controllere, pentru a trimite doar datele necesare clientului sau pentru a primi date de la client care mai apoi să fie translatate în entități folosite de baza de date.

DbContext

În pachetul DbContext se află o singură clasă care este responsabilă de gestionarea relațiilor cu tabela din baza de date. Clasa EntityContext moștenește clasa DbContext (clasa dezvoltată de programatorii Microsoft) și asigură relațiile cu baza de date PostgreSQL. Această clasă oferă posibilitatea programatorului să-și definească singur relațiile între tabele, dacă EntityFramework nu reușește să satisfacă cerințele de funcționalitate.

Listing 1 Creare de cheie primară din două chei străine pentru tabele folosite în relații de many-to-many

```

1      protected override void OnModelCreating(ModelBuilder modelBuilder)
2      {
3          modelBuilder.Entity<SyllabusTeacher>()
4              .HasKey(st => new { st.TeacherId, st.SyllabusId });
5          modelBuilder.Entity<SyllabusSubject>()
6              .HasKey(ss => new { ss.SubjectId, ss.SyllabusId });
7      }
  
```

După cum se poate observa în exemplul de mai sus, pentru tabelul SyllabusTeacher, folosind o funcție predefinită în clasa DbContext: *OnModelCreating* și folosind o tehnică

de programare funcțională se setează cheia primară(PK = primary key) ca fiind o uniune dintre 2 chei străine(FK = foreign key): *TeacherId*, *SyllabusId*. Acest lucru este posibil deoarece cheile străine din tabelul curent sunt chei primare în tabelele lor, deci o uniune de 2 obiecte unice rezultă în alt obiect unic. Exact același lucru se întâmplă și pentru tabelul *SyllabusSubject* folosind comandă de la linia 5.

Profiles

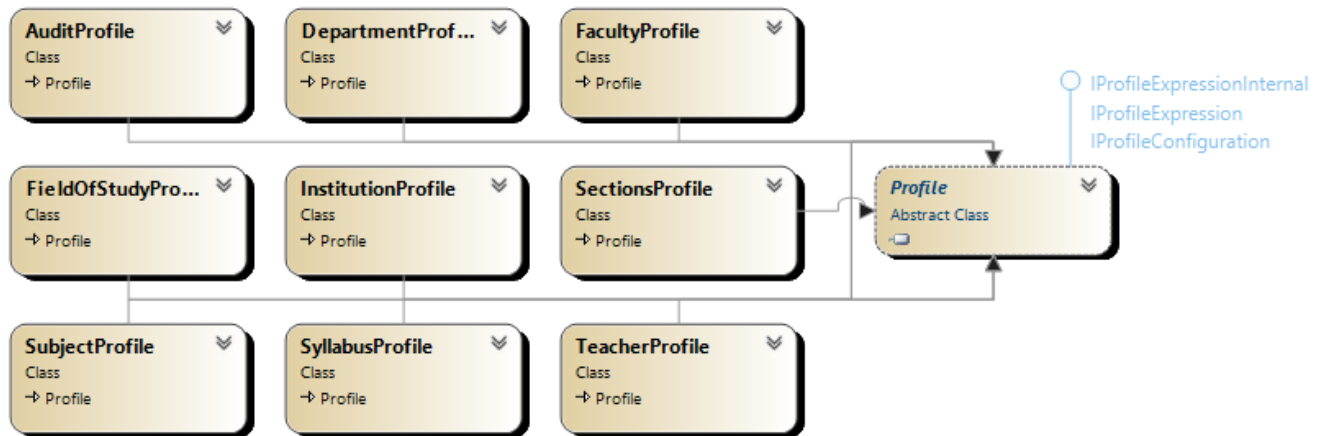


Figura 5.8: Diagrama de clase din pachetul Profiles

Profile-urile sunt clase special făcute care servesc librăriei AutoMapper pentru a mapa DTO-urile la entități. Toate clasele definite în pachetul Profiles mestenesc clasa abstractă *Profile* și au rolul de a ajuta programatorul să reducă riscul de eroare când își transformă entitatea în DTO sau DTO-ul în entitate. O altă utilitate pentru clasele din pachetul Profiles este faptul că poate transfera datele unui obiect în altul folosind doar funcția *Map*. Acesta funcție este foarte utilă datorită faptului că reduce riscului de a greși.

Listing 2 Exemplu de mapare personalizată pentru cazul meu de utilizare

```

1 CreateMap<Section2CreateDto, Entities.Section2>()
2     .ForMember(source => source.Teachers,
3               opt => opt.DoNotValidate());
  
```

Librăria AutoMapper permite utilizatorului să controleze procesul de mapare între două entități. Luând exemplul de mai sus, funcția se poate traduce ca: mapează-mi proprietățile unui obiect de tipul *Section2CreateDto* la proprietățile unei entități *Section2*, dar nu lua în calcul field-ul *Section2CreateDto.Teachers*. Am ales această abordare, deoarece cu toate că eu primesc din partea utilizatorului(aplicația de React) lista de profesori care au fost atribuiți pentru secțiunea 2, în partea de back-end acea lista trebuie transformată în entități de tipul *SyllabusTeacher*.

Controllers

În cadrul acestui pachet sunt definite controllerele care se ocupă de expunerea spre exterior a endpoint-urilor. Cererile utilizatorilor sunt mapate la endpoint-urile definite

în controllere ceea ce permite serverului să știe cu ce informații să ajute utilizatorul. Controller-ele moștenesc clasa ControllerBase definită de către framework-ul ASP.NET Core 6. Înainte de a se înainta o comandă la server, controller-ul verifică dacă user-ul care a trimis cererea îndeplinește criteriile de autorizare pentru a folosi endpoint-ul pe care se mapează cererea. Pentru verificările de autorizare se utilizează jetonul (JWT Token) care este prezent în header-ul de autorizare asociat cererii HTTP, dar voi explica mai multe în secțiunea dedicată securității.

Singurul controller care nu necesită autorizare este: AuthenticationController, deoarece acest controller servește creerii de jetoane JWT și autentificării. Majoritatea controllerelor expun endpoint-uri care îndeplinesc funcționalități CRUD (Create Read Update Delete), însă există și câteva controllere care îndeplinesc mai multe funcționalități. SyllabusController pe lângă funcționalitățile de create, read, update, delete are și un endpoint care permite utilizatorului să descarce o fișă de disciplină în funcție de materie sau id-ul fișei de disciplină.

AuditController este controller-ul care se ocupă de citirea audit-ului de securitate (tabel al bazei de date care salvează înregistrări de tipul utilizator, acțiune, entitate pe care a fost executată acțiunea) din baza de date și trimiterea acestuia la solicitare prin intermediul protocolului de comunicare HTTPS la client (browser-ul de pe care a fost trimisă cererea HTTP). Pentru accesarea acestui end-point utilizatorul trebuie să îndeplinească politică de autorizare "MustBeAdmin". 3

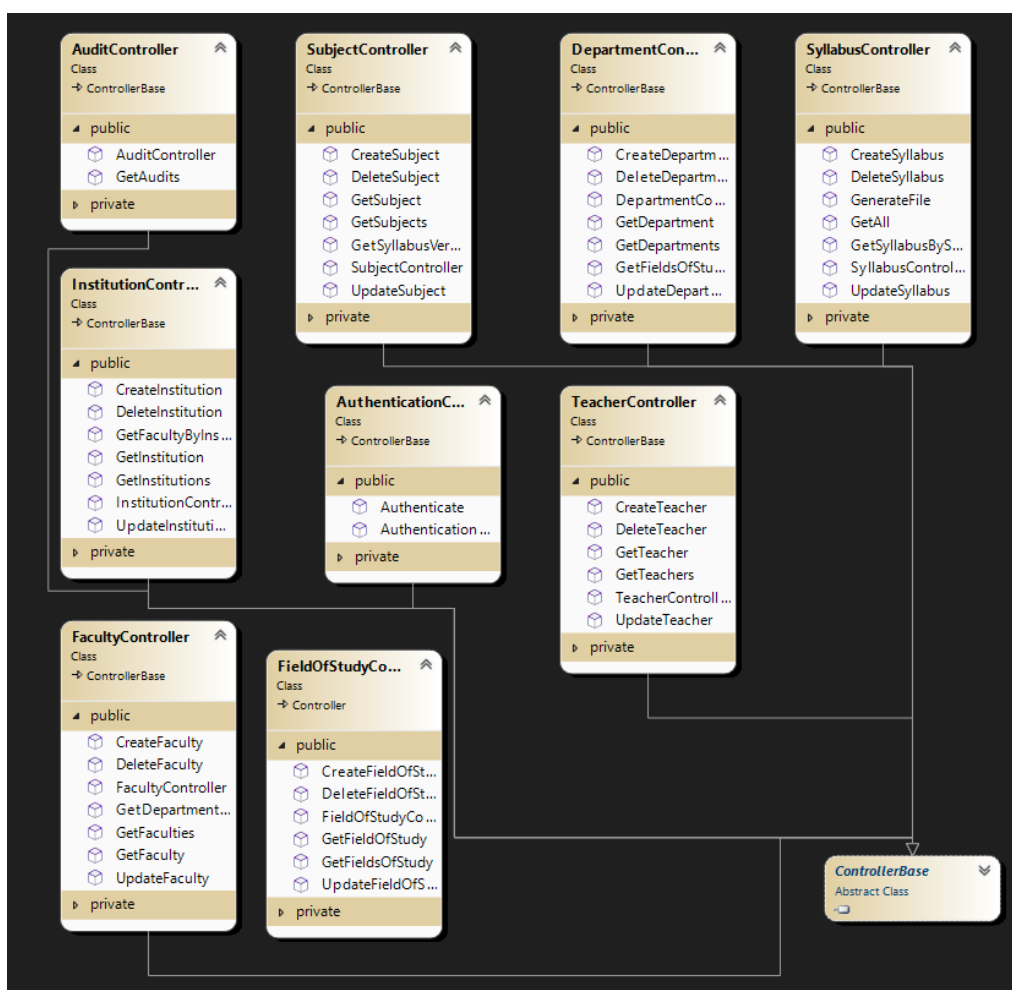


Figura 5.9: Diagrama de clase a pachetului Controllers

Listing 3 Politica de autorizare *MustBeAdmin*

```

1  options.AddPolicy("MustBeAdmin", policy =>
2  {
3      policy.RequireAuthenticatedUser();
4      policy.RequireClaim("role", "Admin");
5  });

```

AuthenticationController este controller-ul care se ocupă de autentificare și generare de jetoane JWT. Deoarece parolă utilizatorului este salvată sub formă de hash în baza de date atunci când se primesc credentialele, se aplică algoritmul de hashing SHA384 pe parolă [14] și se verifică dacă parolă existența în baza de date este aceeași cu parolă introdusă de utilizator. Dacă parolele coincid începe generarea jetonului JWT. JWT Tokenul este securizat cu ajutorul algoritmului HmacSha256. În partea de body a jetonului sunt salvate, pe lângă datele de baza ale unui JWT Token: site-ul care a emis jetonu, site-ul pentru care s-a emis jetonul, data și ora la care s-a emis jetonul, data și ora la care expiră jentonul, rolul utilizatorului care s-a autentificat, id-ul utilizatorului care s-a autentificat și se adaugă 8 ore la data de expirare a jetonului.

InstitutionController este controller-ul responsabil de operațiile adresate entităților de tipul instituție, operații de genul create, read, update, delete cât și operația de adăugare a unei noi facultăți la instituția de învățământ superior. Pentru a folosi operațiile de tipul create, update, delete utilizatorul trebuie să fie autorizat și să îndeplinească politica de autorizare "MustBeAtLeastEditor". 4 Toate operațiile executate în cadrul acestui controller sunt salvate în tabela de audit.

FacultyController este controller-ul responsabil de operațiile adresate entităților de tipul facultate, operații de genul create, read, update, delete cât și operația de adăugare a unui nou departament la facultate. Pentru a folosi operațiile de tipul create, update, delete utilizatorul trebuie să fie autorizat și să îndeplinească politica de autorizare "MustBeAtLeastEditor". 4 Toate operațiile executate în cadrul acestui controller sunt salvate în tabela de audit.

DepartmentController este controller-ul responsabil de operațiile adresate entităților de tipul departament, operații de genul create, read, update, delete cât și operația de adăugare a unui nou domeniu de studii la departament. Pentru a folosi operațiile de tipul create, update, delete utilizatorul trebuie să fie autorizat și să îndeplinească politica de autorizare "MustBeAtLeastEditor". 4 Toate operațiile executate în cadrul acestui controller sunt salvate în tabela de audit.

FieldOfStudyController este controller-ul responsabil de operațiile adresate entităților de tipul domeniu de studiu, operații de genul create, read, update, delete. Pentru a folosi operațiile de tipul create, update, delete utilizatorul trebuie să fie autorizat și să îndeplinească politica de autorizare "MustBeAtLeastEditor". 4 Toate operațiile executate în cadrul acestui controller sunt salvate în tabela de audit.

Listing 4 Politica de autorizare *MustBeAtLeastEditor*

```

1      options.AddPolicy("MustBeAtLeastEditor", policy =>
2      {
3          policy.RequireAuthenticatedUser();
4          policy.RequireClaim("role", "Editor", "Admin");
5      });

```

SubjectController este controller-ul responsabil de operațiile adresate entităților de tipul disciplină, operații de genul create, read, update, delete. Pe lângă operațiile de baza CRUD acest controller mai expune un endpoint necesar operației de vizualizare a versiunilor anterioare ale fișei de disciplină a disciplinei în cauza. Pentru a folosi operațiile de tipul create, update, delete utilizatorul trebuie să fie autorizat și să îndeplinească politica de autorizare "MustBeAtLeastEditor" 4. Pentru a utiliza funcționalitate de vizualizare a versiunilor anterioare ale fișei de disciplină utilizatorul nu are nevoie de autorizări speciale înafara de autorizarea de baza: utilizator autentificat. Toate operațiile executate în cadrul acestui controller sunt salvate în tabela de audit.

TeacherController este controller-ul care se ocupă de cererile care au ca subiect utilizatorii. Acest controller expune end-point-uri pentru operații de tipul create, update,

read și delete. Pentru a utiliza funcționalitățile expuse de TeacherController utilizatorul trebuie să fie autentificat în primul rând. Operația de vizualizare a utilizatorilor este disponibilă tuturor tipurilor de utilizator, însă operațiile de tipul create, update, delete sunt monopolizate doar de utilizatorii care îndeplinesc politică de autorizare "MustBeAdmin"

3. Am hotărât că operația de delete să fie doar un soft-delete ¹ pentru a putea gestiona mai bine personalul și istoricul personalului. Toate operațiile executate în acest controller sunt salvate în tabela de audit.

SyllabusController este controller-ul cel mai important din aplicație. Toate controller-urile prezentate anterior au fost contruite pentru a putea ușura crearea unei fișe de disciplină și pentru a avea date cât mai modelabile. Controller-ul expune end-point-uri pentru operațiile de create, read, update și delete, cât și un endpoint pentru generarea de pdf-uri pentru fișele de disciplină. Pentru a putea utiliza funcționalitățile furnizate de acest controller, utilizatorul trebuie să îndeplinească politica de autorizare "MustBeAtLeastEditor" ⁴. Din nou, și la acest controller, pentru acest tip de entitate, operația de delete este un soft-delete ¹.

Operațiile efectuate de SyllabusController au un grad de complexitate mai mare deoarece precum am spus și în secțiunea dedicată entităților, entitatea Syllabus(fișa de disciplină) depinde de alte 11 entități, fiecare având proprietăți diferite și relații diferite.

Services

Pachetul Services are în componența sa alte 5 pachete, fiecare pachet delimitând subiectul serviciului. În aceste pachete se găsesc interfețele responsabile pentru operațiile pe baze de date și implementarea lor propriu-zisă.

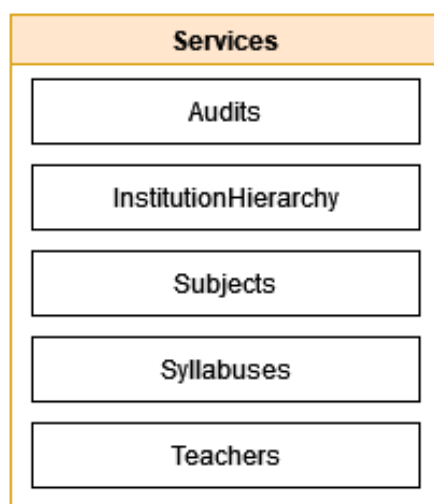


Figura 5.10: Diagrama de clase a pachetului Services

Cele 5 pachete sunt:

- Audits: pachet care încorporează repository-ul pentru tabela de audit

¹soft-delete = operația prin care o resursă nu este eliminată din baza de date, în schimb primește un status de item inactiv

- **InstitutionHierarchy**: pachet care delimitează operațiile pentru tabelele Institutions, Faculties, Departments și FieldsOfStudy
- **Subjects**: pachet care delimitează operațiile pentru tabela Subjects
- **Syllabuses**: cel mai mare pachet, se regăsesc toate repository-urile pentru tabelele folosite pentru fișele de disciplină
- **Teachers**: pachet care delimitează operațiile pentru tabela de Teachers

Fiecare clasă implementează o interfață care definește metodele care trebuie implementate. Pentru a maximiza performanța, accesul la baza de date se face asincron prin fire de execuție separate (thread-uri), astfel serverul poate să răspundă mai repede cererilor clienților fără a provoca costuri suplimentare, sau a necesita echipamente suplimentare.

Utils

În pachetul Utils se află clasa responsabilă de operațiile de convertire a string-urilor. Datorită neconcordanțelor de tipuri între back-end și front-end am decis că unele elemente salvate drept string-uri în baza de date să fie primite sau trimise în formă de listă de string-uri. Pentru acest lucru am ales un separator(simbol aleatoriu din UTF8) care să delimiteze string-urile. Cu ajutorul aceluși separator am scris două funcții personalizate care să îmi transforme string-urile în liste de string-uri și listele de string-uri în string-uri. Acest lucru se întâmplă pentru comprimarea mai multor date pentru mai multe secțiuni, dar voi vorbi mai multe despre acestea în secțiunea legată de funcționalități.

```
private static string SEPARATOR = "🌀";
```

Figura 5.11: Separator folosit pentru convertirea string-urilor

Separatorul adaugă un strat de protecție al datelor. Fără a ști separatorul, nu se poate face o decodare exactă.

¹UTF-8 = format standard de codare a literelor și caracterelor

5.1.3. Front-end

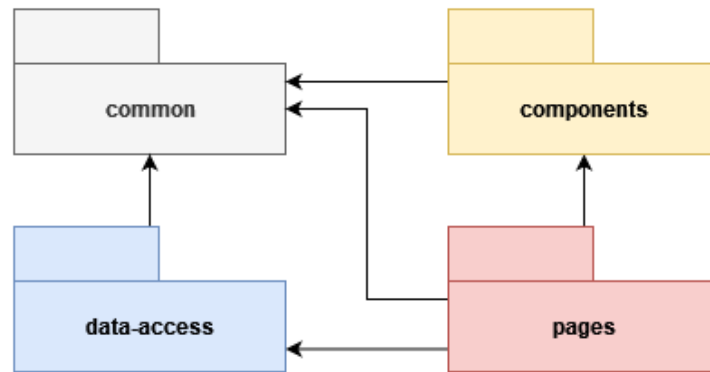
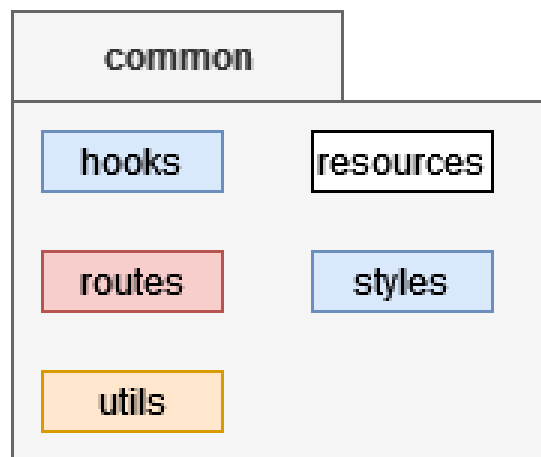


Figura 5.12: Diagrama de pachete pentru aplicația React

Arhitectura propusă de mine este formată din 4 straturi care comunica între ele:

- **common**: strat care păstrează în general constantele și funcțiile care sunt des folosite în cadrul aplicației
- **components**: strat care definește componentele reutilizabile
- **data-access**: strat care definește logica transferului și păstrării datelor
- **pages**: strat care definește paginile și componentele personalizate pentru pagina respectivă

CommonFigura 5.13: Diagrama de pachete pentru pachetul **common**

Pachetul **common** păstrează alte 4 pachete necesare pentru buna funcționare a aplicației:

- **hooks**: în acest pachet sunt salvate hook-uri personalizate, programate de mine
- **resources**: în acest pachet sunt resurse de tipul: poze, text, iconițe etc.
- **routes**: în acest pachet se regăsesc fișiere în care sunt definite rutele interne și externe
- **style**: în acest pachet sunt definite clase de stilizare
- **utils**: în acest pachet sunt salvate diverse funcții care se utilizează în mai multe părți

Pachetul **hooks** este un pachet care aduce funcționalitate suplimentară pentru ușurarea dezvoltării aplicației. Un hook este o componentă care poate utiliza stări și alte funcționalități React specifice claselor, dar fără a defini o clasă. În exemplul expus mai jos, am creat un hook care să mă ajute la gestionarea unui jeton JWT. Pentru a folosi acest hook, atunci când se creează hook-ul trebuie să primească ca parametru un JWT Token, apoi se pot pur și simplu apela funcțiile definite în interiorul hook-ului.

Listing 5 Exemplu de hook personalizat

```

1 export const useToken = (token: string) => {
2     var tokenData: TokenType = token ? jwt_decode(token) : TOKEN_DEFAULT;
3     const isValid = (): boolean => {
4         if (Date.now() >= Number(tokenData.exp) * 1000) return false;
5         return true;
6     };
7     /* alte functionalitati */
8     return {
9         isValid,
10        getRole,
11        getUserId,
12    };
13 };

```

Pachetul **routes** poate fi numit unul dintre cele mai importante pachete ale aplicației, deoarece păstrează în mai multe fișiere *.ts*, separate de domeniul de utilizare, rutele interne și externe utilizate de aplicația React. Rutele interne sunt rutele folosite în cadrul aplicației din browser, de exemplu */dashboard* este ruta care duce spre dashboard. Rutele externe sunt rutele spre end-point-urile serverului, folosind aceste rute aplicația de front-end poate comunica cu aplicația de back-end. Toate aceste rute sunt declarate în pachete separate, deoarece pot exista modificări în aplicație, și este mai ușor să modifice o valoare decât să modifice 100 de fișiere.

Deoarece am decis să renunț la fișierele CSS pentru a utiliza principiul de scriere CSS în JS 3.4, pentru a nu repeta același cod de nenumărate ori, am decis să fac un pachet responsabil pentru păstrarea codului general de stilizare, acest pachet fiind pachetul **style**. În acest pachet am scris atât tema de stilizare a componentelor librăriei Material-UI, cât și dicționare cu valori predefinite pentru culori, fonturi și clase de stilizare.

Pachetul **utils** păstrează fișiere în care am scris funcții personalizate care sunt folosite în mai multe componente, sau în alte funcții, pe scurt, sunt salvate funcții generale asemănătoare cu funcțiile salvate în pachetul Utils din back-end. În acest pachet am

scris funcții care ajută la validări ale JWT Token-ului, funcții responsabile cu parsarea și afișarea datelor calendaristice într-un format citibil și interpretabil de creierul uman, funcții de validare a răspunsurilor primite din partea serverului, pentru a putea afișa notificări corespunzătoare.

Pentru o mai bună gestionare a importurilor, fiecare pachet are un fișier **index.ts** care exportă componente din fișierele de la același nivel cu el. Acest lucru facilitează scrierea mai ordonată și aranjată a importurilor, lucru care facilitează și vitezei de încărcare a paginilor web(importuri mai scurte => text mai puțin => mai puțin cod de interpretat => browser mai fericit). În figura de mai jos se poate vedea cum aproape în fiecare pachet există un fișier **index.ts**. Acest fișier exportă(face vizibil pentru celelalte componente ale aplicației) o cale mai ușor de scris pentru a ajunge la un anumit fișier. De exemplu, pentru a importa funcționalități din fișierul *useToken.ts*, luând în calcul faptul că nu există fișierele *index.ts*, calea către acel fișier ar fi: `"/src/common/hooks/useToken"`, însă datorită fișierelor *index.ts* care expun restul fișierelor de la nivelul lor, calea către fișierul dat ca exemplu anterior devine `"/src/common"`.

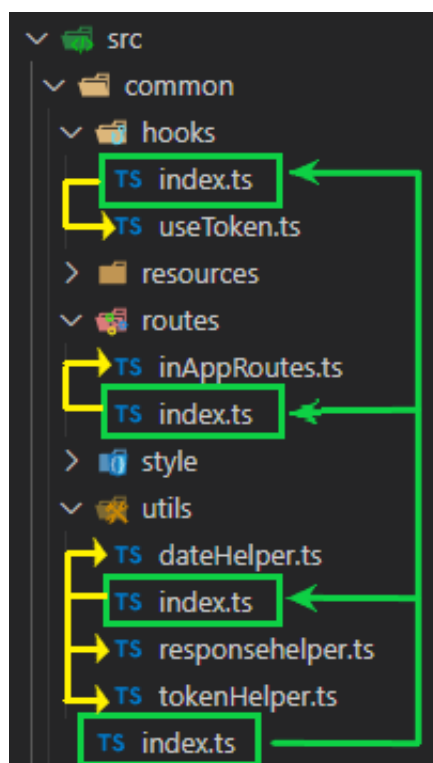


Figura 5.14: Principiul de "best-practice" pentru export-uri din pachete

Components

În pachetul **components** se află componentele generale care pot fi utilizate de diferite pagini și componentele utilizate pentru structurarea paginii. Componentele prezente în acest pachet sunt:

- Router: una dintre cele mai importante componente ale aplicației web. Această componenta face posibilă rutarea dinamică spre diferite pagini
- Layout: reprezintă componenta care dă structura în browser, a aplicației

- **SideMenu**: componentă ajutătoare a componentei **Layout**, reprezintă meniul din partea stângă a aplicației
- **ContentInput**: input personalizat
- **ContentOptionsInput**: input personalizat cu mai multe opțiuni

Componenta **Router** poate fi văzută drept cea mai importantă componentă a aplicației de React. Acesta componentă permite navigarea dinamică între pagini, navigare care poate fi intenționată(apăsarea unui buton de către user) sau neintenționată(utilizatorul nu este autorizat să facă o operație, deci este trimis la pagină de Dashboard). Rutele sunt împărțite în două feluri: rute autorizate și rute neautorizate. Rutele autorizate permit utilizatorului să navigheze în interiorul aplicației, iar rutele neautorizate permit utilizatorului doar să vadă pagină de login.

Componenta **Layout** reprezintă scheletul paginii și delimitează spațiul alocat celorlalte componente. Scheletul implementat de mine 7.18 este format din 2 părți statice(rămân nemodificate pe parcursul utilizării aplicației) și o parte dinamică(se modifică în funcție de conținutul paginii curente).

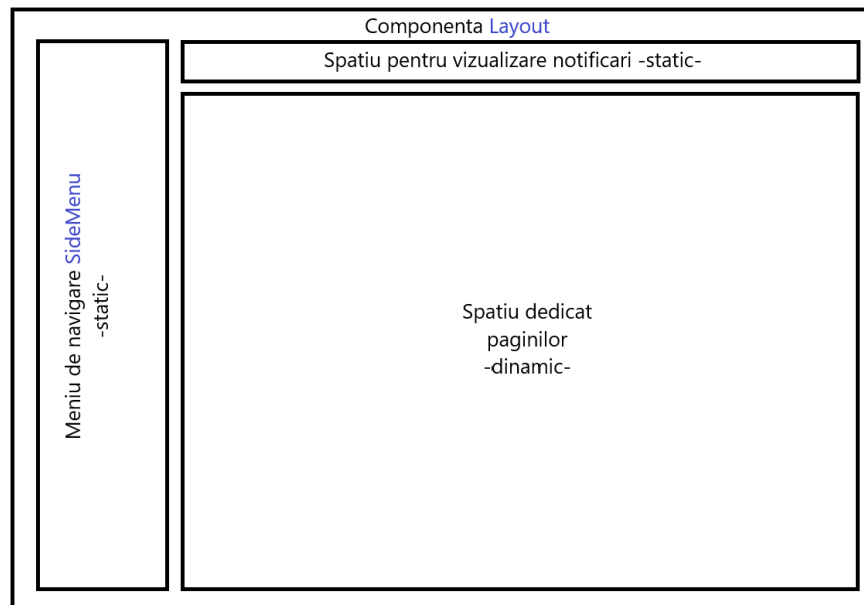


Figura 5.15: Structura aplicației web

Componenta **SideMenu** este meniul de navigare în aplicație, din acest meniu utilizatorul poate să navigheze pe pagina dorită de el.

Componenta **ContentInput** este un input care are mai multe functionalitati. Am decis să îl implementez deoarece în secțiunile 4, 5, 6, 7 ale unei fișe de disciplină, utilizatorul poate să introducă mai multe paragrafe într-un singur field. Cu această funcționalitate în minte am dezvoltat un input care să permită structurarea și vizualizarea paragrafelor în timp real. Paragrafele se salvează într-un vector de string-uri. Paragrafele pot fi adăugate sau șterse de către utilizator. După același principiu am creat și componentă **ContentOptionsInput**, dar această componentă este utilizată pentru secțiunea 8 unde paragrafele conțin 3 tipuri de date: denumire, număr de ore și observații. Astfel, vectorul nu este unu-l de stringuri ci este un vector de obiecte JSON.

Data-access

Acest pachet este responsabil de gestiunea datelor în aplicație, atât date folosite în cadrul aplicației cât și comunicarea cu alte servicii. Pachetul data-access este împărțit în alte 3 pachete:

- service: pachet responsabil de asigurarea comunicării cu serverul de back-end
- store: pachet responsabil de păstrarea datelor în interiorul aplicație
- types: pachet responsabil de definirea tipurilor de date utilizate în aplicație.

Pachetul **service** este format din fișiere delimitate de scopul funcțiilor care se regăsesc în ele. În acest pachet se creează instanța de Axios de care se folosesc restul funcțiilor pentru a realiza comunicarea cu serverul de back-end. Atunci când se creează instanța de Axios, aplicația verifică dacă în Cookies-urile browser-ului există jetonu JWT. Dacă există și este încă valid(nu au trecut cele 8 ore de când a fost emis) atunci se adaugă header-ul de autorizare în requesturile HTTP. Pe lângă instanța de Axios, tot în acest pachet sunt declarate funcțiile folosite pentru comunicarea cu WebAPI.

Listing 6 Adaugarea header-ului de autorizare la instanța de Axios

```
1  if (token) {  
2      return axios.create({  
3          ...defaultConfig,  
4          headers: { ...defaultHeaders, Authorization: `Bearer ${token}` },  
5      });  
6  }
```

Pachetul **store** reprezintă locul unde este definit stratul de memorie intera folosit de către librăria Redux Toolkit. Denumirea de *store* este o convenție generală propusă de către autorii librăriei și adoptată de către programatori. Store-ul meu este format din 4 felii(slice-uri) care păstrează date despre: autentificare, secțiuni și utilizator. Deoarece folosesc Typescript în loc de Javascript a trebuit să definesc și 2 hook-uri care să împacheteze hook-urile de baza: useSelector și useDispatch. Cel mai important dintre slice-uri este slice-ul dedicat secțiunilor fișei de disciplină. Acesta are în componentă câte un obiect cu tip unic specific pentru fiecare secțiune din fișa de disciplină. Datele din acest slice de Redux vor fi mai apoi utilizate pentru crearea și editarea fișelor de disciplină, dar voi spune mai multe legat de acest lucru în partea de funcționalități.

Listing 7 Exemplu de interfata personalizata pentru sectiunea 4

```

1 export interface Section4Type {
2   id: string; selectedSubjects: SelectType[];
3   competences: string[];
4 }
5 export interface Section4CreateDto
6   extends Omit<Section4Type, "id" | "selectedSubjects"> {
7   subjects: string[];
8 }
9 export const mapSection4TypeToSection4CreateDto = (
10   section4: Section4Type
11 ): Section4CreateDto => {
12   const { selectedSubjects, competences } = section4;
13
14   return {
15     subjects: selectedSubjects.map((value) => value.value),
16     competences,
17   };
18 };

```

Pachetul **types** indică faptul că aplicația folosește limbajul de programare Typescript în loc de Javascript în acest pachet am definit toate tipurile complexe utilizate în cadrul aplicației web. Pe lângă tipuri, am declarat și unele funcții care ajută la maparea dintre tipuri. Folosindu-mă de proprietățile interfețelor din Typescript a fost suficient să declar un singur tip per secțiune pe care apoi să-l editez în funcție de nevoie. În exemplul de mai sus, *Section4Type* reprezintă tipul de date folosit de aplicația web, *Section4CreateDto* reprezintă tipul de date folosit în cererea adresată serverului de back-end(tipul de date pe care serverul îl așteaptă), iar funcția *mapSection4TypeToSection4CreateDto* arată procesul de schimbare între tipul de date folosit intern de aplicația web și tipul de date pe care serverul de back-end îl așteaptă. Tipurile pe care le-am definit în acest pachet sunt atât tipuri de date utilizate pentru comunicarea cu alte servicii, cât și tipuri de date utilizate intern de către componentele React.

<pre> 1 interface SelectType { 2 label: string; 3 value: string; 4 } </pre>	<pre> 1 interface ContentOptionsInputType { 2 name: string; 3 duration: number; 4 note: string; 5 } </pre>
---------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------

Figura 5.16: Cele doua tipuri de date utilizate cel mai des de componentele aplicatiei de front-end

Tipurile de date definite mai sus sunt cele mai utilizate tipuri de date în aplicația React. Toate elementele de tipul ComboBox sau Autocomplete sau CustomOptionsInput folosesc unul dintre aceste 2 tipuri pentru generalizarea și ușurarea dezvoltării aplicației. Tipul *SelectType* este folosit de către ComboBox-uri și Autocomplete-uri, deoarece aceste 2 tipuri de componente lucrează cu 2 valori: valoarea afișată și codul unic al acelei valori

afișate, deci ComboBox-urile afișează proprietatea: **label**, dar când se trimit valorile spre server, se ia doar proprietatea de **value**. Această informație va fi utilă în subsecțiunea Funcționalități unde voi prezenta mapările necesare pentru a ajunge de la un obiect tip entitate baza de date la un obiect afisabil pe interfața grafică.

Pages

Pachetul pages este rezervat pentru păstrarea componentelor React care definesc paginile web și funcționalitățile lor. Tot în acest pachet sunt definite și componentele personalizate pentru operațiile efectuate pe pagină definită de folderul în care se află.

5.2. Funcționalități

Lucrarea mea de licență se bazează pe un singur principiu și anume: ușurarea gestionării fișelor de disciplină din cadrul Universității Tehnice din Cluj Napoca. Aplicația web, dezvoltată cu ajutorul librăriei React are un design simplu și ușor de înțeles, ceea ce permite utilizatorilor să își maximizeze rapid înțelegerea aplicației. Aplicația de back-end este astfel construită pentru a nu lăsa erori în cadrul procesului de creare a fișelor de disciplină.

Printre funcționalitățile principale ale aplicației se numără:

- Adăugarea și gestiunea ierarhiei de instituții, facultăți, departamente și domenii de studiu din cadrul universității
- Adăugarea și gestiunea cadrelor didactice
- Adăugarea și gestionarea materiilor
- Adăugarea și gestionarea fișelor de disciplină
- Audit de securitate pentru toate operațiile de mai sus
- Vizualizare versiuni anterioare ale fișelor de disciplină
- Export fișe de disciplină în format PDF

5.2.1. Adăugarea și gestiunea ierarhiei de instituții, facultăți, departamente și domenii de studiu din cadrul universității

Datorită faptului că fișele de disciplină există pentru toate domeniile de studiu din cadrul Universității Tehnice din Cluj-Napoca, am decis să adaug o funcționalitate care permite adăugarea facultăților, departamentelor și a domeniilor de studiu. Pentru a nu limita aplicația doar la Universitatea Tehnică din Cluj-Napoca, am adăugat și posibilitatea de a adăuga instituții, deci aplicația poate fi folosită și de către extensiile universtatii din alte orașe. Un utilizator care se mulează pe cerințele protocolului de autorizare **MustBeAdmin**, poate edita, adăuga și șterge obiecte de tipul instituție, facultate, departament sau domeniu de studii. Ceilalți utilizatori pot doar vedea arborele intitutional.

5.2.2. Adăugarea și gestiunea cadrelor didactice

Cadrele didactice reprezintă un lucru substanțial în orice instituție de învățământ, tot odată reprezintă și detalii indispensabile ale unei fișe de disciplină. Utilizatorii(cadrele didactice) se pot conecta la aplicație și pot modifica detaliile fișelor de disciplină dacă un administrator le-a asignat rolul de Editor. Administratorii au puterea de a adăuga, edita și șterge utilizatori, însă ștergerea este una de tipul soft-delete, ceea ce înseamnă că

contul cadrului didactic este trecut într-o stare de inactivitate, dar se păstrează auditul acțiunilor sale. Utilizatorii inactivi nu pot fi asignați în fișele de disciplină.

5.2.3. Adaugrea și gestionarea materiilor

Pentru a crea o fișă de disciplină trebuie să ai și disciplina căreia să îi atribui fișă. De aceea, utilizatorii care au permisiuni de Editor pot să creeze discipline noi, cărora să le creeze și fișe de disciplină. Fiecare disciplină ajunge să aibă un istoric al fișelor de disciplină. Materiilor le pot fi editate numele și codul tot de utilizatorii care îndeplinesc cerințele protocolului de autorizare "MustBeAtLeastEditor".

5.2.4. Adăugarea și gestionarea fișelor de disciplină

Această funcționalitate reprezintă funcționalitatea principală a aplicației dezvoltate de mine pentru lucrarea de licență. Adăugarea unei fișe de disciplină presupune completarea a 10 secțiuni care apoi formează fișă de disciplină integrală. Deoarece salvarea datelor în baza de date se face în cel mai econom mod al resurselor, datele care sunt trimise din aplicația de front-end către aplicația de back-end sunt stocate în tipuri de date diferite. Aplicația de back-end se ocupă de validarea și procesarea acestora, până în momentul în care datele pot fi salvate în baza de date. În cele ce urmează o să prezint procesul de creare al unei noi fișe de disciplină.

Secțiunea 1

În secțiunea 1 a fișei de disciplină se regăsesc date despre ierarhia de instituții unde se predă disciplină pentru care se încearcă adăugarea unei noi fișe și alte informații legate de formă de învățământ pentru care se predă materia.

<pre> 1 "section1": { 2 "cycleOfStudy": "string", 3 "programOfStudy": "string", 4 "qualification": "string", 5 "formOfEducation": "string", 6 "institutionId": "uuid", 7 "facultyId": "uuid", 8 "departmentId": "uuid", 9 "fieldOfStudyId": "uuid" 10 }, </pre>	<pre> 1 "section1": { 2 "id": "uuid", 3 "institutionId": "uuid", 4 "facultyId": "uuid", 5 "departmentId": "uuid", 6 "fieldOfStudyId": "uuid", 7 "cycleOfStudy": "string", 8 "programOfStudy": "string", 9 "qualification": "string", 10 "formOfEducation": "string" 11 }, </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Figura 5.17: Diferența dintre datele de intrare(POST) și datele de ieșire(GET) în aplicația de back-end

Din fericire pentru secțiunea 1, datele necesare creerii sale sunt destul de asemănătoare. Singură diferența între ele este faptul că la operația de citire a unei secțiuni 1 WebAPI-ul trimite și id-ul secțiunii. Înainte de a se insera datele furnizate de utilizator pentru secțiunea 1, se validează autenticitatea id-urilor. Dacă acestea se găsesc în baza de date atunci secțiunea 1 este validă și se poate trece mai departe.

Secțiunea 2

```

1  "section2": {
2      "yearOfStudy": 0,
3      "semester": 0,
4      "assessment": 0,
5      "category1": 0,
6      "category2": 0,
7      "teacherId": "uuid",
8      "teachers": [
9          "uuid"
10     ]
11 },

1  "section2": {
2      "id": "uuid",
3      "yearOfStudy": 0,
4      "semester": 0,
5      "assessment": 0,
6      "category1": 0,
7      "category2": 0,
8      "teacher": {
9          "id": "uuid",
10         "name": "string",
11         "email": "string",
12         "role": 0 },
13     "teachers": [{
14         "id": "uuid",
15         "name": "string",
16         "email": "string",
17         "role": 0 }]}],

```

Figura 5.18: Diferența dintre datele de intrare(POST) și datele de ieșire(GET) în aplicația de back-end

În secțiunea 2 a fișei de disciplină, se salvează date despre dascălii care vor preda disciplina, atât la curs cât și la laborator, seminar sau proiect, plus alte informații legate de anul de studiu în care se preda disciplina, semestrul, tipul de evaluare: examen, colocviu sau verificare și regimul disciplinei. Pentru această secțiune lucrurile se complică un pic, deoarece tipurile de date sunt diferite între back-end și front-end, respectiv operația de adăugare sau citire. Atunci când se procesează datele pentru adăugarea unei noi fișe de disciplină, informațiile sunt salvate în slice-ul sections din store-ul de Redux. Deoarece datele despre profesor sunt luate dintr-un autocomplete, în Redux datele vor fi salvate în tipul SelectType 5.16. Pentru a trimite mai departe informațiile, acestea vor fi mapate la tipul string și se va lua doar proprietatea de value, de aceea se poate vedea în prima imagine de cod că pentru dascăl se trimite un uuid(se mapează pe tipul string în front-end) și pentru dascălii care țin laboratoare, seminare și proiecte se trimite un vector de uuid-uri. În schimb când se citesc date din back-end, se primesc obiecte mai complicate din care aplicația de front-end își ia doar proprietățile necesare. Pentru validarea acestei secțiuni, aplicația de back-end parcurge vectorul de profesori și verifică dacă aceștia există în tabela Teachers, în schimb dacă se execută o operație de update se inserează noi date în tabela SyllabusTeachers cu id-ul profesorului și id-ul noii versiuni de fișă de disciplină.

Secțiunea 3

În secțiunea 3 a fișei de disciplină se păstrează informații legate despre timpul alocat materiei, timp contorizat în ore și împărțit în timp alocat de facultate pentru a preda materia și timp care ar trebuie alocat de student pentru studiu individual. Pentru această secțiune nu sunt necesare transformări suplimentare ale datelor, ele fiind asemănătoare.

Secțiunea 4


```

1  "section4": {
2      "id": "uuid",
3      "competences": [
4          "string"
5      ],
6      "subjects": [
7          {
8              "id": "uuid",
9              "name": "string",
10             "code": "string",
11             "hasSyllabus": true
12         }
13     ]
14 },

```

Figura 5.19: Diferența dintre datele de intrare(POST) și datele de ieșire(GET) în aplicația de back-end

Secțiunea 4, cu toate că este o secțiune opțională, permite persoanei care creează fișa de disciplină să adauge condiții de curriculum(discipline care trebuie să fie promovate pentru a promova cu succes și disciplină în cauza) și condiții de competențe. Deoarece condițiile de competențe sunt o listă de propoziții m-am gândit să implementez același principiu. În secțiunea 4 se folosește pentru prima dată componentă generală ContentInput, despre care am prezenta mai multe detalii în secțiunea de componente ale aplicației front-end React. Această componentă păstrează o listă de string-uri, care mai apoi va fi trimisă către back-end unde cu ajutorul clasei StringConverter scrisă de mine, o să transform lista de string-uri într-un text care va fi inserat în baza de date. Deoarece am decis că toate operațiile să fie executate în back-end, atunci când se solicită o resursă de tipul Section4, textul va fi remodelat într-o listă de string-uri. Pentru a crea o intrare în baza de date de tipul Section4 utilizatorul trebuie să trimită o listă de id-uri reprezentative pentru materii. Asemănător secțiunii 2, se vor insera înregistrări în tabela SyllabusSubjects care este reprezentativă relației many-to-many între fișa de disciplină și disciplinele de la secțiunea 4.1. Subiectele sunt primite de către front-end în formatul de mai sus, iar aplicația React transformă acel vector de obiecte într-un vector de obiecte de tipul SelectType.

Secțiunea 5

Secțiunea 5 este și ea o secțiune opțională, însă ambele sale input-uri sunt componente ContentInput, ceea ce înseamnă că se va realiza din nou transformarea listelor de string-uri în texte, care se inserează în baza de date, iar atunci când se preia un element din baza de date se transformă în lista de string-uri.

Secțiunea 6

Secțiunea 6 a fișei de disciplină afișează informații legate de competențele specifice acumulate, fie ele profesionale sau transversale. Pentru această componentă se folosesc din nou 2 ContentInput-uri și se va aplica același principiu ca la secțiunea anterioară.

<pre> 1 "section5": { 2 "course": [3 "string" 4], 5 "application": [6 "string" 7] 8 }, </pre>	<pre> 1 "section5": { 2 "id": "uuid", 3 "course": [4 "string" 5], 6 "application": [7 "string" 8] 9 }, </pre>
-------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------

Figura 5.20: Diferența dintre datele de intrare(POST) și datele de ieșire(GET) în aplicația de back-end

Secțiunea 7

Secțiunea 7 a fișei de disciplină afișează informații legate de obiectivele acestei discipline. Pentru această secțiune am folosit un input simplu multiline și un ContentInput.

Secțiunea 8

În fișa de disciplină curentă secțiunea 8 este afișată sub formă unui tabel care conține informații legate de curs și laborator. Informațiile furnizate de această secțiune sunt: denumirea cursului/laboratorului, durata și observații, și încă două informații un pic mai generale: metode de predare și bibliografie. Această secțiune fiind unică comparativ cu toate celelalte am ales o altă abordare. Nu am putut să fac o concatenare de string-uri, deoarece aici un obiect era mai complicat având în compoziție două string-uri și un număr. De aceea am decis să mai adaug încă un tabel care va forma o relație de one-to-many cu secțiunea 8, deci o secțiune poate avea mai multe section8Element, după cum le-am numit eu.

După cum se poate observa în exemplele de JSON de mai sus elementele de tipul section8Element sunt împărțite în categorii: laborator și curs, ceea ce a făcut necesară adăugarea unui nou field în tabela care să poată face diferențierea între elementele pentru laborator și cele pentru curs. Această secțiune poate fi numită cea mai complicată deoarece are în compoziție 4 ContentInput-uri și 2 ContentOptionsInput-uri. Concluzia este că primele 4 proprietăți se vor transforma individual într-un singur text concatenat din stringurile prezente în lista, iar celelalte două se vor salva separat într-o altă tabelă.

Secțiunea 9

Secțiunea 9 este cea mai banală secțiune. Această păstrează informații sub formă unui string imens.

Secțiunea 10

Secțiunea 10 oferă informații referitoare la evaluare. Acesta secțiune, cu toate că este organizată sub formă unui tabel asemănător cu secțiunea 8, numărul de date posibile este presetat, ceea ce îmi spune că pot exista doar 16 field-uri. Aceste field-uri sunt interpretate majoritatea de tipul de date string, sau lista de string-uri, despre care am discutat anterior.

```

1  "section8": {
2    "teachingMethodsCourse": [
3      "string"
4    ],
5    "teachingMethodsLab": [
6      "string"
7    ],
8    "bibliographyCourse": [
9      "string"
10   ],
11   "bibliographyLab": [
12     "string"
13   ],
14   "lecturesCourse": [
15     {
16       "name": "string",
17       "duration": 0,
18       "note": "string"
19     }
20   ],
21   "lecturesLab": [
22     {
23       "name": "string",
24       "duration": 0,
25       "note": "string"
26     }
27   ]
28 },

1  "section8": {
2    "id": "uuid",
3    "teachingMethodsCourse": [
4      "string"
5    ],
6    "teachingMethodsLab": [
7      "string"
8    ],
9    "bibliographyCourse": [
10     "string"
11   ],
12   "bibliographyLab": [
13     "string"
14   ],
15   "lecturesCourse": [
16     {
17       "id": "uuid",
18       "name": "string",
19       "duration": 0,
20       "note": "string"
21     }
22   ],
23   "lecturesLab": [
24     {
25       "id": "uuid",
26       "name": "string",
27       "duration": 0,
28       "note": "string"
29     }
30   ]
31 },

```

Figura 5.21: Diferența dintre datele de intrare(POST) și datele de ieșire(GET) în aplicația de back-end

5.2.5. Vizualizare versiuni anterioare ale fișelor de disciplină

Deoarece fișele de disciplină se schimbă relativ repede, am decis să implementez o funcționalitate care să permită vizualizarea versiunilor anterioare ale fișelor de disciplină. Acesta funcționalitate am implementat-o împrumutând principiile tabelelor temporale care sunt un feature disponibil în multe alte tipuri de baze de date, dar nu și în bazele de date de tip PostgreSQL. Am prezentat aspectele teoretice într-o secțiune în capitolele anterioare, așa că o să spun cum am decis să implementez acest tip de tabel temporal. Atunci când se creează o fișă de disciplină nouă, se inserează un nou obiect în tabela SyllabusVersions. Obiectul de tipul SyllabusVersion conține o cheie străină către un obiect de tipul fișă de disciplină și încă două proprietăți: CreatedAt care ne spune dată și ora la care a fost adăugat obiectul și UpdatedAt care ne spune dată și ora la care s-a creat o versiune nouă a fișei de disciplină. Deci atunci când se editează o fișă de disciplină după ce se trece de validări, creare de secțiuni și se ajunge la inserarea noului obiect în baza de date, se actualizează câmpul UpdatedAt cu dată și ora curentă, și apoi se adaugă o nouă inserție cu id-ul noii fișe de disciplină. Atunci când se șterge o fișă de disciplină, această nu se șterge cu adevărat din baza de date, doar se trece într-o formă de inactivitate prin actualizarea câmpului UpdatedAt. În tabela de discipline, utilizatorul poate apăsa pe un buton din categoria de Actions și îi apare un modal care îi spune câte versiuni a mai avut fișă curentă și îi permite să descarce un pdf cu versiunea dorită de fișă de disciplină.

5.2.6. Export fișe de disciplină în format PDF

Această funcționalitate a fost dezvoltată pe ultima sută de metri, nu generează cel mai rețușat fișier PDF, dar datele sunt disponibile și lizibile în fișierul generat. Utilizatorul poate să își genereze un fișier PDF atât pentru versiunea curentă a unei fișe de disciplină a unei materii cât și pentru fișele de disciplină mai vechi aflate în istoricul fișelor de disciplină.

5.2.7. Audit de securitate pentru toate operațiunile de mai sus

O aplicație buna de management trebuie să aibă un audit de securitate, așa că am decis să implementez și eu unul. Auditul de securitate poate fi văzut doar de către admini și arată într-un tabel toate operațiunile executate în ordine invers cronologică. Informațiile stocate în acest audit de securitate sunt de tipul tuplelor și conțin următoarele informații: cine a realizat acțiunea, ce acțiune a fost făcut, asupra cărui obiect a fost făcută acțiunea și note despre acțiunea respectivă.

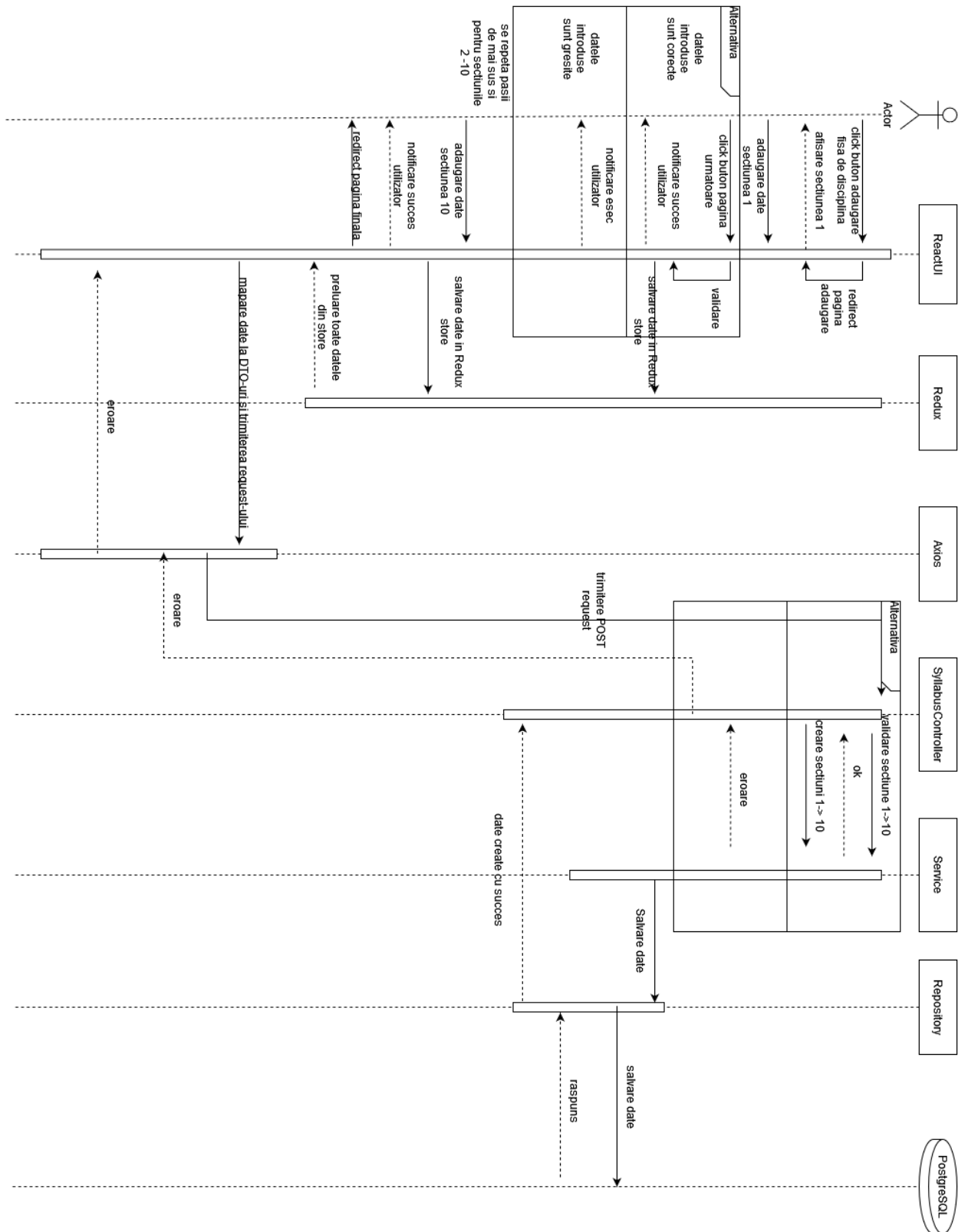


Figura 5.22: Diagramă de secvență pentru operația de adăugare fișa de disciplină

Capitolul 6. Testare și Validare

Testarea progresivă pe parcursul dezvoltării unei aplicații reprezintă cheia succesului acelei aplicații. O aplicația care nu are erori, care funcționează 100% bine nu există, cel puțin nu de la prima versiune dată clientului. Echipa de dezvoltatori trebuie să facă tot ce le stă la îndemână să reducă numărul de erori în aplicați, aceste lucruri pot fi remediate doar prin teste automate sau manuale. În cazul aplicației mele, testarea a fost realizată manual folosind aplicația de front-end pentru a testa aplicația de back-end și invers. În cadrul dezvoltării aplicației de back-end am folosit și alte unelte de testare a endpoint-urilor precum: SwaggerUI sau Postman. Datorită faptului că aplicațiile de back-end și de front-end au fost dezvoltate în paralel comunicarea a putut fi testată la fiecare commit nou adăugat în repository-ul de GitHub.

6.1. Back-end

Aplicația de back-end: WebAPI-ul scris în ASP.NET Core 6 a fost dezvoltată în așa fel încât să testeze și să valideze corectitudinea datelor. În cazul în care datele introduse de utilizator sunt invalide, sau nu respectă formatul prestabilit, aplicația încheie procesarea cererii și trimite un mesaj de eroare.

Datorită construcției framework-ului ASP.NET Core 6, serverul de back-end are o toleranță ridicată la erori. Dacă framework-ul sesizează că o operație nu a funcționat bine se și încheie requestul și se trimite un mesaj de eroare spre utilizator. Mesajele de eroare utilizate au o mare diversificare: începând cu mesajul de eroare 400(bad request), 401(Unauthorized), 403 (Forbidden) până la 404(resource not found). Această diversitate de mesaje de eroare rezultă într-o getiune foarte bună pe partea de front-end pentru erorile de API apărute. Pe lângă statusul mesajului de eroare, am decis să mai trimit și un scurt mesaj care să ajute utilizatorul să înțeleagă ce a greșit pentru a putea corecta. Mesajele de obicei au un format asemănător cu cel al auditului, de exemplu: utilizatorul a încercat să creeze o resursă, dar există deja altă entitate care are același nume/cod. Datorită mesajelor detaliate utilizatorului îi este ușor să înțeleagă ce a greșit și poate remedia situația înainte de a înainta un alt request.

Din nefericire aplicația de back-end a fost testată doar manual, însă este deschisă automatizării și adăugării de teste de tip unit(teste care verifică corectitudinea funcțiilor prin folosirea funcției cu parametri care generează un răspuns cunoscut, dacă răspunsul este același cu cel așteptat atunci se poate spune că funcția a fost scrisă corect), integration(sunt teste care testează end-point-urile WebAPI-ului meu, din nou, se adaugă date care generează un răspuns deja cunoscut, dacă răspunsul la care a ajuns funcția este același cu răspunsul așteptat atunci funcția a fost scrisă corect) sau chiar E2E(End-To-End, teste care simulează funcționalitățile aplicației și testează dacă se întâmplă lucrurile care ar trebui să se întâmple).

6.2. Font-end

Aplicația de front-end a fost și ea de asemenea testată în regim manual, și calibrată să funcționeze în parametri optimi, sincron cu aplicația de back-end. Comparativ cu aplicația de back-end căreia răspunsul îi poate fi văzut și interpretat doar dacă ai cunoștințe în domeniul tehnic, aplicația de front-end oferă alternative mai prietenoase pentru utilizator precum: colorarea cu roșu a unui input, afișarea unui mesaj de eroare, afișarea unui Toaster care conține un mesaj de eroare, sau interpretează mesajul primit de la serverul de back-end și îl transpune într-un element mai ușor de urmărit și înțeles.

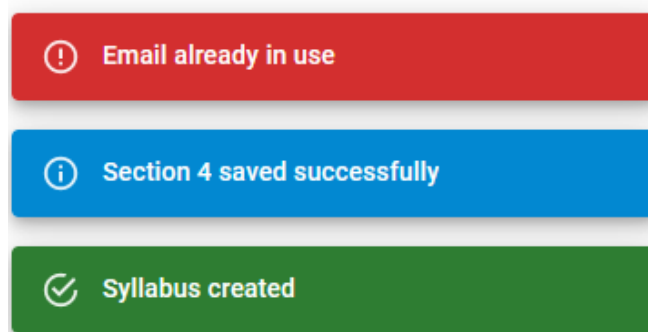


Figura 6.1: Exemple de mesaje de atenționare pentru utilizator

Pentru testarea aplicației web am folosit extensii ale browser-ului Mozilla Firefox Developer Edition. Aceste extensii au rulat o serie de teste care verifică au verificat accesibilitatea, compatibilitatea cu noile versiuni, performanța și securitatea aplicației web. În urmă rulării testelor am observat erori care țin atât de partea de CSS(proprietăți care nu sunt disponibile pe toate tipuri de browser) cât și partea de HTML(am descoperit unele componente care nu aveau un identificator unic, sau care aveau același indetificator unic) și de Typescript(nu erau setate unele headere pentru requesturi sau chiar lipseau de tot).

Capitolul 7. Manual de Instalare și Utilizare

7.1. Instalare

Pentru a continua dezvoltarea aplicației, în cele ce urmează voi descrie pas cu pas procesul pe care trebuie să îl parcurgă un programator pentru a seta environment-ul de dezvoltare.

7.1.1. Resurse hardware

Resursele hardware necesare pentru continuarea dezvoltării aplicației sunt:

- stație de lucru
 - minimum:
 - * procesor: 1.8Ghz, quad-core, 64bit
 - * memorie: 4 GB RAM,
 - * spațiu stocare: 20-50 GB HDD
 - recomandat:
 - * procesor: 3.20 Ghz, octa-core, 64bit
 - * memorie: 16 GB RAM,
 - * spațiu stocare: 50 GB SSD
- periferice
- monitor
- conexiune stabilă la internet

7.1.2. Resurse software

Resursele software minim necesare pentru continuarea dezvoltării aplicației sunt:

- sistem de operare: Windows 10 | 11, Linux, MacOS
- .Net 6
- ASP.NET Core 6
- Entity Framework 6
- C# 10
- Node 14.0.0
- npm 5.6
- Visual Studio 2022(recomandat) și Visual Studio Code
- Postman
- PostgreSQL
- Browser la alegere, eu prefer Firefox Developer Edition
- Git ultima versiune

7.1.3. Back-end

Instalare pas cu pas

Pași care trebuie urmați pentru a instala resursele necesare dezvoltării aplicației:

- Visual Studio: visualstudio.microsoft.com/downloads , eu am folosit Community Edition

- .NET: dotnet.microsoft.com/downloads -> se descarcă .NET SDK x64
- Postman: postman.com/downloads
- PostgreSQL: postgresql.org/download
- accesare GitHub și clonare repository: github.com/EugeniuCojocaru/licenta_be
- se deschide proiectul în Visual Studio
- se deschide Package Manager Console și se rulează comandă **update-database**, această va procesa toate fișierele de migrări și va forma baza de date

7.1.4. Front-end

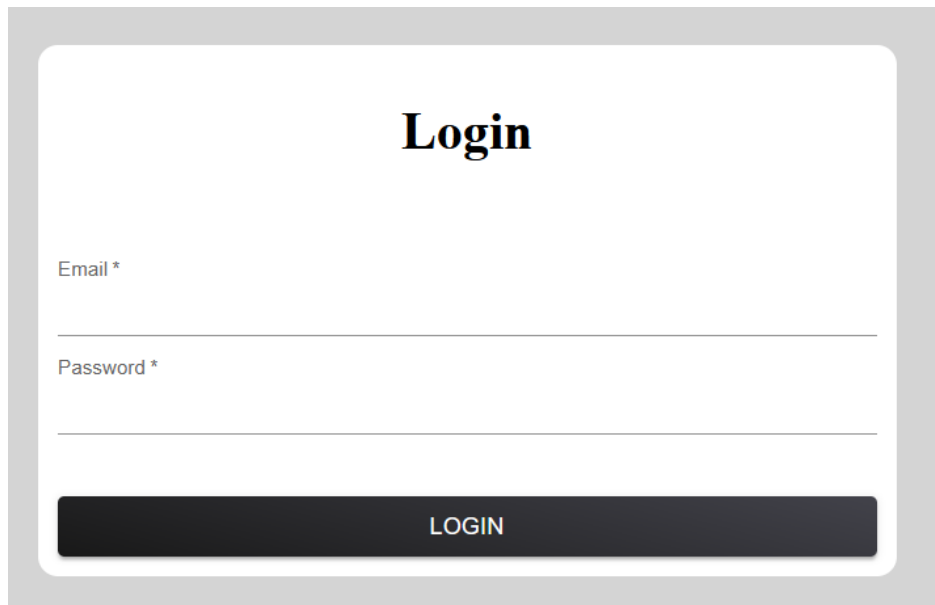
Instalare pas cu pas

Pașii care trebuie urmați pentru a instala resursele necesare dezvoltării aplicației:

- Visual Studio Code: visualstudio.microsoft.com/downloads
- Node: nodejs.org/en/download
- Browser: mozilla.org/en-US/firefox/developer
- accesare GitHub și clonare repository: github.com/EugeniuCojocaru/licenta_2022_frontend
- se deschide proiectul în Visual Studio Code
- se deschide un terminal la root-ul proiectului și se rulează comandă **npm install**
- după ce e gata instalarea se pornește aplicația folosind comandă **npm start**

7.2. Manual de utilizare

Pentru a putea utiliza aplicația, persoană în cauză trebuie să aibă un cont valid salvat în baza de date. Se introduc date în form-ul de logare și apoi se apasă butonul de autentificare. Dacă informațiile introduse sunt valide atunci persoană este redirecționată pe pagină Dashboard.



The image shows a login form with a white background and a light gray border. At the top, the word "Login" is centered in a bold, black, serif font. Below it, there are two input fields. The first field is labeled "Email *" in a small, gray font. The second field is labeled "Password *" in a small, gray font. Both fields have a thin gray border. At the bottom of the form, there is a dark gray button with the word "LOGIN" in white, uppercase letters.

Figura 7.1: Formul de logare în aplicație

Pentru a demonstra funcționalitățile aplicației folosesc un cont cu rol de Admin, deci toate funcționalități îmi sunt disponibile. Pentru a vedea ierarhia instituțională a

universității se apasă pe butonul **Institutions** din partea stânga a ecranului. Ca urmare a apăsării acelui buton utilizatorul este redirecționat către o altă pagină.

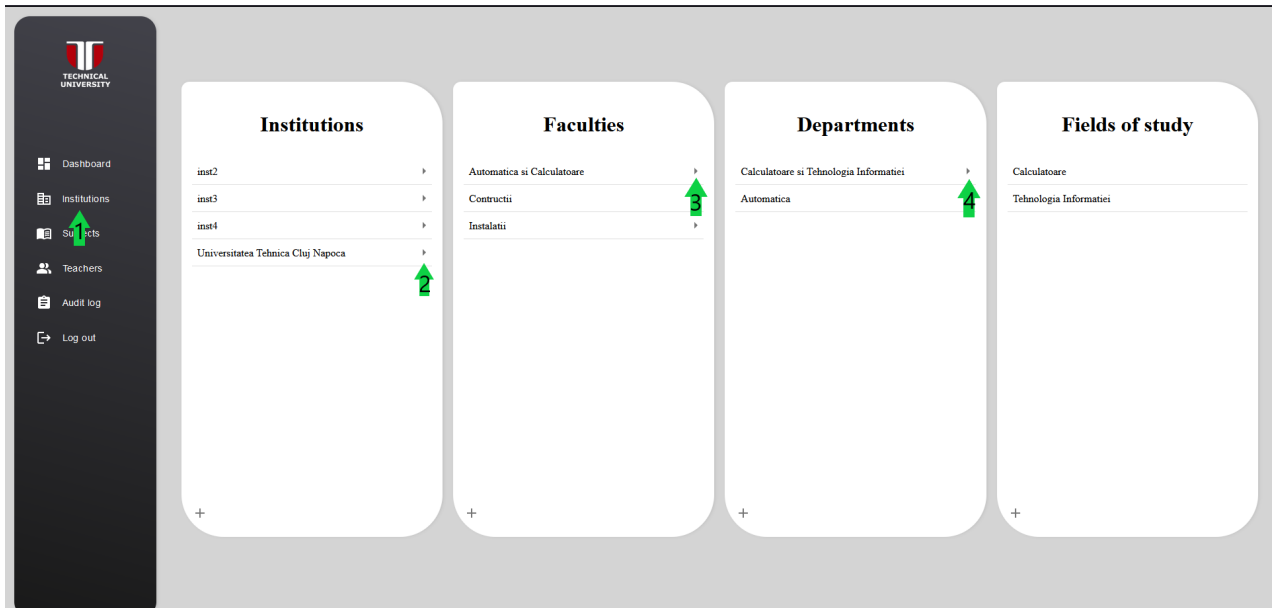


Figura 7.2: Pagina destinata gestionarii institutiilor

Orice utilizator poate să vadă lista de instituții, facultăți, departamente și domenii de studiu, însă doar utilizatorii cu dreptul de Admin pot să adauge, editeze sau șteargă obiecte de pe această pagină. Pentru a edita informația utilizatorul trebuie să dea dublu-click pe denumire, ceea ce va deschide meniul de editare. În acet meniu există 3 butoane: Save, Delete și Cancel(pentru a ieși din modul de editare). Dacă se apasă plusul din partea de jos a paginii, utilizatorul poate adaugă un nou obiect. Butoanele, cu toate că sunt doar iconițe afișează sunb ele informații referitoare la operația pe care o execută.

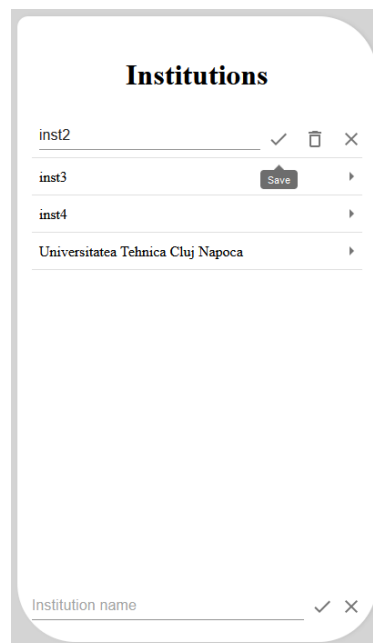


Figura 7.3: Meniul de editare si adaugare

Următoarea funcționalitate este cea de gestiune a disciplinelor. Pe această pagină se regăsește un tabel multifuncțional format din 3 coloane. Operațiile care pot fi executate asupra datelor din tabel se regăsesc în coloana 3. În figura ?? se pot vedea funcționalitățile disponibile pentru o disciplină: editare, creare fișă nouă de disciplină, ștergere fișă de disciplină, descărcare fișă disciplină și vizualizare istoric fișă disciplină.

Code	Name	Actions
0	disciplina0	[edit] [delete] [history]
1	disciplina1	[edit] [delete] [history]
2	disciplina2	[edit] [delete] [history]
123	DISI	[edit] [delete] [history]
3	disciplina3	[edit] [delete] [history]
4	disciplina4	[edit] [delete] [history]
5	disciplina5	[edit] [delete] [history]
6	disciplina6	[edit] [delete] [history]

Figura 7.4: Pagina destinata gestionarii disciplinelor

Apăsând pe butonul **Add new** se deschide un modal care permite adugarea unei noi discipline. Apăsândpe iconița de vizualizare isotoric se deschide un alt modal care afișează un tabel cu versiunile anterioare ale fișelor de disciplină pentru materia selectată.

Add subject

Name *

Code *

CANCEL CREATE WITH SYLLABUS CREATE

Syllabus Id	Created At	Updated At	Actions
603297f8-cdf9-48af-b612-f8d5dbb8fc64	28/06/2022 14:48:04	01/01/1 00:00:00	[download]
5893c355-a486-4602-88bc-853c78db0ef7	28/06/2022 03:08:46	28/06/2022 03:38:09	[download]

Figura 7.5: Modal de adăugare disciplină și modal de vizualizare istoric fișe disciplină

Pagina dedicată gestionării utilizatorilor este asemănătoare cu cea pentru gestionarea disciplinelor. La fel există un buton pentru adăugarea de utilizatori noi. Parolă implicită pentru crearea unui nou utilizator este **Licenta2022UTCN**, iar utilizatorul nou trebuie să își schimbe parolă la prima logare.



Users			
Name	Email	Role	Actions
admin	admin@admin.admin	Admin	 
admin2	admin2@utcn.com	Admin	 
user1	user1@utcn.com	User	 
user2	user2@utcn.com	User	 

Figura 7.6: Pagina de gestiune utilizatori

Add user

Name *

Email *

Role *

Admin

CANCEL CREATE

Figura 7.7: Modal adaugare utilizatori

Pagina de audit este doar pentru vizualizare, nu se pot efectua alte operații pe acea pagină.

User	Operation	Entity	At date	Notes
admin	UPDATE	Teacher	02/07/2022 00:09:01	god -> user2
admin	UPDATE	Teacher	02/07/2022 00:08:42	user1 -> user1
admin	UPDATE	Teacher	02/07/2022 00:08:33	admin -> admin
admin	UPDATE	Teacher	02/07/2022 00:08:12	admin -> admin
admin	UPDATE	Teacher	02/07/2022 00:08:01	1sdfsdfsdf -> admin2
admin	UPDATE	Teacher	02/07/2022 00:07:40	13 -> user1
admin	UPDATE	Subject	01/07/2022 21:55:32	sdfsdmjfkfsdfsdfsdfsdf -> disciplina7
admin	UPDATE	Subject	01/07/2022 21:54:56	2 -> disciplina6

Figura 7.8: Pagina audit

Pentru adăugarea unei noi fișe de disciplină trebuie completate cele 10 secțiuni:

1. Data about the program of study

Institutions *

Universitatea Tehnica Cluj Napoca

Faculties *

Automatica si Calculatoare

Departments *

Calculatoare si Tehnologie Informatiei

Fields of study *

Calculatoare

Cycle of study *

qwe

Qualification *

qwe

Program of study *

qwe

Form of education *

qwe

Figura 7.9: Sectiunea 1

2. Data about the subject

Subject name *

disciplina0

Course responsible *

admin2

Teachers in charge of seminars/ laboratory/ project

admin admin2

Year of study *

II

Semester *

2

Type of assessment *

Colloquium

Subject category *

Mandatory

Speciality

Figura 7.10: Sectiunea 2

3 Estimated total time

Number of hours per week: 10

Course *

3

Seminar *

2

Laboratory *

2

Project *

3

Number of hours per semester: 140

Course *

42

Seminar *

28

Laboratory *

28

Project *

42

Individual study: 12

Manual, lecture material (nd notes, bibliography) *

2

Supplementary study in the library, online and in the fields *

2

Preparation for seminars/laboratory works, homework, reports, portfolios, essays *

2

Tutoring *

2

Exams and tests *

2

Other activities *

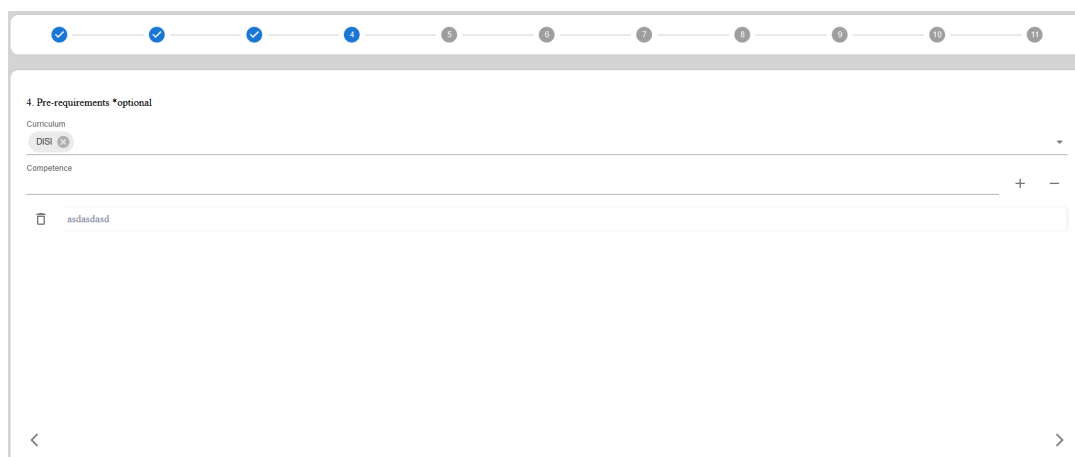
2

Total hours per semester: 152

Credits *

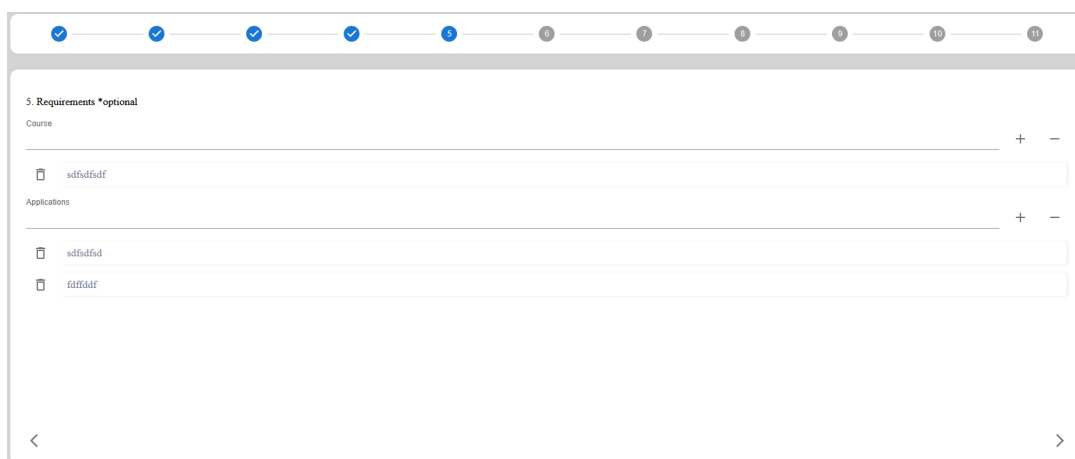
7

Figura 7.11: Sectiunea 3



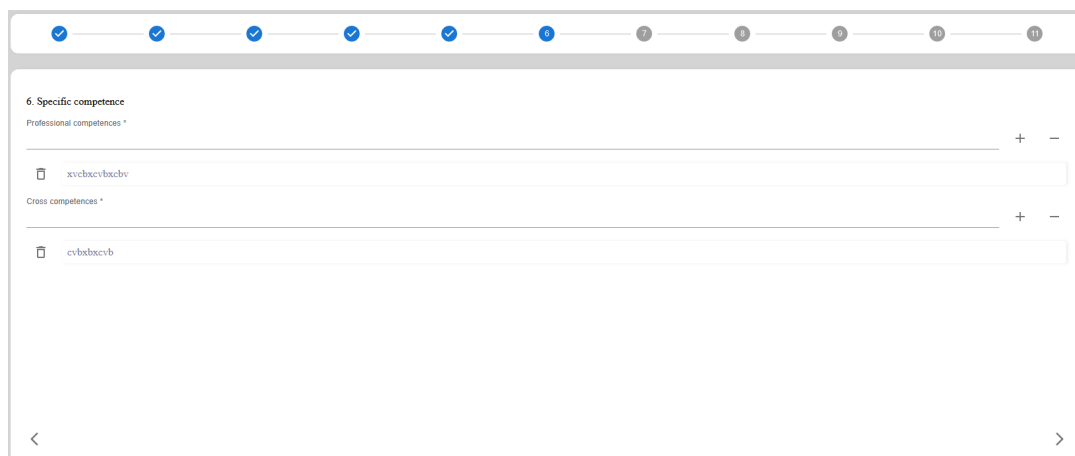
The screenshot shows a mobile application interface for 'Section 4: Pre-requirements *optional'. At the top, a progress bar contains 11 circular icons; the first four are blue with white checkmarks, and the remaining seven are grey with white numbers 4 through 11. The main content area has a title '4. Pre-requirements *optional' and a 'Curriculum' section with a 'DISI' button. Below this is a 'Competence' section with a text input field containing 'asdasdasd'. The interface includes expand/collapse icons (+/-) and navigation arrows (</>) at the bottom.

Figura 7.12: Sectiunea 4



The screenshot shows a mobile application interface for 'Section 5: Requirements *optional'. The progress bar at the top shows the fifth icon as a blue circle with a white number 5. The main content area has a title '5. Requirements *optional' and a 'Course' section with a text input field containing 'sdfsfdf'. Below this is an 'Applications' section with two text input fields, the first containing 'sdfsfdf' and the second containing 'fdfdf'. The interface includes expand/collapse icons (+/-) and navigation arrows (</>) at the bottom.

Figura 7.13: Sectiunea 5



The screenshot shows a mobile application interface for 'Section 6: Specific competence'. The progress bar at the top shows the sixth icon as a blue circle with a white number 6. The main content area has a title '6. Specific competence' and a 'Professional competences *' section with a text input field containing 'xvcbxcvbxcbv'. Below this is a 'Cross competences *' section with a text input field containing 'cvbxbxcvb'. The interface includes expand/collapse icons (+/-) and navigation arrows (</>) at the bottom.

Figura 7.14: Sectiunea 6

7. Discipline objective (as results from the key competences gained)

General objective *

xcvbxcbvxcvbxcb

Specific objectives

- xcvbxcbvbf
- 3345345

Figura 7.15: Secțiunea 7

8. Contents

Lectures

sdgsgdsg

Title: sdgsgdsg

Duration (in hours): 0

Notes

Time allocated: 2

sdgsgdsg

Title: sdgsgdsg

Duration (in hours): 0

Notes

Time allocated: 1

Figura 7.16: Secțiunea 8

9. Bridging course contents with the expectations of the representatives of the community, professional associations and employers in the field

Description *

sdgsgdsgdsg

Figura 7.17: Secțiunea 9

2. Data about the subject

Course assessment criteria *	Course assessment methods *	Course weight in the final grade *
sdfgsdfg	sdfgsdfg	100
Seminar assessment criteria	Seminar assessment methods	Seminar weight in the final grade
		0
Laboratory assessment criteria	Laboratory assessment methods	Laboratory weight in the final grade
		0
Project assessment criteria	Project assessment methods	Project weight in the final grade
		0
Minimum standard of performance		
sdfgsdfg		
Conditions for participating in the final exam		
+ -		
sdfgsdfg		
Conditions for promotion *		
dsfgsdgsldgsdf		

< >

Figura 7.18: Sectiunea 10

După completarea acestor pagini se apasă butonul de creare fișa disciplină.

Capitolul 8. Concluzii

8.1. Rezultate obținute

Produsul final, rezultat în urmă finalizării lucrării de licență, este unul pe măsura eforturilor, deoarece consider că am reușit să creez o aplicație de gestiune a fișelor de disciplină funcțională și utilizabilă. Elementele de securitate întăresc încrederea în calitatea și integritatea aplicației. Datorită funcționalității de role-management administratorul poate să gestioneze mai bine resursele atât umane cât și de date. Consider că pentru versiunea 0.0.1 aplicația este decentă, bine-înțeles există lucruri de îmbunătățit, dar aceste lucruri trebuie discutate cu persoana care utilizează această unealtă.

Aplicația poate să stocheze informații legate de fișele de disciplină și chiar să genereze fișiere pdf cu informațiile disponibile. Conectarea se face doar pe baza de JWT Token. Web API-ul nu permite executarea operațiilor neautorizate, autorizarea fiind furnizată de către informațiile stocate în JWT Token. Toate obiectivele propuse în capitolul 2 au fost îndeplinite cu succes. Pe lângă aplicația în sine, mi-am dezvoltat cunoștințele mele prin folosirea de limbajelor de programare care nu au fost studiate în detaliu la facultate(Javascript, Typescript, C#), framework-uri noi(ASP.NET Core 6) și diverse librării(Redux, Styled Components, Entity Framework).

8.2. Descriere a posibilelor dezvoltări și îmbunătățiri ulterioare

Deoarece nimic nu este perfect și aplicația de gestiune a fișelor de disciplină dezvoltată de mine are nevoie de unele îmbunătățiri ulterioare dintre care pot evidenția următoarele:

- îmbunătățirea arhitecturii:
 - Back-end
 - * refacerea pachetului responsabil de service în aplicația de back-end
 - * refacere tabeli Teachers în 3 tabele: Users, Teachers, Roles. Această schimbare ar permite și altor utilizatori să primească drepturi de acces și să ofere o mână de ajutor în aplicație
 - * îmbunătățirea tabeli de Audit pentru a permite stocarea mai multor informații pentru a furniza un audit de securitate mai detaliat
 - Front-end
 - * mutarea tuturor funcțiilor de comunicare cu aplicația de back-end în Redux, acest lucru al facilita imens viteză de răspuns a aplicației
 - * folosirea răspunsurilor de tip Promise pentru comunicarea cu WebAPI-ul, acestea fiind mai rapide decât blocul `try{}catch{}`
- standardizarea tipurilor de date folosite atât de back-end cât și de front-end, pentru a scădea numărul de mapari necesare pentru a ajunge la tipul de date corect
- adăugarea funcționalitatilor de filtrare și paginare => crearea unui tabel generalizat pentru aplicația de front-end
- îmbunătățirea interfeței grafice să fie mai clară și mia explicită(lucru care se poate face doar cu ajutorul feedback-ului clientului)

- refacerea fișierului PDF exportat astfel încât să folosească generearea de PDF pe baza unui fișier html
- îmbunătățirea interfeței grafice astfel încât aplicația să fie utilizabilă și de pe dispozitive mobile
- un utilizator nou adăugat să primească o parolă standard pe adresa de email, iar la prima logare aplicația să îl oblige să își schimbe parola pentru a putea accesa funcționalitățile aplicației
- adăugarea mai multor informații utile pe Dashboard
- furnizarea opțiunii de vizualizare a versiunilor anterioare a fișelor de disciplină fără a genera pdf-uri

Bibliografie

- [1] “Api development for everyone.” [Online]. Available: <https://swagger.io/>
- [2] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 1st ed. USA: Prentice Hall PTR, 2008.
- [3] Rick-Anderson, “Create web apis with asp.net core.” [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/web-api/?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0
- [4] “Javascript with syntax for types.” [Online]. Available: <https://www.typescriptlang.org/>
- [5] “Getting started with redux,” Dec 2021. [Online]. Available: <https://redux.js.org/introduction/getting-started>
- [6] “Getting started.” [Online]. Available: <https://axios-http.com/docs/intro>
- [7] Js-Cookie, “Js-cookie/js-cookie: A simple, lightweight javascript api for handling browser cookies.” [Online]. Available: <https://github.com/js-cookie/js-cookie>
- [8] “React router: Declarative routing for react.” [Online]. Available: <https://v5.reactrouter.com/web/guides/quick-start>
- [9] “Installation - material.” [Online]. Available: <https://v4.mui.com/getting-started/installation/>
- [10] Styled-Components, “Components: Documentation.” [Online]. Available: <https://styled-components.com/docs>
- [11] “Documentation.” [Online]. Available: <https://www.postgresql.org/docs/>
- [12] “Automapper.” [Online]. Available: <https://docs.automapper.org/en/stable/>
- [13] Rick-Anderson, “Getting started - ef core.” [Online]. Available: <https://docs.microsoft.com/en-us/ef/core/get-started/overview/first-app?tabs=netcore-cli>
- [14] BcryptNet, “Bcryptnet/bcrypt.net: Bcrypt.net - bringing updates to the original bcrypt package.” [Online]. Available: <https://github.com/BcryptNet/bcrypt.net>

Anexa A. Exemplu fisa de disciplina

FIȘA DISCIPLINEI

1. Date despre program

1.1 Instituția de învățământ superior	Universitatea Tehnică din Cluj-Napoca
1.2 Facultatea	Automatică și Calculatoare
1.3 Departamentul	Calculatoare
1.4 Domeniul de studii	Calculatoare si Tehnologia Informatiei
1.5 Ciclul de studii	Licență
1.6 Programul de studii / Calificarea	Calculatoare romana/ Inginer
1.7 Forma de învățământ	IF – învățământ cu frecvență
1.8 Codul disciplinei	45.

2. Date despre disciplină

2.1 Denumirea disciplinei	Sisteme distribuite				
2.2 Titularii de curs	Prof. dr. ing. Tudor Cioară – Tudor.Cioara@cs.utcluj.ro				
2.3 Titularul/Titularii activităților de seminar/laborator/proiect	Conf. dr. ing. Ionut Anghel – Ionut.Anghel@cs.utcluj.ro S.I.dr.ing. Cristina.Pop – Cristina.Pop@cs.utcluj.ro S.I.dr.ing. Marcel Antal – Marcel.Antal@cs.utcluj.ro S.I.dr.ing. Claudia Daniela Pop – Claudia.Pop@cs.utcluj.ro				
2.4 Anul de studiu	4	2.5 Semestrul	7	2.6 Tipul de evaluare (E – examen, C – colocviu, V – verificare)	E
2.7 Regimul disciplinei	DF – fundamentală, DD – în domeniu, DS – de specialitate, DC – complementară				DS
	DI – Impusă, DOp – opțională, DFac – facultativă				DI

3. Timpul total estimat

3.1 Număr de ore pe săptămână	5	din care:	Curs	2	Seminar		Laborator	2	Proiect	1
3.2 Număr de ore pe semestru	70	din care:	Curs	28	Seminar		Laborator	28	Proiect	14
3.3 Distribuția fondului de timp (ore pe semestru) pentru:										
(a) Studiul după manual, suport de curs, bibliografie și notițe										18
(b) Documentare suplimentară în bibliotecă, pe platforme electronice de specialitate și pe teren										6
(c) Pregătire seminarii / laboratoare, teme, referate, portofolii și eseuri										24
(d) Tutoriat										
(e) Examinări										12
(f) Alte activități:										
3.4 Total ore studiu individual (suma (3.3(a)...3.3(f)))							60			
3.5 Total ore pe semestru (3.2+3.4)							130			
3.6 Numărul de credite							5			

4. Precondiții (acolo unde este cazul)

4.1 de curriculum	Retele de Calculatoare, Proiectare Software, Tehnici de Programare, Baze de date
4.2 de competențe	Abilitatea de a analiza si de a proiecta o retea locala, folosind simulatoare disponibile. Abilitatea de a proiecta o aplicatie folosind arhitecturi layered. Abilitatea de a scrie cod intr-un limbaj OOP. Abilitatea de a proiecta si implementa o baza de date relationala precum si de a scrie interogari, atat in SQL cat si intr-un framework ORM

5. Condiții (acolo unde este cazul)

5.1. de desfășurare a cursului	Tabla, proiector, calculator, conexiune la Internet Platforma Microsoft Teams pentru predare online Site-ul cursului: http://www.coned.utcluj.ro/~salomie/DS_Lic
5.2. de desfășurare a laboratorului / proiectului	Calculatoare, software specific Platforma Microsoft Teams pentru predare online Site-ul cursului: http://www.coned.utcluj.ro/~salomie/DS_Lic

6. Competențele specifice acumulate

6.1 Competențe profesionale	<p>C4 - Îmbunătățirea performanțelor sistemelor hardware, software și de comunicații (2 credite)</p> <p>C4.1 - Identificarea și descrierea elementelor definitorii ale performanțelor sistemelor hardware, software și de comunicații</p> <p>C4.2 - Explicarea interacțiunii factorilor care determină performanțele sistemelor hardware, software și de comunicații</p> <p>C4.3 - Aplicarea metodelor și principiilor de bază pentru creșterea performanțelor sistemelor hardware, software și de comunicații</p> <p>C4.4 - Alegerea criteriilor și metodelor de evaluare a performanțelor sistemelor hardware, software și de comunicații</p> <p>C4.5 - Dezvoltarea de soluții profesionale pentru sisteme hardware, software și de comunicații bazate pe creșterea performanțelor</p> <p>C5 - Proiectarea, gestionarea ciclului de viață, integrarea și integritatea sistemelor hardware, software și de comunicații (2 credite)</p> <p>C5.1 - Precizarea criteriilor relevante privind ciclul de viață, calitatea, securitatea și interacțiunea sistemului de calcul cu mediul și cu operatorul uman</p> <p>C5.2 - Utilizarea unor cunoștințe interdisciplinare pentru adaptarea sistemului informatic în raport cu cerințele domeniului de aplicații</p> <p>C5.3 - Utilizarea unor principii și metode de bază pentru asigurarea securității, siguranței și usurinței în exploatarea sistemelor de calcul</p> <p>C5.4 - Utilizarea adecvată a standardelor de calitate, siguranță și securitate în prelucrarea informațiilor</p> <p>C5.5 - Realizarea unui proiect incluzând identificarea și analiza problemei, proiectarea, dezvoltarea și demonstrând o înțelegere a nevoii de calitate</p> <p>C6 - Proiectarea sistemelor inteligente (1 credit)</p> <p>C6.1 - Descrierea componentelor sistemelor inteligente</p> <p>C6.2 - Utilizarea de instrumente specifice domeniului pentru explicarea și înțelegerea funcționării sistemelor inteligente</p> <p>C6.3 - Aplicarea principiilor și metodelor de bază pentru specificarea de soluții la probleme tipice utilizând sisteme inteligente</p> <p>C6.4 - Alegerea criteriilor și metodelor de evaluare a calității, performanțelor și limitelor sistemelor inteligente</p> <p>C6.5 - Dezvoltarea și implementarea de proiecte profesionale pentru sisteme inteligente</p>
6.2 Competențe transversale	N/A

7. Obiectivele disciplinei

7.1 Obiectivul general al disciplinei	Capacitatea de a dezvolta și implementa sisteme software distribuite
7.2 Obiectivele specifice	<ul style="list-style-type: none">- Capacitatea de a proiecta sisteme distribuite la nivel architectural și de componente utilizând principalele concepte și paradigme ale sistemelor distribuite precum și relațiile lor cu alte discipline din știința calculatoarelor.- Capacitatea de a identifica principalele modele și tehnologii care pot fi folosite în proiectarea sistemelor distribuite fiind dat un set de constrângeri.- Capacitatea de a utiliza tehnologii Java și .NET pentru proiectarea sistemelor distribuite.- Capacitatea de a utiliza tehnologiile serviciilor Web – XML, SOAP, WSDL, UDDI precum și servicii REST- Capacitatea de a dezvolta servicii Web folosind tehnologiile Java și .NET.- Capacitatea de a dezvolta aplicații client pentru sisteme distribuite folosind tehnologii bazate pe Javascript- Capacitatea de a proiecta și dezvolta o platformă pentru deploymentul unei aplicații distribuite, considerând serverele implicate și setările de rețea necesare

8. Conținuturi

8.1 Curs	Nr.ore	Metode de predare	Observații
Introducere in sisteme distribuite, Caracterizarea Sistemelor Distribuite	2	Folosirea metodelor multimedia de predare si acces la Internet Predare fata in fata sau online folosind platforma Microsoft Teams Studentii sunt invitati sa colaboreze la proiectele de cercetare ale lectorului Ore de consultatii in timpul semestrului si inaintea examenului	
Modele si arhitecturi pentru sisteme distribuite, middleware	2		
Calitatea serviciilor, aspecte non-functionale ale sistemelor distribuite, metrice	2		
Comunicarea inter-procese, message-passing, sockets	2		
Model computational distribuit, Timp si cauzalitate, Ceasuri logice	2		
Stari globale, Snapshots, algoritmi distribuiti	2		
Procesarea distribuita a datelor – concepte si arhitectura de referinta	2		
RPC, RMI, XML RPC, gRPC, SOA	2		
Procesarea distribuita a datelor – tehnici de distributie a datelor	2		
Tranzactii distribuite si controlul concurenței	2		
Tratarea erorilor in Sisteme Distribuite	2		
SOA si Servicii Web	2		
Sisteme de calcul bazate pe cloud	2		
Sisteme P2P, Sisteme Adaptive, Internetul Lucrurilor, Sisteme Cyber-Fizice	2		
Bibliografie (bibliografia minimală a disciplinei conținând cel puțin o lucrare bibliografică de referință a disciplinei, care există la dispoziția studenților într-un număr de exemplare corespunzător)			
1. G. Coulouris, J.Dollimore, T.Kindberg – Distributed Systems. Concepts and Design, Addison Wesley, 2005			
2. A. Tanenbaum, M. van Steen – Distributed Systems, Prentice Hall, 2002			
3. A.D. Kshemkalyan M.Singhal - Distributed Computing, Cambridge Press 2008			
4. Ioan Salomie, Tudor Cioara, Marcel Antal - Lecture Notes, Lab Notes Project Notes and Assignments http://www.coned.utcluj.ro/~salomie/DS_Lic			
8.2 Aplicații (seminar/laborator/proiect)*	Nr.ore	Metode de predare	Observații
Paradigma Request-Reply (2 ședințe de laborator)	4	Teme si exemple predefinite Predare fata in fata sau online folosind platforma Microsoft Teams Scurta prezentare a temelor de laborator, discutii pe baza temelor, implementarea temelor pe calculator, miniproiect individual pe calculator Unealta pentru integrarea continua, testare si deployment a proiectelor	
Comunicare asincrona (2 ședințe de laborator)	4		
Apel de metoda distribuit (2 ședințe de laborator)	4		
Servicii Web SOA (2 ședințe de laborator)	4		
Proiect: Aplicatie distribuita complexa – Serviciu medical online de monitorizare si asistare la ingrijirea la domiciliu	2		
Dezvoltarea si integrarea serviciilor	4		
Deployment folosind Docker	2		
Test de laborator si prezentarea si evaluarea proiectelor studentilor	4		
Bibliografie (bibliografia minimală pentru aplicații conținând cel puțin o lucrare bibliografică de referință a disciplinei care există la dispoziția studenților într-un număr de exemplare corespunzător)			
1. Ioan Salomie, Tudor Cioara, Ionut Anghel, Tudor Salomie – Distributed Computing and Systems – A practical Approach, Albastra Publ. House, 2008			
2. M. Antal, C. Pop, D. Moldovan, T. Petrican, C. Stan, I. Salomie, T. Cioara, I. Anghel, Distributed Systems – Laboratory Guide, Editura UTPRESS Cluj-Napoca, 2018 ISBN 978-606-737-329-5, 2018, https://biblioteca.utcluj.ro/files/carti-online-cu-coperta/329-5.pdf			
3. Ioan Salomie, Tudor Cioara, Marcel Antal - Lecture Notes, Lab Notes Project Notes and Assignments http://www.coned.utcluj.ro/~salomie/DS_Lic			

*Se vor preciza, după caz: tematica seminariilor, lucrările de laborator, tematica și etapele proiectului.

9. Coroborarea conținuturilor disciplinei cu așteptările reprezentanților comunității epistemice, asociațiilor profesionale și angajatorilor reprezentativi din domeniul aferent programului

Este o disciplină a domeniului "Calculatoare și Tehnologia Informației". Ea îi instruește pe studenți în dezvoltarea și implementarea sistemelor software distribuite. Conținutul disciplinei a fost stabilit pe baza analizei disciplinelor echivalente de la alte universități precum și pe baza cerințelor angajatorilor IT din România. De asemenea conținutul disciplinei a fost evaluat de agenții guvernamentale românești (CNEAA și ARACIS).

10. Evaluare

Tip activitate	Criterii de evaluare	Metode de evaluare	Pondere din nota finală
Curs	Gradul de asimilare a cunoștiintelor despre sisteme distribuite predate in cadrul cursului	Examen scris (examen fata in fata sau online folosind Microsoft Teams)	55%
Seminar			
Laborator	-Capacitatea de a proiecta sisteme distribuite la nivel architectural si de componente utilizand principalele concepte si paradigme ale sistemelor distribuite precum si relatiile lor cu alte discipline din stiinta calculatoarelor -Capacitatea de a identifica principalele modele si tehnologii care pot fi folosite in proiectarea sistemelor distribuite fiind date un set de constrangeri -Prezență, Activitate	Evaluare lucrari laborator si proiect (examinare fata in fata sau online folosind Microsoft Teams) Unealta pentru integrare continua, deployment si testare a aplicatiilor distribuite	30% 15%
Proiect			
Standard minim de performanță: Sa poata proiecta si implementa sisteme software distribuite Calcul nota disciplina: 30% laborator + 15% proiect + 55% examen final Conditii de participare la examenul final: Laborator ≥ 5, Proiect ≥5 -predarea la timp a tuturor lucrarilor de laborator si minim nota 5 pe fiecare lucrare; prezenta la cel putin 11 lucrari de laborator Conditii de promovare: Examen final ≥ 5			

Data completării:	Titulari	Titlu Prenume NUME	Semnătura
	Aplicații	Prof.dr.ing. Tudor Cioara Conf.dr.ing. Ionut Anghel S.I.dr.ing. Cristina Pop S.I.dr.ing. Marcel Antal S.I.dr.ing.. Claudia Pop	
Data avizării în Consiliul Departamentului Calculatoare		Director Departament Prof.dr.ing. Rodica Potolea	
Data aprobării în Consiliul Facultății de Automatică și Calculatoare		Decan Prof.dr.ing. Liviu Miclea	