

UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU
FACULTATEA DE INGINERIE
DEPARTAMENTUL DE CALCULATOARE ȘI INGINERIE ELECTRICĂ

PROIECT DE DIPLOMĂ

Îndrumător : Conf. dr. mat. Crețulescu Radu

Absolvent: Stan Dragoș-Ciprian
Specializarea Ingineria Sistemelor Multimedia

- Sibiu, 2024

- UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU
- FACULTATEA DE INGINERIE
- DEPARTAMENTUL DE CALCULATOARE ȘI INGINERIE ELECTRICĂ

Platforma online pentru invatare programare

Îndrumător : Conf. dr. mat. Crețulescu Radu

Absolvent: Stan Dragoș-Ciprian
Specializarea Ingineria Sistemelor Multimedia

Contents

1	Introducere	5
1.1	Scop.....	5
1.2	Obiective	5
1.3	Motivație.....	5
2	Aspecte teoretice.....	6
2.1	Prezentarea mediului de dezvoltare.....	6
1.	NextJs.....	6
2.	Tailwind.....	8
3.	Flowbite,Flowbite-React	8
4.	Next-video.....	9
5.	React-pdf-vier	9
6.	Nestjs	9
7.	JWT Autentification	11
8.	Exceljs.....	11
9.	JavaScript	11
10.	TypeScript	11
11.	Nodejs	12
12.	Express.js.....	12
13.	Mongodb.....	13
	14
3	Arhitectura aplicatiei	15
3.1	User store:.....	15
3.2	User Requirements	16
3.3	Instalare	18
	NEXTJS:.....	18

NESTJS:	21
UseGuard nestjs:	24
3.4 Modele BD	25
4 Implemetarea practica	26
4.1.1 Modulul Creare (student/professor/admin).....	26
4.1.2 Modulul logarea (student/professor/admin)	27
4.1.3 Modulul creare curs	28
4.1.4 Modulul Video/Pdf/Exercitii	28
4.1.5 Modulul Compiler:	31
4.2 Implementarea modului...	32
4.2.1 Modulul Creare (student/professor/admin).....	32
4.2.2 Modulul logare (student/professor/admin)	41
4.2.3 Modulul creare curs	44
4.2.4 Modulul Video/Pdf/Exercitii	49
4.2.5 Modulul Exercices	66
5 Concluzii	81
5.1 Realizarea obiectivelor	82
6 Bibliografie	83
6.1 Webliografie.....	83

1 Introducere

1.1 Scop

Scopul principal a fost dezvoltarea unei platforme web complete, care să utilizeze cele mai recente tehnologii. Prin intermediul acestei platforme, îmi propun să ofer studenților și profesorilor un mediu în care pot învăța programarea într-un mod ușor și rapid.

1.2 Obiective

1. Crearea unei aplicații web fullstack cu MongoDB.
2. Crearea unui design simplu, rapid și previzibil (din design ar trebui să îți dai seama ce trebuie să faci).
3. Alegerea precisă a rolurilor (utilizator, student, profesor, administrator), iar dacă o pagină nu le este destinată, aceștia nu ar trebui să poată accesa.
4. Butoane speciale și pagini în funcție de roluri.
5. Pentru administrator, oferirea posibilității de a adăuga profesori, de a schimba cursurile între profesori și de a șterge profesori.
6. Pentru profesor, oferirea posibilității de a crea cursuri și de a adăuga în ele videoclipuri, PDF-uri și exerciții legate de programare.
7. Pentru profesor, oferirea posibilității de a face cursurile publice/privat, de a adăuga studenții la cursurile private și de a desemna colaboratori care să poată adăuga conținut în curs.
8. Pentru student, oferirea posibilității de a accesa întregul curs postat de profesor.

1.3 Motivație

Crearea și implementarea acestei platforme web de învățare a programării urmărește să democratizeze accesul la educație în domeniul tehnologiei, facilitând o învățare adaptabilă și accesibilă pentru o gamă largă de utilizatori. Prin integrarea tehnologiilor de ultimă oră și prin structurarea unui mediu user-friendly, platforma își propune să maximizeze eficiența procesului educațional și să stimuleze colaborarea dintre studenți și profesori

2 Aspecte teoretice

Aceasta aplicatie este o aplicatie fullstack web proiect care se focuseaza in principal pe 2 framework-uri Nextjs pentru partea de frontend si Nestjs pentru partea de backend. Nextjs este folosit in principal pentru ca contine parte de server-rendering ceea ce ne permite noua in principal, ca request-urile pe care le facem la catre backend sa fie mult mai securizate, si un alt motiv este pentru o experienta mult mai buna pentru utilizator pentru ca server side render va permite utilizatorului sa vada ca si cum ar urmarii o pagina cu HTML si CSS pur fara sa aiba in spate react hydratation. De asemenea pentru partea de frontend in proiect au fost folosite cateva framework-uri pentru CSS ca tailwind sau flowbite React pentru a fii mult mai simplu sa facem un design pentru UI. La partea de backend am folosit Nestjs in principiu pentru ca este usor de folosit si contine toate partiile de backend de care am nevoie.

2.1 Prezentarea mediului de dezvoltare

In aplicatie pentru a implementa partea de frontend vom avea nevoie de aceste tool-uri si framework-uri:

1. NextJs:

„Next.js este un framework bazat pe React destinat dezvoltării de aplicații web complexe. Acesta folosește componente React pentru a proiecta interfețe de utilizator și oferă funcționalități și optimizări adiționale prin Next.js.

În spatele scenei, Next.js identifică și setează în mod automat uneltele necesare pentru React, precum bundling-ul, compilarea și multe altele. Aceasta vă permite să vă focalizați pe dezvoltarea aplicației, economisind timpul petrecut cu configurarea.

Routing: Un router bazat pe sistemul de fișiere, construit pe componente de server, care suportă layout-uri, rutare încorporată, stări de încărcare, gestionarea erorilor și multe altele." [\[1\]](#)

Caracteristici principale:

- „Rendering: Redare pe partea de client și server cu componente de client și server. Optimizat suplimentar cu redare statică și dinamică pe server cu Next.js. Streaming pe runtime-urile Edge și Node.js.
- Data Fetching: Simplificarea extragerii datelor cu async/await în componente de server, și o extensie a API-ului fetch pentru memoizarea cererilor, cache-ul datelor și revalidarea.

- Stilizare: Suport pentru metodele preferate de stilizare, inclusiv module CSS, Tailwind CSS și CSS-in-JS.
- Optimizations: Optimizări pentru imagini, fonturi și scripturi pentru a îmbunătăți indicatorii esențiali ai web-ului și experiența utilizatorului aplicației.
- TypeScript: Suport îmbunătățit pentru TypeScript, cu o verificare mai bună a tipurilor și o compilare mai eficientă, precum și un plugin TypeScript personalizat și un verificator de tip.”^[2]

Routing în Next.js

„În Next.js, sistemul de routing este un aspect esențial pentru construirea unei aplicații eficiente și bine structurate. Documentația oficială a Next.js explică detaliat cum funcționează routing-ul și cum să-l implementăm în proiectele noastre.”^[3]

- Structura Fisierelor și Routing-ul Automat:

„Next.js folosește un sistem de routing bazat pe fișiere. Asta înseamnă că structura fișierelor din directorul pages reflectă automat rutele aplicației tale. De exemplu, un fișier pages/about.js va crea o rută /about.”^[4]

- Dynamic Routing:

„Next.js permite crearea rutelor dinamice prin folosirea sintaxei colilor pătrate în numele fișierelor. De exemplu, un fișier pages/posts/[id].js va crea o rută dinamică unde id poate fi orice valoare (e.g., /posts/1, /posts/hello-world).”^[5]

- API Routes:

„În Next.js, putem crea API routes folosind directorul pages/api. Aceste rute sunt funcții serverless care rulează pe serverul Next.js, permițându-ne să construim API-uri direct în aplicația noastră Next.js.”^[6]

Server Rendering și Client Rendering în Next.js

În Next.js, gestionarea datelor și modalitatea de randare sunt aspecte esențiale pentru performanța și experiența utilizatorilor. Documentația oficială a Next.js explică diferențele și utilizările corecte ale Server Rendering și Client Rendering.

- Server Rendering (SSR):

„Server Rendering, sau Server-Side Rendering (SSR), implică generarea conținutului paginilor pe server, înainte ca acesta să fie trimis către client. Acest lucru înseamnă că utilizatorul primește conținutul deja randat, îmbunătățind performanța inițială și SEO.”^[7]

- Client Rendering (CSR):

„Client Rendering, sau Client-Side Rendering (CSR), implică randarea conținutului pe client, adică în browser-ul utilizatorului. Acest mod este util pentru interacțiuni dinamice și aplicații unde experiența utilizatorului se schimbă frecvent fără a necesita reîncărcarea paginii. În Next.js, CSR se poate realiza folosind hook-uri precum `useEffect` pentru a prelua date după ce componenta a fost montată.”^[8]

2. **Tailwind**

„Tailwind CSS este un utilitar de primă clasă pentru dezvoltarea interfețelor de utilizator. Acesta permite dezvoltatorilor să creeze design-uri personalizate fără a părăsi marcajul HTML. În loc să aibă stiluri predefinite și o mulțime de componente preconstruite, Tailwind permite dezvoltatorilor să construiască rapid interfețe de utilizator complexe din componente mici și reutilizabile. În plus, Tailwind CSS oferă un set de utilități de nivel înalt care permit dezvoltatorilor să creeze rapid interfețe de utilizator responsive și interactive. Acesta include utilități pentru poziționare, spațiere, tipografie și culori, precum și utilități pentru flexbox, grilă și alte tehnologii CSS moderne. În concluzie, Tailwind CSS este un instrument puternic pentru dezvoltatorii care doresc să creeze interfețe de utilizator personalizate și responsive cu un efort minim. Este rapid, flexibil și fiabil, oferind un timp de rulare zero și o experiență de dezvoltare fluidă.”^[1]

3. **Flowbite,Flowbite-React**

„Flowbite este o bibliotecă open-source de componente UI, construită pe baza framework-ului CSS Tailwind, un utilitar de top, care include suport pentru tema întunecată, un sistem de design Figma, template-uri și multe altele. Aceasta cuprinde toate componentele frecvent utilizate necesare unui site web, precum butoane, meniuri derulante, bare de navigare, ferestre modale, precum și unele elemente interactive mai sofisticate, cum ar fi selectorii de date. Toate aceste elemente sunt create utilizând clasele de utilitate din Tailwind CSS și

JavaScript pur, cu suport pentru TypeScript. În plus, Flowbite oferă o integrare strânsă cu Tailwind CSS, permițând dezvoltatorilor să creeze interfețe de utilizator personalizate și responsive cu un efort minim. Acesta este un instrument puternic pentru dezvoltatorii care doresc să creeze interfețe de utilizator personalizate și responsive cu un efort minim. Este rapid, flexibil și fiabil, oferind un timp de rulare zero și o experiență de dezvoltare fluidă.”^[1]

4. Next-video

„Este un modul react care permite integrarea clipurilor video în aplicații Next.js. Acesta îmbunătățește atât elementul <video>, cât și aplicația Next prin adăugarea de funcții pentru optimizarea automată a conținutului video.”^[1]

5. React-pdf-viewer

„React-pdf-viewer este o componentă esențială în dezvoltarea aplicațiilor web cu React care necesită vizualizarea fișierelor PDF. Aceasta permite utilizatorilor să încarce și să vizualizeze documente PDF direct în aplicația lor React, oferind o experiență de utilizator intuitivă și fluidă.

Redare de înaltă calitate: React-pdf-viewer asigură o redare clară și precisă a documentelor PDF, păstrând toate detaliile grafice și textuale.

Interfață de utilizare intuitivă: Componentele oferite de react-pdf-viewer sunt ușor de integrat și personalizat, permițând dezvoltatorilor să creeze interfețe adaptate nevoilor utilizatorilor finali.

Navigare simplă: Utilizatorii pot naviga prin paginile documentului PDF utilizând butoane de navigare sau scroll, ceea ce face vizualizarea și citirea documentelor mult mai accesibilă.

Funcționalități avansate: Include funcții precum zoom, rotație și descărcare, oferind o experiență de vizualizare completă și flexibilă.”^[1]

În aplicație pentru a implementa partea de backend vom avea nevoie de aceste tool-uri și framework-uri:

6. Nestjs:

„Nest este un framework destinat creării de aplicații Node.js eficiente și scalabile pe partea de server. Acesta utilizează JavaScript avansat, este construit și suportă integral TypeScript (deși permite dezvoltatorilor să scrie în JavaScript pur) și îmbină elemente de

POO (Programare Orientată pe Obiecte), PF (Programare Funcțională) și PFR (Programare Funcțională Reactivă).

În spatele scenei, Nest utilizează cadre de server HTTP solide, precum Express (implicit) și poate fi configurat opțional să utilizeze Fastify!

Nest oferă o abstracție peste aceste cadre comune Node.js (Express/Fastify), dar, de asemenea, le face disponibile direct dezvoltatorilor. Acest lucru le oferă dezvoltatorilor libertatea de a folosi numeroasele module terțe care sunt disponibile pentru platforma de bază.”[\[1\]](#)

Controlere

„În NestJS, controlerele sunt esențiale pentru gestionarea cererilor și răspunsurilor HTTP, fiind punctul central prin care se interacționează cu aplicația.

Controlerele în NestJS sunt responsabile pentru gestionarea cererilor primite și returnarea răspunsurilor către client. Fiecare metodă dintr-un controller corespunde unui endpoint specific al aplicației și gestionează cereri HTTP pentru acea rută.

Controlerele sunt decorate cu decoratori specifici, cum ar fi `@Controller()`, `@Get()`, `@Post()`, `@Put()`, `@Delete()`, care specifică tipul cererii și ruta asociată.

Un controller se creează folosind clasa `@Controller()` din NestJS. Metodele din această clasă sunt decorate cu decoratori corespunzători pentru a specifica tipul de cerere și ruta.”[\[1\]](#)

Module

„Modulele în NestJS sunt folosite pentru a organiza aplicația în blocuri logice și pentru a gestiona eficient dependențele între aceste blocuri. Fiecare aplicație NestJS are cel puțin un modul principal, denumit `AppModule`. Modulele sunt definite folosind decoratorul `@Module()` și conțin metadata care descriu componentele, serviciile și alte module pe care le utilizează.

Modulele ajută la organizarea aplicației în segmente distincte și independente. Fiecare modul poate importa alte module pentru a utiliza funcționalitățile oferite de acestea. Acest lucru facilitează reutilizarea codului și separarea preocupărilor.”[\[1\]](#)

Guard-uri

„Guard-urile în NestJS sunt clase care implementează interfața CanActivate și sunt folosite pentru a determina dacă o cerere poate continua sau nu pe baza anumitor criterii. Ele sunt executate înainte ca un handler de rută să fie apelat, permițând astfel controlul accesului la resursele protejate.”^[1]

7. JWT Authentication

„Oferiți utilizatorilor posibilitatea de a se autentifica folosind un nume de utilizator și o parolă, generând un JWT pentru utilizare în solicitările ulterioare către endpoint-urile API securizate. Suntem în procesul de a îndeplini această cerință. Pentru a o finaliza, va trebui să dezvoltăm codul care generează un JWT. Implementați rute API care sunt securizate prin verificarea prezenței unui JWT valid ca token bearer.

În plus, în cadrul framework-ului NestJS, JWT (JSON Web Token) este adesea utilizat pentru a implementa autentificarea. NestJS oferă un modul JWT care simplifică procesul de implementare a autentificării bazate pe JWT. Acest modul permite generarea de token-uri JWT, validarea lor și protejarea rutelor pe baza acestora. De asemenea, NestJS suportă strategii de autentificare personalizate, ceea ce înseamnă că puteți configura modul în care aplicația dvs. gestionează autentificarea și autorizarea.”^[1]

Aceas logica si la student/professor

8. Exceljs

„Ofera posibilitatea de a citi/scrie dintr-un excel folosind javascripts.”^[1]

9. JavaScript

„JavaScript este un limbaj de programare flexibil, interpretat sau compilat just-in-time, care oferă funcții de primă clasă. Deși este renumit în principal ca limbaj de scripting pentru paginile web, este utilizat și în alte medii, precum Node.js, Apache CouchDB și Adobe Acrobat. Este un limbaj dinamic, bazat pe prototipuri, care acceptă mai multe paradigme, funcționează pe un singur fir de execuție și sprijină stilurile de programare orientate pe obiecte, imperative și declarative, inclusiv programarea funcțională.”^[1]

10. TypeScript

„TypeScript este un limbaj de programare orientat pe obiecte, cu sursă deschisă, dezvoltat și susținut de Microsoft Corporation. Lansat în 2012, a devenit din ce în ce mai popular în comunitatea dezvoltatorilor. Este un super set strict al limbajului JavaScript,

ceea ce înseamnă că orice se poate implementa în JavaScript poate fi realizat și în TypeScript, beneficiind în plus de funcționalități îmbunătățite (orice cod JavaScript valid este și cod TypeScript valid). Codul TypeScript se transformă în cod JavaScript, permițând integrarea ușoară în proiectele JavaScript. Acesta este conceput în principal pentru proiecte de anvergură.”^[1]

11. Nodejs

„Node.js este un mediu de execuție JavaScript extensiv, cu sursă deschisă, care funcționează pe multiple platforme. Este ales frecvent pentru dezvoltarea unei varietăți largi de aplicații, de la servere web până la utilitare de linie de comandă și aplicații de rețea. Una dintre principalele caracteristici ale Node.js este utilizarea motorului JavaScript V8, nucleul Google Chrome, care rulează codul JavaScript direct pe hardware-ul calculatorului fără intermediari, oferind astfel performanțe de top. Acest aspect face din Node.js o soluție ideală pentru procesarea de mare viteză și sarcini intensiv computaționale. Node.js operează pe un model asincron, bazat pe evenimente. Fiecare aplicație Node.js rulează într-un singur proces, gestionând multiple cereri de la utilizatori printr-un model non-blocant. Acesta nu creează un nou fir pentru fiecare cerere, ci utilizează un sistem de callback-uri pentru operațiunile I/O, precum citirea din fișiere sau cererile de rețea, care ar putea altfel să încetinească execuția. Biblioteca standard a Node.js oferă funcții asincrone I/O care ajută la evitarea blocării codului JavaScript. Majoritatea bibliotecilor Node.js sunt construite pe aceste paradigme non-blocante, făcând comportamentul blocant o excepție, nu o regulă. Acest lucru îmbunătățește drastic eficiența și timpul de răspuns al aplicațiilor. Ecosistemul Node.js este unul dintre cele mai active, cu peste un milion de pachete disponibile în registrul npm, care oferă soluții pentru aproape orice problemă de dezvoltare software.

Aplicații web și API-uri: Datorită naturii sale asincrone, Node.js este ideal pentru construirea de aplicații web performante și API-uri RESTful.

Aplicații de o singură pagină (SPA): Servere pentru clienți Angular, React sau Vue.js.

Scripturi de automatizare și utilitare: Perfect pentru dezvoltarea unor scripturi care automatizează diverse sarcini.

Aplicații în timp real: Chat-uri și jocuri online beneficiază de capacitatea Node.js de a procesa evenimente în timp real.”^[1]

12. Express.js

„Express.js este un framework web rapid, flexibil și minimalist pentru Node.js. Acesta simplifică în mod eficient dezvoltarea aplicațiilor web și a API-urilor folosind JavaScript pe partea de server. Fiind open-source, Express este dezvoltat și susținut de fundația Node.js. Express.js pune la dispoziție un set puternic de funcționalități care îți sporesc productivitatea și îți eficientizează aplicația web. Facilitează organizarea funcționalităților aplicației cu ajutorul middleware-ului și al rutării, adaugă utilitare utile obiectelor HTTP din Node și ușurează generarea dinamică a obiectelor HTTP.”^[1]

13. **Mongodb**

„MongoDB, cea mai populară bază de date NoSQL, este o bază de date orientată pe documente, cu sursă deschisă. Termenul „NoSQL” înseamnă „non-relațional”, ceea ce indică faptul că MongoDB nu se bazează pe structura de baze de date relaționale, ci utilizează un mecanism complet diferit pentru stocarea și accesarea datelor. Acest format de stocare se numește BSON (asemănător formatului JSON).”^[1]

3 Arhitectura aplicatiei

3.1 User store:

1. Vizitatori
 - a. Vizitatorii pot vizualiza cursurile (titlul și descrierea), însă nu au acces la materialele profesorilor.
 - b. Dacă doresc să se alăture unui curs, sunt redirecționați către pagina de înregistrare.
2. Înregistrare/Autentificare
 - a. Vizitatorii pot să se înregistreze pe platformă sau să se autentifice.
 - b. Datele lor sunt trimise către baza de date, iar cele sensibile, precum parola, sunt criptate.
3. Studenți
 - a. Studenții au acces la aceleași informații ca vizitatorii, dar pot participa la cursuri prin apăsarea butonului "Alătură-te".
 - b. ID-ul studentului este salvat într-un câmp special din baza de date, ceea ce le permite accesul la materialele profesorilor.
4. Acces la Cursuri
 - a. Studenții pot viziona tutorialele video sau citi PDF-urile aferente cursului.
 - b. De asemenea, au acces la o secțiune de exerciții care conține cerințe, exemple și un compilator ce suportă diverse limbaje de programare.
 - c. Dacă algoritmul rezolvă corect problema, codul se salvează în baza de date.
5. Editare Profil
 - a. Studenții, profesorii și administratorii își pot modifica numele de utilizator, adresa de email, parola sau poza de profil.
 - b. Aceste modificări actualizează baza de date.
6. Crearea Cursurilor
 - a. Profesorii pot crea cursuri, adăugând un titlu, o descriere, un banner și setând cursul ca public sau privat.
 - b. Aceste informații creează un nou câmp în baza de date.
7. Adăugarea Materialelor
 - a. Profesorii pot încărca tutoriale video, PDF-uri și exerciții.

- b. Informațiile se salvează în baza de date, iar materialele video și PDF-urile sunt stocate pe disc în backend.
- 8. Exerciții
 - a. Profesorii pot adăuga exerciții, furnizând cerințele, exemple și funcții cu parametrii necesari.
 - b. Inputurile și outputurile validate de profesori se verifică automat pentru soluțiile oferite de studenți.
- 9. Adăugarea Studenților
 - a. Profesorii pot adăuga studenți înscriși pe platformă, chiar și la cursuri private.
- 10. Colaboratori
 - a. Profesorii pot adăuga alți profesori ca și colaboratori la un curs, oferindu-le aceleași permisiuni.
- 11. Operațiuni CRUD
 - a. Profesorii au permisiunea de a efectua operațiuni CRUD în baza de date pentru cursurile proprii.
- 12. Administratori - Adăugarea Profesorilor
 - a. Administratorii pot adăuga profesori prin încărcarea unui fișier Excel care conține datele acestora.
 - b. Backend-ul procesează fișierul și încarcă datele în baza de date.
- 13. Administratori - Operațiuni CRUD
 - a. Administratorii au dreptul de a efectua operațiuni CRUD în baza de date.

3.2 User Requirements

- I. Înregistrare
 - 1. Admin:
 - a. Oricine dorește să devină administrator trebuie să fie adăugat direct în baza de date.
 - b. Înregistrarea se face doar pe calculatorul care are acces la baza de date, prin crearea unui obiect JSON cu datele specifice ale administratorului.
 - c. Datele din JSON trebuie introduse manual în baza de date, iar procesul ar trebui să includă validări pentru a preveni intrările duplicate sau erorile de format.

- d. Acest proces ar putea fi gestionat de către un utilizator cu permisiuni speciale.

2. Profesor:

- a. Cei care doresc să fie profesori trebuie să facă o cerere către un administrator.
- b. Administratorul este singurul care poate aproba și adăuga profesori în baza de date.
- c. Solicitarea trebuie să conțină numele și adresa de e-mail a profesorului, trimise prin e-mail către administrator.
- d. Administratorul ar trebui să verifice manual detaliile și să aprobe sau să respingă cererea.

3. Student:

- a. Studenții se pot înregistra direct pe site, completând câmpurile obligatorii: numele, e-mail-ul și parola.
- b. La crearea profilului, se generează automat o imagine implicită a utilizatorului, dar aceasta poate fi schimbată ulterior.

II. Autentificare

- 1. Formularele de autentificare trebuie să conțină câmpuri pentru e-mail și parolă.
- 2. După introducerea datelor, acestea trebuie verificate în baza de date pentru confirmarea rolului (admin, profesor, student).
- 3. Sistemul va identifica gradul utilizatorului și va trimite către browser un id cryptat care va reprezenta user-ul (student,professor,admin).

III. Creare curs:

- 1. Doar profesorii pot crea cursuri.
- 2. În interfață, profesorii trebuie să poată adăuga numele cursului, descrierea(maxim de 250 de cuvinte) și să definească nivelul de vizibilitate (public/privat).

IV. Adăugare curs:

1. După introducerea datelor generale, profesorul poate adăuga materiale de curs în format video, PDF sau exerciții
2. Video:
 - a. Profesorul poate adăuga un titlu, o descriere și un fișier video.
 - b. Fișierul video trebuie să fie de format MP4 și să aibă mărimea maximă de 50 MB
 - c. După încărcare, se redirecționează către o pagină de previzualizare, unde poate verifica materialul adăugat.
3. PDF:
 - a. Se poate încărca un fișier PDF, împreună cu un titlu și un fișier pdf
 - b. Fișierul pdf poate să aibă dimensiunea maximă 50 MB
 - c. După încărcare, profesorul poate previzualiza documentul pentru a se asigura că este corect.
4. Exerciții:
 - a. Profesorii pot introduce numele problemei, cerințele, exemplele și tipul de funcție care trebuie folosit, numele funcției, parametri funcției dacă este cazul, și un fișier excel care să conțină input-ul și output-ul algoritmului din cerință.
 - b. În excelul respectiv, într-o coloană pe primul rând va reprezenta output-ul algoritmului (în care output-ul poate fi un număr, string, vector sau char) iar rezultatele de rânduri va reprezenta inputul funcției.
 - c. După ce sau introdus toate datele în platformă, profesorul va fi redirecționat către pagina de vizualizare, unde poate testa algoritmul cu limbajele de programare introduse de platformă.

3.3 Instalare

NEXTJS:

Această aplicație a fost creată cu ultima versiune de nextjs care a fost pe atunci nextjs 13 iar dacă vrem să instalăm nextjs cu orice versiune nouă care apare trebuie să scriem în

powershell aceasta instrucțiune:

```
npx create-next-app@13.5.4
```

Selectam tailwind si typescript.

Si instalam aceste biblioteci care ne vor trebuie pe parcursul proiectului:

```
npm install next-client-cookies
```

```
npm install next-images
```

```
npm install cookies-next
```

```
npm install @react-pdf-viewer/core
```

```
npm install canvas
```

```
npm install flowbite
```

```
npm install flowbite-react
```

```
npm install xlsx
```

- Pentru că Next.js conține componenta de server rendering, va trebui să creăm un director numit api în care vom crea fișiere care vor conține toate un fișier route. Acest fișier route va reprezenta cererile către backend. Facem acest lucru pentru a evita problemele de cross-origin protection.
- Iar ca aplicatia sa fie complete in next.config.js trebuie sa scriem aceste initializarii:

```
/** @type {import('next').NextConfig} */
const nextConfig = {
  images: {
    domains: ['localhost', 'img.freepik.com', 'veterinaire-tour-
hassan.com'],},
  webpack: (config) => {
    config.module.rules.push({test: /\.node/,use: "raw-loader",});
    return config;},
  async headers() {
    return [{source: "/(.*)",
      headers: [
        { key: "Access-Control-Allow-Credentials", value: "true" },
        { key: "Access-Control-Allow-Origin", value: "http://localhost:3000"
      },
      { key: "Access-Control-Allow-Methods", value:
"GET,DELETE,PATCH,POST,PUT" },
      { key: "Access-Control-Allow-Headers", value: "X-CSRF-Token, X-
Requested-With, Accept, Accept-Version, Content-Length, Content-MD5, Content-
```

```
Type, Date, X-API-Version" },    ]    }]  }}
```

```
module.exports = nextConfig
```

In tsconfig.json trebuie sa scriem asta:

```
{
  "compilerOptions": {
    "target": "es5",
    // "lib": ["dom", "dom.iterable", "esnext", "scripthost", "ES6"],
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "Node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "forceConsistentCasingInFileNames": true,
    "jsx": "preserve",
    "incremental": true,
    "plugins": [
      { "name": "next" }
    ],
    "paths": { "@/*": [ "./src/*" ] },
    "include": ["next-env.d.ts", "**/*.ts", "**/*.tsx", ".next/types/**/*.ts"],
    "exclude": ["node_modules"]
  }
}
```

Pentru a putea rula tailwind in tailwind.config.ts trebuie sa scriem asta:

```
import type { Config } from "tailwindcss";
const config: Config = {
  content: [
    "./node_modules/flowbite-react/lib/**/*.js",
    "./src/pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./src/components/**/*.{js,ts,jsx,tsx,mdx}",
    "./src/app/**/*.{js,ts,jsx,tsx,mdx}",
    "./pages/**/*.{ts,tsx}",
    "./public/**/*.html",
  ],
}
```

```

theme: {
  extend: {
    scale: {
      '103': '1.03',
    },
    screens:{
      'desktop': '1280px',},
    backgroundImage: {
      "gradient-radial": "radial-gradient(var(--tw-gradient-stops))",
      "gradient-conic":
        "conic-gradient(from 180deg at 50% 50%, var(--tw-gradient-stops))",
    },
    colors: {
      "navbar-color": "hsl(215, 57%, 28%)",    },},},
  plugins: [
    require("flowbite/plugin")],};
export default config;

```

NESTJS:

```

npm i -g @nestjs/cli
nest new project-name
npm install --save @nestjs/jwt
npm i @nestjs/mongoose mongoose
npm i --save class-validator class-transformer
npm i cookie-parser
npm i -D @types/cookie-parser
npm install exceljs
npm install multer
npm install rxjs
npm install ts-transformer-keys
In fisierul tsconfig.build.json trebuie sa scriem asta:

```

```

{
  "extends": "./tsconfig.json",
  "exclude": ["node_modules", "test", "dist", "**/*spec.ts"]
}

```

In fisierul nest-cli.json trebuie sa scriem asta:

```

{
  "$schema": "https://json.schemastore.org/nest-cli",
  "collection": "@nestjs/schematics",

```

```

    "sourceRoot": "src",
    "compilerOptions": {
      "deleteOutDir": true
    }
  }
}

```

In fisierul package.json trebuie sa updatam "scripts":

```

"scripts": {
  "build": "nest build",
  "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
  "start": "nest start",
  "start:dev": "nest start --watch",
  "start:debug": "nest start --debug --watch",
  "start:prod": "node dist/src/main.js",
  "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
  "test": "jest",
  "test:watch": "jest --watch",
  "test:cov": "jest --coverage",
  "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-
node/register node_modules/.bin/jest --runInBand",
  "test:e2e": "jest --config ./test/jest-e2e.json"
},

```

In fisierul tsconfig.json trebuie sa scriem asta:

```

{
  "compilerOptions": {
    "module": "commonjs",
    "declaration": true,
    "removeComments": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "allowSyntheticDefaultImports": true,
    "target": "ES2021",
    "sourceMap": true,
    "outDir": "./dist",
    "baseUrl": "./",
    "incremental": true,
    "skipLibCheck": true,
    "strictNullChecks": false,
    "noImplicitAny": false,

```

```

    "strictBindCallApply": false,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": false,
    "strict": true}}

```

In fisierul main.js trebuie sa scriem asta:

```

import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import * as cookieParser from 'cookie-parser';
import * as bodyParser from 'body-parser';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  const options = {
    origin: 'http://localhost:3001',
    methods: 'GET,HEAD,PUT,PATCH,POST,DELETE,OPTIONS',
    preflightContinue: false,
    optionsSuccessStatus: 204,
    credentials: true,
    allowedHeaders: '*',});
  app.enableCors(options);
  app.use(cookieParser());
  app.use(bodyParser.json({ limit: '50mb' }));
  app.use(bodyParser.urlencoded({ limit: '50mb', extended: true }));
  await app.listen(3000);}
bootstrap();

```

Pentru fiecare modul pe care in implementem trebuie sa inseram data de baze mongodb .

Exemplu:

```

@Module({
  imports: [
    MongooseModule.forFeature([ { name: 'Admin', schema: AdminSchema } ]),
    JwtModule.register({
      secret: 'your-secret-key',
      signOptions: { expiresIn: '24h' },
    }),
    ProfessorModule,],
  controllers: [AdminsController],
  providers: [AdminsService],
  exports: [AdminsService],})

```

Si in nestjsx daca vrem sa facem requesturii in baza de date in timp real trebuie introducem aceasta linie de cod in code in services:

```
onModuleInit() {  
  this.adminModel.watch().on('change', (change) => {  
    console.log(change);  
  });  
}
```

UseGuard nestjs:

```
import {  
  CanActivate,  
  ExecutionContext,  
  Injectable,  
  NotFoundException,  
} from '@nestjs/common';  
import { JwtService } from '@nestjs/jwt';  
import { AdminsService } from 'src/Schemas/Use-case/admins/admins.service';  
@Injectable()  
export class AdminGuard implements CanActivate {  
  constructor(private readonly jwtService: JwtService,  
    private readonly studentService: AdminsService,) {}  
  async canActivate(context: ExecutionContext): Promise<boolean> {  
    const request = context.switchToHttp().getRequest();  
    try {const id = request.cookies['id'];  
      const decodedToken = this.jwtService.verify(id);  
      const admin = await this.studentService.getAdmin(decodedToken.sub);  
      if (admin?.role === 'admin' && admin.role !== null) {  
        return true;  
      } else {return false;}} catch (error) {  
        throw new NotFoundException('Unauthorized access');}}}
```

Compiler:

Python: [Download Python | Python.org](#)

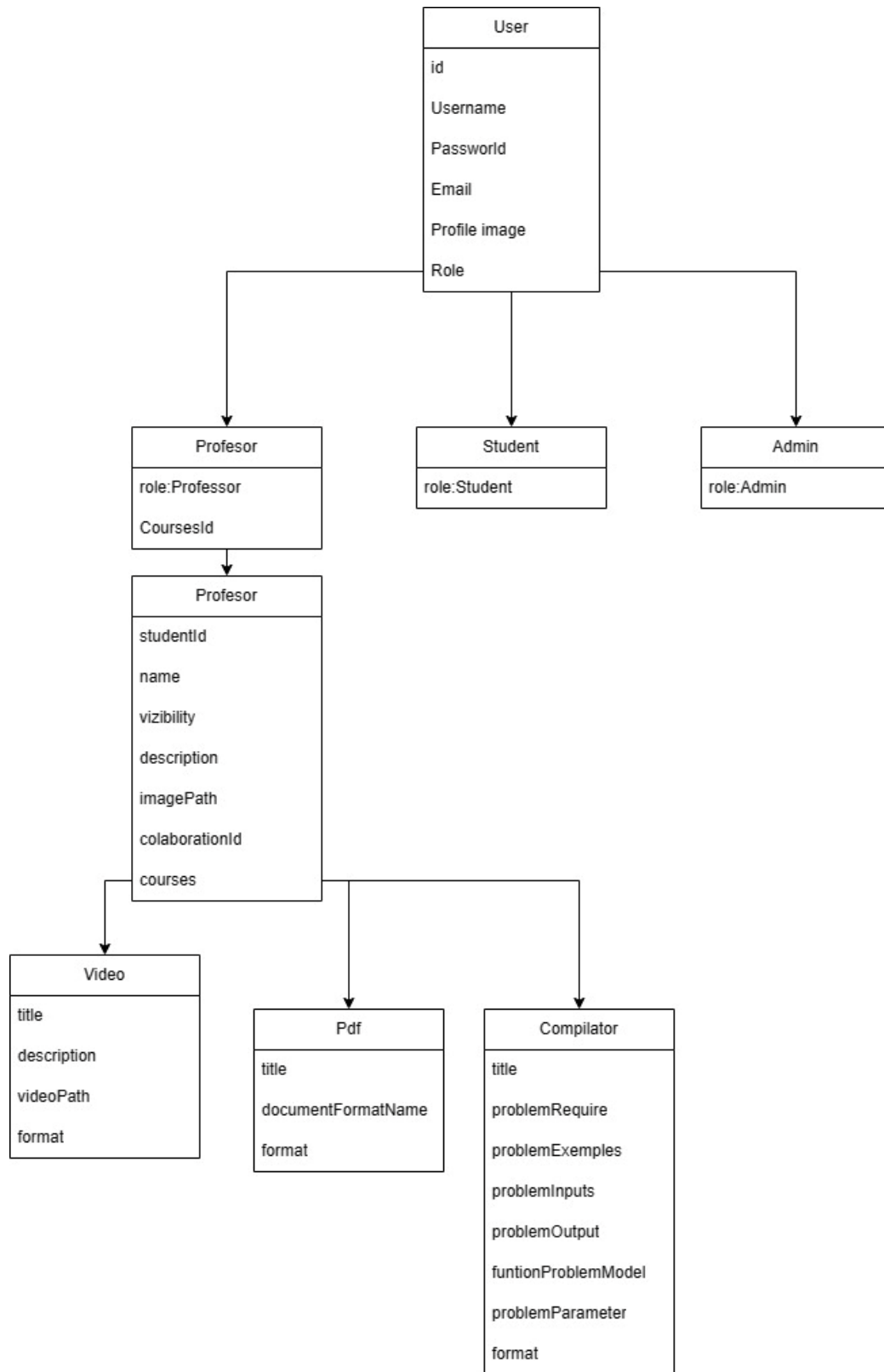
JavaScript/nodejs: [Node.js — Download Node.js® \(nodejs.org\)](#)

C++: [GCC, the GNU Compiler Collection - GNU Project](#)

Java: [Download Java for Windows](#)

JDK: [Java Downloads | Oracle](#)

3.4 Modele BD



4 Implemetarea practica

4.1.1 Modulul Creare (student/professor/admin)

În platformă există trei tipuri de roluri principale: student, profesor și admin. Fiecare rol se creează diferit, dar există un lucru comun: studentul/profesorul/adminul trebuie să introducă aceste date (nume, email, parolă) indiferent de modul în care sunt creați. De asemenea, emailul trebuie să fie unic, altfel nu se va putea crea acel utilizator, indiferent de rol.:

- Studentul se creează punându-și datele în platformă (nume, email, parolă).
- Profesorul se creează de către admin, prin inserarea în platformă a unui fișier Excel care conține datele necesare aferente profesorilor respectivi.
- Adminul se creează direct în baza de date prin inserarea unui fișier JSON cu datele aferente adminului.

Explicație cod:

a. Student

În frontend, studentul se poate înregistra pe platformă introducând numele, emailul și parola în câmpurile de text existente în interfața utilizatorului (UI). Când studentul apasă pe butonul de înregistrare, se va apela clasa `userAuthenticationManager`. Această clasă are rolul, în primul rând, de a verifica dacă studentul respectă formatul de email, adică dacă emailul conține numele, simbolul `@` și un punct. Acest lucru se va realiza cu ajutorul unei expresii regulate care verifică pașii menționați mai sus și, dacă nu, se va anunța UI-ul că emailul este incorect. Dacă formatul emailului este corect, se vor trimite cereri către backend pentru a verifica dacă emailul există în colecțiile de studenți, profesori sau administratori și, dacă există, se va anunța frontend-ul că emailul există și acest lucru va apărea în UI. Dacă totul este în regulă, se vor trimite către backend toate datele studentului, iar acesta va fi înregistrat în baza de date cu datele introduse. Odată ce datele au fost introduse în baza de date, în backend se va cripta ID-ul studentului cu componenta JWT și se va trimite către frontend, unde frontend-ul va lua cheia respectivă și o va pune într-un cookie pentru a fi folosită pentru restul paginilor. Studentul va fi apoi redirecționat către pagina

principală, unde va avea posibilitatea să se înscrie la cursuri.

b. Profesor

Profesorii pot fi adăugați doar de către admin. Pentru a adăuga profesori, adminul trebuie să intre pe pagina /adminWorkspace/newProfessor, unde va avea un câmp în care poate încărca un fișier Excel. În acest fișier Excel, prima coloană va reprezenta numele profesorului, emailul și parola. În backend, se va verifica dacă emailul este unic, folosind aceeași logică ca la student.

Exemplu de fișier Excel:

	A	B	C	D
1	username	email	password	
2	John2	john8975@university.edu	1234	
3	John	john@university.edu	1234	
4				
5				
6				

Când fișierul Excel este trimis în backend, se va folosi clasa Excel.Workbook.

Apoi, se va lua fiecare linie din fișierul Excel, emailul va fi transformat din hyperlink în text normal și, dacă totul este în regulă, datele vor fi adăugate în baza de date.

c. Admin

Adminul se va crea doar prin încărcarea unui fișier JSON în baza de date sau prin scrierea de comenzi MongoDB. Datele adminului vor fi, în principiu, datele unui utilizator, dar la rol va fi specificat "admin".

4.1.2 Modulul logarea (student/professor/admin)

Pentru logare există o singură pagină unde orice utilizator (student, profesor sau admin) se poate conecta, iar platforma va recunoaște, în primul rând, după email, ce rol are utilizatorul logat. Acest lucru se face prin trimiterea de cereri către colecțiile de studenți, profesori și administratori și verificarea dacă utilizatorul există. De fiecare dată când se loghează un utilizator, acesta va primi o cheie criptată cu biblioteca JWT din NestJS (unde cheia conține ID-ul persoanei). Când utilizatorul dorește să facă anumite cereri specifice rolului pe care îl are, se va verifica cheia respectivă și, dacă nu se potrivește cu identitatea sa, nu va putea face acea cerere.

Explicatie Cod:

În frontend, toți utilizatorii (student, profesor, admin) au aceeași interfață utilizator (UI) în care vor fi nevoiți să introducă emailul și parola și să apese pe butonul de logare. Când se apasă pe buton, se va apela clasa `userAuthenticationManager`, unde, în primul rând, se va verifica dacă emailul este corect. În această clasă există funcția `verifyUser`, în care se vor lua datele introduse de utilizator în UI. Se vor trimite pe rând cereri către backend, urmărindu-se ce tip de utilizator este (student, profesor, admin), și, dacă utilizatorul este găsit, se va trimite cheia criptată; dacă nu, se va trimite un string gol. Dacă pentru cele trei tipuri de utilizatori se returnează un string gol, înseamnă că utilizatorul nu există.

4.1.3 Modulul creare curs

Orice profesor care a fost adăugat de către admin va avea generat un buton care îi va permite să acceseze o pagină unde va putea să creeze oricâte cursuri dorește. Pe pagina `/professorworkspace` unde va avea un câmp în care va putea să introducă numele cursului, să decidă dacă vrea să facă cursul public sau privat și să adauge o scurtă descriere a cursului, de maxim 250 de caractere. După ce profesorul a încărcat cursul, el va putea să decidă ce materiale dorește să încarce pentru curs.

În curs există trei tipuri de materiale pe care le poate încărca profesorul: Video, PDF și Exerciții.

Explicatie cod:

După ce profesorul s-a logat, are posibilitatea să creeze cursuri. Acest lucru se face accesând pagina dedicată, unde va trebui să completeze toate câmpurile cu specificațiile menționate mai sus. Dacă nu completează aceste câmpuri, UI-ul va afișa un mesaj de avertizare. Dacă totul este în regulă, datele vor fi trimise către backend și vor fi încărcate în baza de date, împreună cu datele profesorului. Cursurile vor fi încărcate într-o colecție numită „courses”, iar când cursul a fost adăugat, în câmpul `courseId` al profesorului se va încărca ID-ul noului curs adăugat. Dacă totul este în regulă, profesorul va putea să adauge în curs materiale de tip video, PDF sau exerciții.

4.1.4 Modulul Video/Pdf/Exercitii

a. Video

La secțiunea video, profesorul trebuie să adauge un titlu pentru video, o descriere care poate să conțină oricâte caractere dorește și video-ul efectiv, care trebuie să

fie în format mp4 și să aibă dimensiunea maximă de 50 MB. Dacă aceste condiții nu sunt îndeplinite, va apărea un mesaj de eroare pe ecran.

b. Pdf

La secțiunea PDF, profesorul trebuie să adauge un titlu pentru PDF și fișierul PDF efectiv, care poate să aibă dimensiunea maximă de 50 MB. Dacă aceste condiții nu sunt îndeplinite, va apărea un mesaj de eroare pe ecran.

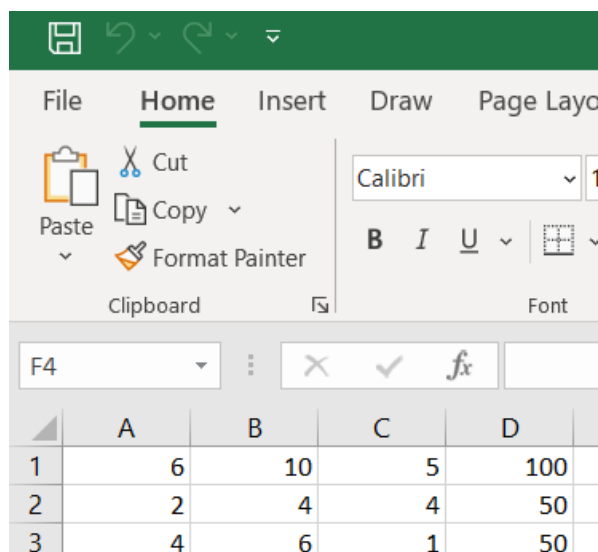
c. Exerciții

La secțiunea Exerciții, profesorul trebuie să adauge un titlu pentru exercițiu, cerința problemei, câte exemple dorește despre cum ar trebui să arate algoritmul și, foarte important, câteva date legate de problemă. Pentru ca compilatorul să funcționeze, va trebui să trimitem către backend un model despre cum dorim să arate funcția noastră, astfel încât să fie executată de limbajele de programare puse la dispoziție de platformă: Python, C++, JavaScript/Node.js și Java.

Primul lucru pe care trebuie să-l facem este să dăm un nume funcției, fără spații, și să specificăm ce tip dorim să fie. După aceea, putem (sau nu) să specificăm ce parametri dorim să aibă, câți parametri să aibă și să selectăm ce tip sunt acești parametri.

Ultimul pas, care este cel mai important, este să oferim backend-ului, pentru funcția noastră, câteva exemple de output-uri și input-uri corecte, care pot fi în număr cât dorim noi. Acest lucru se face importând pe platformă un fișier Excel, în care o coloană reprezintă un exemplu de input și output pentru funcția respectivă. În acea coloană, primul rând reprezintă output-ul, iar restul liniilor reprezintă inputurile funcției respective..

Un exemplu de format excel:



	A	B	C	D
1	6	10	5	100
2	2	4	4	50
3	4	6	1	50

Acesta este un exemplu unei functii care isi doreste sa adune 2 numere de exemplu $2+4=6$.

Explicatie Cod:

a. Video

În frontend se vor respecta pașii menționați mai sus pentru video, iar dacă nu sunt respectați, va apărea un mesaj de eroare. Dacă aceste condiții sunt îndeplinite, se va trimite către backend, în primul rând, video-ul. În backend, video-ul va fi interceptat și încărcat local în fișierul VideoTutorial. În acest fișier, se va crea mai întâi un folder cu numele profesorului, iar în acel folder se va crea un alt fișier cu numele cursului. În acest fișier se va încărca video-ul respectiv, având numele urmat de data și ora când a fost creat, pentru a face acel video unic. Înainte de a se încărca video-ul, există un filtru care verifică pașii menționați mai sus (dacă are sub 50MB etc.).

După ce video-ul a fost încărcat, către UI se va trimite locația video-ului, care va fi folosită pentru afișarea video-ului în UI. După ce UI-ul a primit locația video-ului, se vor trimite și restul datelor ca JSON și acestea se vor încărca în baza de date.

b. Pdf

În mare parte, la PDF se aplică aceeași logică ca la video, doar că pentru PDF nu se fac două cereri către baza de date, ci una singură.

c. Exerciții

La exerciții, totul se face din frontend. În primul rând, se completează toate câmpurile

TextField, iar când vine vorba de încărcarea fișierului Excel de către UI, acesta va prelua fișierul respectiv și va parcurge coloanele una câte una. Prima linie, care reprezintă output-ul, va fi salvată într-un string cu forma „Output(6)”, iar restul liniilor, care reprezintă input-ul, vor fi salvate într-un string de forma „Input(2, 4)”. Când se creează numele funcției și numele parametrilor, acestea vor fi salvate sub forma „int.add”.

După ce toate datele au fost completate, se va trimite către backend un JSON care conține toate datele din UI și se vor încărca în baza de date.

4.1.5 Modulul Compiler:

Sigur, iată textul corectat:

Acest modul îi va permite studentului să își testeze cunoștințele prin oferirea unui compilator web care poate rula limbaje precum Python, JavaScript/Node.js, Java și C++. Când studentul dorește să își testeze algoritmul, va apăsa pe butonul de compilare, iar în backend se vor testa inputurile și outputurile specificate de profesor în cerința respectivă. Dacă outputurile studentului se potrivesc cu cele ale profesorului, înseamnă că rezolvarea este corectă; dacă nu, rezolvarea este greșită.

Explicație cod:

În principiu, pentru fiecare limbaj de programare există o clasă dedicată care se ocupă de lucrul cu limbajul respectiv. Aceste clase conțin două funcții importante: una care trimite către UI modelul de compilator cu funcția respectivă și alta care trimite către compilator ce trebuie să se execute, cu algoritmul scris de student.

În aceste clase, se preia din baza de date numele funcției și parametrii și se adaptează în funcție de limbajul de programare. De exemplu, dacă funcția se numește int.add, în Python se va șterge int și va rămâne doar add: `def add(...)`; în C++ va fi `int add(...){...}` și tot așa. Această logică se aplică atunci când se trimite către UI modelul de limbaj. Când dorim să compilăm, luăm scriptul scris de student și apelăm funcția în funcție de limbajul de programare folosit. De exemplu, dacă folosim Python, vom apela funcția în modul acesta: `print(add(2,4))`, iar rezultatul se va compara cu outputul din baza de date. Lucrurile se schimbă atunci când avem inputuri de tip vector, deoarece formatul

vectorilor în C++ și Java este diferit de cel din Python și JavaScript. Există o funcție specială pentru aceste două clase care va încerca să adapteze aceste inputuri pentru cele două limbaje. Pentru compilare, există o clasă numită `compileHandle` care va compila efectiv codul pentru limbajul respectiv. Pentru a compila efectiv, a fost necesară instalarea unor mașini virtuale pentru Python, Node.js și Java, iar pentru C++ s-a folosit `g++`. Practic, luăm scriptul de la student, îl facem compilabil prin clasa dedicată și apoi trimitem codul către mașina virtuală care va compila scriptul. Răspunsul obținut, care ar apărea în mod normal în consolă, îl comparăm cu outputurile din baza de date.

1. Modulul admin

Adminul are posibilitatea, pe lângă adăugarea profesorilor, să schimbe cursurile profesorilor în cazul în care un profesor nu mai poate ține cursul din motive obiective.

4.2 Implementarea modului...

4.2.1 Modulul Creare (student/professor/admin)

4.2.1.1 Student:

Backend:

- Funcția `createStudent` (Interogare Bază de Date):

Această funcție facilitează interacțiunea cu baza de date. Primește ca parametru un obiect DTO (`createStudentDto`) conținând datele studentului, instanțiază un nou model `Student` și îl înregistrează în baza de date. În cazul apariției unor erori, este lansată o excepție pentru a semnaliza eșecul operațiunii către frontend.

```
async createStudent(createStudentDto: StudentDto): Promise<IStudent> {
  try {const newStudent = await new
this.studentModel(createStudentDto);
    return newStudent.save();
  } catch (error) {
    throw new Error('Failed to create student');}}}
```

- Endpoint-ul `createStudent`:

Acest endpoint de tip POST (`/new`) gestionează cererile de înregistrare a studenților. Preia datele studentului din corpul cererii (prin decoratorul `@Body()`), apelează funcția `createStudent` din serviciul dedicat (`studentService`) pentru a crea studentul în baza de

date, generează un jeton JWT (JSON Web Token) pe baza ID-ului studentului (utilizând jwtService) și returnează acest jeton ca răspuns. În cazul în care adresa de email există deja, este lansată o excepție specifică (EmailAlreadyExistsException).

```
@Post('/new')
async createStudent(@Body() createStudentDto: StudentDto,):
Promise<{ access_token: string }> {
  try {const newStudent =
    await this.studentService.createStudent(createStudentDto);
    const payload = { sub: newStudent._id };
    return {
      access_token: await this.jwtService.signAsync(payload),};
  } catch (e) {
    throw new EmailAlreadyExistsException();}}
```

Frontend :

- Componenta SignComponents:

Această componentă React este responsabilă de randarea formularului de înregistrare. Utilizează variabila (isEmailExist) pentru a urmări dacă adresa de email introdusă este deja înregistrată. Instanțiază clasa UserAuthenticationManager pentru a gestiona logica de autentificare. Funcția handleUser este apelată la trimiterea formularului, verificând validitatea adresei de email (prin isEmailVerify) și inițiind procesul de înregistrare (signUser). De asemenea, definește structura datelor utilizatorului (user) și elementele formularului (nume de utilizator, email, parolă, imagine de profil), afișând o alertă în cazul în care email-ul există deja.

```
export const SignComponents = () => {
  const [isEmailExist, setIsEmailExist] = useState(false);
  const userAutentification = new UserAuthenticationManager ();
  const handleUser = (user: any) => {
    if (userAutentification.isEmailVerify(user.email)) {
      userAutentification.signUser(user,setIsEmailExist);}};
  const [user, setUser] = useState({
    username: "",
    email: "",
    password: "",
    profileImage: "http://localhost:3000/default/img",
  });
};
```

```

return (<AccountCard name="Sign In">
  <TextBox registerType=["username", "email", "password"]>
reg={null}
  setUser={setUser}user={user}>
  <RedirectComponents redirectHref={"/account/log"} name={"Do
you have a account?"} />
  {isEmailExist ? (<Alert color="failure"
icon={HiInformationCircle}>
    <span className="font-medium">Your email already
exists!</span>
    </Alert>) : <></>}>
  <Submit registerType={[]} reg={null} setUser={setUser}
user={user}
  handleUser={() =>
    handleUser(user)}></TextBox></AccountCard>);};

```

- Clasa UserAuthenticationManager:

Această clasă conține logica de autentificare a utilizatorilor. Oferă metode pentru logare (logUser) și înregistrare (signUser), utilizând clase specializate (Student, Professor, Admin) pentru a gestiona roluri specifice. Include metode auxiliare pentru verificarea formatului adresei de email (isEmailVerify) și existența acesteia în baza de date (emilExist). Interacționează cu API-ul backend pentru a efectua operațiunile de înregistrare și logare.

```

export class UserAuthenticationManager {
  private rout: any = useRouter();
  private entity: any = null;
  private expression: RegExp = /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i;
  private isId(id: string) {
    setCookie("id", id);
    this.rout.push("/");
    this.rout.refresh();}
  private async verifyStudentLog(user: any): Promise<string> {
    this.entity = new Student(user);
    return await this.entity.logStudent();}
  private async verifyProfessorLog(user: any): Promise<string> {
    this.entity = new Professor(user);
    return await this.entity.logProfessor();}

```

```

private async verifyAdminLog(user: any): Promise<string> {
  this.entity = new Admin(user);
  return await this.entity.logAdmin();}
private async verifyStudentSign(user: any): Promise<string> {
  user.role = "Student";
  this.entity = new Student(user);
  return await this.entity.NewStudent();}
private async verifyProfessorSign(user: any): Promise<string> {
  this.entity = new Professor(user);
  return await this.entity.NewProfessor();}
public isEmailVerify(email: any): boolean {
  return this.expression.test(email);}
public emilExist: boolean = false;
getEmilExist(): boolean {
  return this.emilExist;}
public async signUser(
  user: any,
  setIsEmailExist: Dispatch<SetStateAction<boolean>>
): Promise<any> {
  const id = await this.verifyStudentSign(user);
  if (id !== "Your email already exists") {
    setIsEmailExist(false);
    this.isId(id);
  } else {
    setIsEmailExist(true);}}
private async verifyUser(user: any): Promise<any> {
  return [
    this.verifyStudentLog(user),
    this.verifyProfessorLog(user),
    this.verifyAdminLog(user),];}
public async logUser(
  user: any,
  setIsEmailExist: Dispatch<SetStateAction<boolean>>
) {let isBreak=false;
  for (let userLog of await this.verifyUser(user)) {
    const id = await userLog;
    if (id !== " ") {
      this.isId(id);

```

```

        isBreak=true;
        break;}}
    if(!isBreak)
        setIsEmailExist(true);}}

```

4.2.1.2 Professor:

Backend:

- Funcția addNewProfessor (Interogare Bază de Date):

Această funcție adaugă un nou profesor în baza de date prin intermediul serviciului professorService. În cazul unei erori, aceasta este înregistrată în consolă și propagată mai departe.

```

    async addNewProfessor(professor: ProfessorDto) {
        try {return await
        this.professorService.createProfessor(professor);
        } catch (error) { console.error(error);
        throw error;}}

```

- Endpoint-ul uploadFile:

Acest endpoint protejat de AdminGuard (doar administratorii au acces) permite încărcarea unui fișier Excel (@UploadedFile()) conținând date despre profesori. Fișierul este încărcat în memorie (file.buffer), iar datele sunt extrase din prima foaie de calcul (workbook.getWorksheet(1)).

Se efectuează validări asupra formatului fișierului (antetul coloanelor) și asupra formatului adreselor de email. Pentru fiecare rând valid, se creează un obiect ProfessorDto cu datele extrase și se adaugă în baza de date utilizând addNewProfessor. Rezultatele operațiunilor sunt colectate în promises, iar apoi filtrate pentru a returna doar dacă sau adăugat profesorii.

```

@UseGuards(AdminGuard)
    async uploadFile(@UploadedFile() file: Express.Multer.File) {
        try {
            const workbook = new Excel.Workbook();
            const buffer = file.buffer;
            await workbook.xlsx.load(buffer);
            const worksheet = workbook.getWorksheet(1);

```

```

const promises = [];
worksheet.eachRow(async (row, rowNumber) => {
  if (rowNumber === 1) {
    if (row.getCell(1).value !== 'username' ||
        row.getCell(2).value !== 'email' ||
        row.getCell(3).value !== 'password'
    ) {promises.push(Promise.resolve("Your excel didn't respect
the format"),);
      return;}} else {
    const professor: ProfessorDto = {
      _id: new Types.ObjectId(),
      username: 'professor',
      email: 'professor@example.com',
      password: 'password',
      profileImage: 'http://localhost:3000/default/img',
      role: 'professor',
      coursesId: [],};
    professor.username = row.getCell(1).value.toString();
    const emailValue =typeof row.getCell(2).value === 'string'
      ? row.getCell(2).value
      : JSON.parse(JSON.stringify(row.getCell(2).value));
    const email =
      emailValue.text === undefined ? emailValue :
emailValue.text;
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      promises.push(Promise.resolve('Invalid email format'));
      return; }
    professor.password = row.getCell(3).value.toString();
    professor.email = email;
    const promise =
this.adminService.addNewProfessor(professor);
    promises.push(promise);});
    const results = await Promise.all(promises);
    return results.filter((result) => result !== true);
  } catch (error) {
    console.error(error);
    throw error;}}

```

Frontend:

- Componenta UploadProfessor:

Această componentă React gestionează încărcarea fișierului Excel conținând datele profesorilor. Utilizează starea (warning, files, isExcel) pentru a afișa mesaje de avertizare, a stoca fișierul selectat și a verifica dacă fișierul are extensia corectă (.xls sau .xlsx).

Funcția handleFileChange apelează serviciul AdminsService pentru a trimite fișierul către backend. Funcția handleSetFiles este apelată când utilizatorul selectează un fișier, verificând tipul acestuia și actualizând starea corespunzător.

```
export const UploadProfessor = () => {
  const [warning, setWarning] = useState([
    `You don't have the right file format!`,
  ]);
  const [files, setFiles] = useState<File | null>(null);
  const [isExcel, setIsExcel] = useState(false);
  const handleFileChange = async () => {
    const admin = new AdminsService();
    await admin.sendProfessor(files, setWarning, setIsExcel,
getCookie("id"));
  };
  const handleSetFiles = (e: any) => {
    const file = e.target.files[0];
    if (
      file.type ===
      "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" ||
      file.type === "application/vnd.ms-excel"
    ) {
      setIsExcel(false);
      setFiles(e.target.files[0]);
    } else {
      setIsExcel(true);
    }
  };
  return (
    <><AccountCard name="Upload Professor">
```

```

    <>
    <div className="flex items-center justify-center w-full my-
4">
      <label
        htmlFor="dropzone-file"
        className="flex flex-col items-center justify-center w-
full h-64 border-2 border-gray-300 border-dashed rounded-lg cursor-
pointer bg-gray-50  hover:bg-gray-100">
        <div className="flex flex-col items-center justify-
center pt-5 pb-6">
          <IconCloud />
          <p className="mb-2 text-sm text-gray-500 dark:text-
gray-400">
            <span className="font-semibold">Click to
upload</span> or drag
            and drop</p>
          <p className="text-xs text-gray-500 dark:text-gray-
400">Exel only</p>
        </div>
        <input
          id="dropzone-file"
          type="file"
          className="hidden"
          onChange={handleSetFiles}/>
        </label>
      </div>{isExel &&
        warning.map((warn, index) => (
          <Alert
            color="failure"
            icon={HiInformationCircle}
            key={index}
            className="mb-4">
              <span className="font-medium">{warn}</span>
            </Alert>))}
      <Link
        href={` /AddProfessorTutorial`}
        className="mt-4 inline-block font-medium text-blue-600
dark:text-blue-500 hover:underline hover:text-red-600">

```

```

        Tutorial
    </Link>
    <button
        type="submit"
        className=" my-4 relative h-12 w-full min-w-[200px] text-
white bg-blue-700 hover:bg-blue-800 focus:ring-4 focus:ring-blue-300
font-medium rounded-lg text-sm px-5 py-2.5 mr-2 mb-2 dark:bg-blue-600
dark:hover:bg-blue-700 focus:outline-none dark:focus:ring-blue-800"
        onClick={() => !isExcel && handleFileChange()}>
        Submit
    </button></>
</AccountCard></>));};

```

- Clasa AdminsService

Aceasta clasa contine logica necesară pentru a trimite și gestiona datele profesorilor către backend prin intermediul unui fișier Excel. Aceasta facilitează comunicarea dintre interfața utilizator (frontend) și server (backend) în contextul administrării informațiilor despre profesori.

Metoda principală a acestei clase, sendProfessor, preia fișierul Excel selectat de administrator, îl pregătește pentru transmiterea către server și inițiază o cerere HTTP către un endpoint specific (/admin/exel) pe backend. În cadrul acestei cereri, fișierul este transmis împreună cu ID-ul administratorului pentru autentificare și autorizare. După ce serverul procesează fișierul, răspunsul este analizat de metoda sendProfessor. În cazul în care serverul returnează un array care specifică ,că profesorul a fost creat cu success, iar daca nu acest este interpretat ca erori sau avertismente și sunt afișate adminului prin intermediul funcției setWarning. Variabila isExcel este, de asemenea, actualizată pentru a indica dacă fișierul a fost procesat cu succes sau nu.

```

export class AdminsService {
    public async sendProfessor(
        file: any,
        setWarning: Dispatch<SetStateAction<string[]>>,
        setIsExcel: Dispatch<SetStateAction<boolean>>,
        id:string|undefined
    ) {const exel=new FormData();
        exel.set("file",file);

```



```

        const res = await fetch(`${urlBackend}admin/exel`,
sendFiles(exel,id));
        const z = await res.json();
        if (typeof z !== "boolean") {          setWarning(z);
setIsExel(true);}} }

```

4.2.1.3 Admin

Admin-ul se poate crea doar din baza de date si doar cel care detine baza de date poate sa adauge admini,si acest lucru se poate face ori incarand in mongodb compass un json care sa arate ceva de genul acesta:

```

{username: 'Joe doe',
  email: 'joe.doe@email.com',
  password: '1234',
  profileImage: 'http://localhost:3000/default/img',
  role: 'admin'}

```

or trebuie sa scriem in mongodb shell aceste comenzi:

```

use ProgramingLandb
db.admins.insertOne({username: 'Joe doe',
  email: 'joe.doe@email.com',
  password: '1234',
  profileImage: 'http://localhost:3000/default/img',
  role: 'admin'})

```

Si va trebui verificat dinainte daca email-ul este uniq.

4.2.2 Modulul logare (student/professor/admin)

4.2.2.1 Student

Backend:

- Funcția logUser are rolul de-a verifica daca student-ul exist in baza de date iar daca exista va crea va returna toate datele student-ului din baza de date.Iar daca nu va returna o eroare.

```

async logUser(email: string, password: string): Promise<IStudent> {
  try {const user = await this.studentModel.findOne({
    email: email,

```

```

        password: password,
    });
    return user;
} catch (error) {
    console.error(error);
    throw new Error('Failed to log user');}}

```

- Funcția makeJwt are rolul sa returneze o cheie de autentificare jwt in cazul in care student-ul exista si daca nu există va returna un string gol.

```

async makeJwt(student: any) {
    try {if (student !== null) {
        const payload = { sub: student._id };
        return {access_token: await
this.jwtService.signAsync(payload),,}}
        return {access_token: ' ',,}}
    } catch (error) {
        console.error(error);
        throw new Error('Failed to make JWT');}}

```

- Endpoint-ul logStudent:

Acest endpoint gestionează procesul de autentificare a unui student existent. Este un endpoint de tip POST (/log) care primește datele de autentificare (email și parolă) sub forma unui obiect LogDto.

```

@Post('/log')
async logStudent(@Body() log: LogDto): Promise<{ access_token: string
}> {
    try {const logStudent = await
this.studentService.logUser(log.email,log.password, );
        return this.studentService.makeJwt(logStudent);
    } catch (error) {
        console.error(error);
        throw new Error('Internal Server Error'); }}

```

Frontend:

Componenta LogIn este responsabilă de randarea formularului de autentificare și gestionarea

logicii asociate. Utilizează variabile de stare (isEmailExist, user, isEmail) pentru a urmări validitatea datelor introduse de utilizator și pentru a controla afișarea mesajelor de eroare.

Instanțiază clasa UserAuthenticationManager pentru a gestiona logica de autentificare, inclusiv verificarea formatului adresei de email și comunicarea cu backend-ul pentru a efectua autentificarea propriu-zisă.

Clasa UserAuthenticationManager conține o metoda numita logUser in care aceasta metoda are rolul de-a trimite request-uri la totii userii si user-ul corespunzator va fii marcat cu rolul lui.

Funcția handleUser este apelată la trimiterea formularului. Aceasta verifică validitatea adresei de email utilizând metoda isEmailVerify din UserAuthenticationManager și, dacă este validă, inițiază procesul de autentificare prin apelarea metodei logUser. În caz contrar, setează variabila isEmail pentru a indica faptul că adresa de email este invalidă și a afișa un mesaj de eroare corespunzător.

Componenta definește structura datelor utilizatorului (user) și elementele formularului de autentificare (câmpuri pentru email și parolă, buton de trimitere). De asemenea, afișează un link către pagina de înregistrare pentru utilizatorii care nu au încă un cont și o alertă de eroare în cazul în care autentificarea eșuează (dacă isEmailExist este true).

În plus, componenta utilizează un hook useEffect pentru a actualiza starea isEmail în funcție de validitatea adresei de email introduse de utilizator. Aceasta asigură o validare în timp real a câmpului de email și previne trimiterea formularului cu date invalide.

```
export const LogIn = () => {
  const [isEmailExist, setIsEmailExist] = useState(false);
  const [user, setUser] = useState({username: "",email: "",
    password: "",profileImage: "http://localhost:3000/default/img",
    role: "",});
  let [isEmail, setIsEmail]: any = useState(false);
  const userAuthentication = new UserAuthenticationManager();
  useEffect(() => {
    const handleEmail = () => {
      if(user.email===''){ setIsEmail(true);
      }else{setIsEmail(false); }};
    handleEmail();
  });
}
```

```

    }, [user, isEmail]);
    const handleUser = (user: any) => {
      if (userAuthentication.isEmailVerify(user.email)) {
        userAuthentication.logUser(user, setIsEmailExist);
      } else {
        setIsEmail(true);
      }
      return (
        <><AccountCard name="Log In">
          <TextBox registerType={["email", "password"]} reg={null}
            setUser={setUser} user={user}>
            <RedirectComponents redirectHref={"/account/sign"} name={"You
            don't have an account?"} /> {isEmailExist ? (
              <Alert color="failure" icon={HiInformationCircle}>
                <span className="font-medium">Your email or
                password are incorect!</span> </Alert>
              ) : <></>
            }
            <Submit registerType={[]} reg={null} setUser={setUser}
              user={user}
              handleUser={() => {isEmail===false?handleUser(user):null}}
            />
          </TextBox></AccountCard></>);
    };
  }
}

```

Logica este aceeași și pentru admin sau professor.

4.2.3 Modulul creare curs

Backend:

- Funcția createNewCourse (Interogare Bază de Date):

Această funcție facilitează crearea unui nou curs în baza de date și asocierea acestuia cu profesorul creator. Primește ca parametri un obiect DTO (course de tip CoursesDto) conținând detaliile cursului și ID-ul profesorului (professorId). În cadrul funcției, se instanțiază un nou model Courses utilizând datele primite, iar apoi acesta este salvat în baza de date. Ulterior, se obține ID-ul decriptat al profesorului și se preia obiectul profesor din baza de date. ID-ul noului curs este adăugat în lista de cursuri asociate profesorului, iar modificările sunt salvate. În cazul apariției unor erori, este lansată o excepție cu un mesaj descriptiv.

```

async createNewCourse(course: CoursesDto, professorId: string) {
  try {const newCourse = await new this.coursesModel(course);
    newCourse.save();
    const decryptId = await

```

```

    this.professorService.decryptJwt(professorId);
    const professor = await
this.professorService.getProfessor(decryptId);
    professor.coursesId.push(newCourse._id);
    professor.save();
    return newCourse.name; catch (error) {
    throw new Error(`Error while creating new course: ${error}`);}}

```

Endpoint-ul createCourse:

Acest endpoint de tip POST (/new), protejat de ProfessorGuard, gestionează cererile de creare a unor noi cursuri. Preia ID-ul profesorului autentificat din cookie-uri (@Cookies('id')) și datele cursului din corpul cererii (@Body()). Ulterior, apelează funcția createNewCourse din serviciul dedicat (courseService) pentru a efectua operațiunile de creare și asociere a cursului cu profesorul. În cazul unei erori, aceasta este lansată o excepție generică.

```

@Post('/new')
@UseGuards(ProfessorGuard)
async createCourse(@Cookies('id') id, @Body() createCourseDto:
CoursesDto,
): Promise<string> {try {
    const newCourse = await
this.courseService.createNewCourse(createCourseDto, id);
    return newCourse;
} catch (error) {console.error(error);
    throw new Error('An error occurred while creating the course.')}
}}

```

Frontend:

- Componenta CoursesName:

Această componentă funcțională React (FC) este responsabilă de randarea interfeței pentru crearea sau actualizarea unui curs. Primește proprietățile isUpdated (indicând dacă este vorba de o actualizare), courseName (numele cursului în cazul unei actualizări) și setDialog (o funcție pentru gestionarea dialogurilor).

Componenta utilizează hook-ul useState pentru a gestiona starea internă a datelor cursului

(course) și a numelui acestuia (nameCours). De asemenea, variabila isAllRight este utilizată pentru a urmări dacă toate câmpurile obligatorii au fost completate.

Funcția handleUser gestionează crearea unui nou curs, asigurând validarea datelor și apelând metoda newCourse a unei instanțe a clasei Courses pentru a trimite datele către backend. Funcția handleUserUpdate are un rol similar, dar pentru actualizarea unui curs existent.

În final, componenta returnează structura JSX a formularului, incluzând câmpuri pentru introducerea numelui, vizibilității și descrierii cursului, un mesaj de eroare condiționat de starea isAllRight și un buton de trimitere care apelează funcția corespunzătoare (handleUser sau handleUserUpdate) în funcție de context.

```
export const CoursesName: FC<{isUpdated: boolean; courseName: string;
  setDialog: Dispatch<SetStateAction<JSX.Element | undefined>> |
  undefined;
}> = ({ isUpdated, courseName, setDialog }) => {
  const [course, setCourse] = useState({name: "",visibility: false,
description: "",
  imagePath: `${urlBackend}default/cours/1`,studentId:
[],colaborationId: [],
  courses: [],});
  const [nameCours, setNameCours] = useState({ Name: "" });
  const [isAllRight, setIsAllRight] = useState(true);
  const rout = useRouter();
  const isId = (path: string) => {
    rout.push(`http://localhost:3001/professorworkspace/${path}`);
    rout.refresh();};
  function randomIntFromInterval(min:number, max:number) {
    return Math.floor(Math.random() * (max - min + 1) + min) }
  const handleUser = async (user: any) => {
    course.name = HandleGenericFuntion.replaceSpaceWithUnderline(
      nameCours.Name);
    if (course.name && course.description) {setIsAllRight(true);
      course.imagePath =
`${urlBackend}default/cours/${randomIntFromInterval(1,7)}`;
      const test = new Courses(user);
      const t = await test.newCourse();
```

```

        isId(t); } else {
            setIsAllRight(false); });
const handleUserUpdate = async (user: any) => {
    course.name = nameCours.Name;
    const test = new Courses(user);
    const t = await test.updateCourse(courseName);
    if (setDialog) {setDialog(undefined);
        rout.refresh();}};
return (<><AccountCard name="Create courses">
    <TextBox registerType={["Name"]} reg={null}
setUser={setNameCours}
    user={nameCours}>
        <SelectVizibility registerType={["Name"]} reg={null}
setUser={setCours}
    user={course}/>
        <TextareaWithLimit registerType={[]} reg={null}
setUser={setCours}
    user={course}/>
        {!isAllRight ? (<Alert color="failure"
icon={HiInformationCircle}
className="mt-4"><span className="font-medium">Please fill all the
fields. </span></Alert>) : (<></>)}<Submit registerType={[]}
reg={null}
        setUser={setCours}user={course}handleUser={() =>
            isUpdated ? handleUserUpdate(course) :
handleUser(course) }/>
    </TextBox></AccountCard></> );};

```

- Clasa Courses:

Această clasă servește drept model pentru un curs în cadrul aplicației frontend. Ea conține datele relevante despre un curs, cum ar fi numele, vizibilitatea, descrierea, imaginea asociată, ID-urile studenților înscriși, ID-urile colaboratorilor și o listă cu cursurile Video/Pdf/Exercitii.

Constructorul clasei primește un obiect de tip ICourses (o interfață care definește structura datelor unui curs) și inițializează proprietățile clasei cu valorile corespunzătoare din acest obiect.

Metoda newCourse ,această metodă asincronă gestionează crearea unui nou curs. Obține ID-ul profesorului, pregătește un obiect JSON cu datele cursului și trimite o cerere POST către

`endpoint-ul /api/handleNewCourseApi(in care acest api va trimite către backend un request).

Dacă cererea are succes, extrage numele cursului din răspunsul serverului și îl returnează. În caz contrar, apelează funcția notFound.

```
export class Courses implements ICourses {
  name: string = "";
  vizibility: boolean = false;
  studentId: any = [];
  description: string = "";
  imagePath: string = "";
  colaborationId: any = [];
  courses: any = [];
  constructor(courses: ICourses) {
    this.name = courses.name;
    this.vizibility = courses.vizibility;
    this.studentId = courses.studentId;
    this.description = courses.description;
    this.imagePath = courses.imagePath;
    this.colaborationId = courses.colaborationId;
    this.courses = courses.courses;
  }
  public async updateCourse(courseName: string) {
    const id = getCookie("id")?.toString();
    const option = {
      method: "Post",
      credentials: "include" as RequestCredentials,
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ courseBody: this, oldCourseName:
courseName }),
    };
    const api = await fetch(
      `/api/handleUpdateCourseApi`,
      sendToServerCookies(JSON.stringify(option), id)
    );
    if (!api.ok) notFound();
  }
}
```



```

public async newCourse() {
  const id = getCookie("id")?.toString();
  const option = {
    method: "POST",
    credentials: "include" as RequestCredentials,
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({ courseBody: this }),
  };
  const api = await fetch(
    "/api/handleNewCourseApi",
    sendToServerCookies(JSON.stringify(this), id)
  );
  if (api.ok) {
    const { text } = await api.json();
    return text;
  } else {
    notFound();}}

```

4.2.4 Modulul Video/Pdf/Exercitii

4.2.4.1 Video

Backend:

- Funcția addMediaFormat (Interogare Bază de Date):

Această funcție are rolul de a adăuga un nou video la un curs existent. Ea primește ca parametri ID-ul cursului (coursesId) și obiectul media (media) care conține detaliile formatului video.

Funcția folosește findById pentru a găsi cursul în baza de date pe baza ID-ului său. Adaugă noul obiect media (în cazul acesta video-ul) în array-ul courses al cursului găsit. După salvează cursul actualizat în baza de date. După returnează indexul la care a fost adăugat noul format media în array-ul courses. Acest index va fi folosit pentru a identifica materialul video în frontend. În cazul în care apare o eroare (de exemplu, dacă nu se găsește cursul sau dacă există o problemă la salvare), este aruncată o excepție.

```

async addMediaFormat(coursesId: Types.ObjectId, media: IVideo |

```

```

IDocumentFormat | ICompilers,) {
    try {const coursesInput: ICourses = await
this.videoModel.findById(coursesId);
    coursesInput.courses.push(media);
    await coursesInput.save();
    return coursesInput.courses.length - 1; } catch (error) {
    throw new Error('Failed to add media format'); }}

```

Endpoint-urile:

- addVideoForVideoCourses:

Acest endpoint gestionează încărcarea efectivă a fișierului video pe server. Este un endpoint de tip POST, protejat de ProfessorGuard, ceea ce înseamnă că doar profesorii autentificați pot accesa acest endpoint. Utilizează un interceptor de fișiere (FileInterceptor) pentru a gestiona încărcarea fișierului. De asemenea, specifică locația de stocare a fișierului și un filtru pentru a permite doar fișiere video. Si dacă totul este ok va returna către frontend locația video-ului.

```

@Post('/:professorName/:courseName/add/video/videoInput')
@UseGuards(ProfessorGuard)
@UseInterceptors(
    FileInterceptor('file', {storage:
diskStorage(fileHandle.destinationVideo()),
    fileFilter: fileHandle.filterVideo(),})
    async addVideoForVideoCourses(
        @Req() req: Request,
        @Cookies('id') id: string,
        @Body('filename') filename: string,
    ) {try {const { dest }: any = req.body;
        const finalDest =dest.replace('E:\\Licenta-Platforma-de-
invatare-programare\\backend\\src\\VideoTutorial\\',
        ''),).replace(/\\/g, '/') + '/' +
            filename; return finalDest; } catch (error) {
            console.error(error);
            throw new Error('An error occurred while adding the video for
the video course.',);}}

```

- createTextForVideoCourses:

Acest endpoint gestionează crearea unui nou material video în baza de date, adăugând informații despre video (titlu, descriere, cale fișier) la cursul corespunzător. Este un endpoint de tip POST, protejat de ProfessorGuard. Preia numele cursului din parametrii rutei (@Param) și datele video din corpul cererii (@Body).

```
@Post('/:courseName/add/video/textInput')
@UseGuards(ProfessorGuard)
async createTextForVideoCourses(
  @Param('courseName') courseNameId: string,
  @Body() createCourseDto: IVideo,
): Promise<string> {
  try {const courseId: Types.ObjectId =
    await this.videoService.takeCoursId(courseNameId);
    const videoDto = createCourseDto;
    videoDto.format = 'Video';
    const courses = await this.videoService.addMediaFormat(courseId,
videoDto,);
    return await courses.toString();} catch (error) {throw new
Error(
  'An error occurred while creating the text for the video
course.',);}}
```

Frontend:

- Componenta UploadVideo:

Componenta React UploadVideo este responsabilă de randarea interfeței de încărcare și actualizare a videoclipurilor în cadrul unui curs. Aceasta primește proprietăți precum isUpdated, videoName, courseName, setDialog și professorEmail pentru a-și adapta comportamentul în funcție de context. Starea internă a componentei este gestionată prin variabilele videoDescription, isAllRight și warning, care stochează respectiv descrierea videoclipului, validitatea câmpurilor formularului și eventualele mesaje de avertizare.

Funcția handleNewVideo gestionează încărcarea unui nou videoclip, prevenind comportamentul implicit al formularului, extrăgând datele videoclipului și verificând dacă toate câmpurile sunt completate. Dacă validarea are succes, se creează o instanță a clasei VideoManaging și se apelează metoda sendText pentru a trimite datele către backend și a obține ID-ul videoclipului.

Ulterior, utilizatorul este redirecționat către pagina de vizualizare a videoclipului nou creat. În cazul în care există câmpuri necompletate, se setează `isAllRight` la false și se afișează un mesaj de avertizare.

Similar, funcția `handleVideoUpdate` gestionează actualizarea unui videoclip existent. După validarea datelor, se apelează metoda `sendTextUpdate` a clasei `VideoManaging` pentru a trimite datele actualizate către backend. Dacă actualizarea are succes, dialogul modal este închis.

În ceea ce privește randarea, componenta utilizează elemente precum `VideoCard` pentru a conține formularul, `TitleInput` și `TextareaInput` pentru introducerea titlului și descrierii videoclipului, `UploadVideoInput` pentru încărcarea fișierului video și `Alert` pentru afișarea mesajelor de avertizare. Butonul de trimitere declanșează funcția `handleNewVideo` sau `handleVideoUpdate` în funcție de valoarea proprietății `isUpdated`.

În plus, componenta utilizează hook-ul `useRouter` pentru a redirecționa utilizatorul după încărcarea cu succes a unui videoclip nou și depinde de clasa externă `VideoManaging` pentru a gestiona comunicarea cu backend-ul.

```
export const UploadVideo: FC<{isUpdated: boolean; videoName: string;
coursName: string; setDialog: Dispatch<SetStateAction<JSX.Element |
undefined>> | undefined; professorEmail: string;}> = ({ isUpdated,
videoName, coursName, setDialog, professorEmail }) => {
  const [videoDescription, setVideoDescription] = useState({
    title: "", filePath: "",description: "",});
  const rout = useRouter();
  const [isAllRight, setIsAllRight] = useState(true);
  const [warning, setWarning] = useState("");
  const handleNewVideo = async (event:any) => {
    event.preventDefault();
    const { title, description, filePath } = videoDescription;
    if (title !== "" && description !== "" && filePath !== "") {
      const videoText: VideoManaging = new VideoManaging(coursName);
      const videoId = await videoText.sendText(title, description,
filePath);
      rout.push(`/CoursView/${professorEmail}/${coursName}/${videoId}/view`)
```

```

;
    } else {
        setIsAllRight(false);
        setWarning("Please fill all the fields correctly."); });
const handleVideoUpdate = async (event:any) => {
    event.preventDefault();
    if(!isAllRight){
        return;}
    const videoText: VideoManaging = new VideoManaging(videoName);
    await videoText.sendTextUpdate(videoDescription.title,
videoDescription.description, videoDescription.filePath, coursName);
    if (setDialog) {
        setDialog(undefined); });
return (<><VideoCard>
    <TitileInput setVideoDescription={setVideoDescription} />
    <TextareaInput setVideoDescription={setVideoDescription} />
    <UploadVideoInput
        setVideoDescription={setVideoDescription}
        setWarning={setWarning}
        setIsAllRight={setIsAllRight}/>
    {!isAllRight ? (
        <Alert color="failure" icon={HiInformationCircle}
className="mt-4">
            <span className="font-medium">{warning}</span>
        </Alert>) : (<></>)}
    <form onSubmit={isUpdated ? handleVideoUpdate :
handeNewVideo}>
        <button
            type="submit"
            className=" my-4 relative h-12 w-full min-w-[200px] text-
white bg-blue-700 hover:bg-blue-800 focus:ring-4 focus:ring-blue-300
font-medium rounded-lg text-sm px-5 py-2.5 mr-2 mb-2 dark:bg-blue-600
dark:hover:bg-blue-700 focus:outline-none dark:focus:ring-blue-800"
            // onClick={!isUpdated ? handeNewVideo : handleVideoUpdate}>
            Submit</button></form></VideoCard></>);};

```

- Clasa VideoManaging:

Clasa VideoManaging gestionează interacțiunea dintre frontend și backend pentru operațiunile

legate de videoclipuri. Constructorul inițializează clasa cu numele videoclipului. Metoda privată `setVideoText` trimite datele textuale ale videoclipului către backend pentru a crea o nouă intrare în baza de date, returnând ID-ul videoclipului creat dacă operațiunea are succes.

Metoda privată `setVideoTextUpdate` este similară, dar actualizează datele textuale ale unui videoclip existent. Metoda privată `getProfessorName` obține numele profesorului curent din backend printr-o cerere POST.

Metoda privată `setVideo` încarcă fișierul video pe server și returnează calea către fișierul încărcat. Metoda privată `setUpdateVideo` este similară, dar actualizează fișierul video asociat unui videoclip existent.

Metoda publică `sendTextUpdate` actualizează complet un videoclip existent, obținând numele profesorului, încărcând fișierul video (dacă este cazul) și actualizând datele textuale. Metoda publică `sendText` creează un nou videoclip, obținând numele profesorului, încărcând fișierul video și creând o nouă intrare în baza de date.

```
export class VideoManaging {
  private videoName: string = "";
  constructor(videoName: string) {
    this.videoName = videoName; }
  private async setVideoText(title: string,description: string,
video: string
): Promise<string> {
  const test = await fetch("/api/handleNewVideoApi",
sendToServerCookies(
  JSON.stringify({ title: title,description:
description,videoPath: video, videoName:
this.videoName,}),undefined));
  const { text, ok } = await test.json();
  if (ok) {
    return text;
  } else {
    notFound();} }
  private async setVideoTextUpdate( title: string,description:
string,video: string,
coursName: string): Promise<string> {
```

```

        const test = await fetch(
"/api/handleUpdateCourseApi/handleSendUpdateVideoApi",
        sendToServerCookies(
            JSON.stringify({ title: title,description:
description,videoPath: video,videoName: this.videoName,coursName:
coursName,  }),undefined) );
        const { text, ok } = await test.json();
        if (!ok) notFound();
        return text;}
private async getProfessorName(): Promise<string> {
    const option = {
        method: "POST",
        credentials: "include" as RequestCredentials,
        headers: {
            "Content-Type": "application/json",
            Cookie: `id=${getCookie("id")}`,  },  };
    const professor = await
fetch("/api/handleProfessorApi/professorName",option);
    const { text, ok } = await professor.json();
    if (ok) return text;
    else notFound();
}
private async setVideo(filePath: string,professorName: string
): Promise<string> {
    const body = new FormData();
    body.set("file", filePath);
    body.set("professorName", professorName);
    body.set("videoName", this.videoName);
    const response = await
fetch("/api/handleVideoApi/videoInput",sendFiles(body,
getCookie("id")));
    const { text, ok } = await response.json();
    if (!ok) notFound();return text;}
private async setUpdateVideo(filePath: string,professorName:
string,courseName: string
): Promise<string> {
    const body = new FormData();
    body.set("file", filePath);

```

```

        body.set("professorName", professorName);
        body.set("courseName", courseName);
        body.set("videoName", this.videoName);
        const response = await
fetch("/api/handleVideoApi/videoInputUpdate",
        sendFiles(body, getCookie("id")));
        const { text, ok } = await response.json();
        if (!ok) notFound();
        return text;}

    public async sendTextUpdate(title: string,description: string,file:
string,coursName: string
    ) {const professorName = await this.getProfessorName();
        let video = "";
        if (file !== "") {
            video = await this.setUpdateVideo(file, professorName,
coursName);}
        const response: string = await this.setVideoTextUpdate(
            title,description,video,coursName);
        return await response;}

    public async sendText(title: string, description: string, file:
string) {
        const professorName = await this.getProfessorName();
        const video = await this.setVideo(file, professorName);
        const response: string = await this.setVideoText(title,
description, video);
        const r = await response;
        return r;}}

```

4.2.4.2 Pdf

În mare parte, codul este asemănător cu tot codul de la video, dar la frontend nu vom mai avea un câmp de descriere.

4.2.4.3 Exerciții

Backend:

- Funcția addMediaFormat

Această funcție este responsabilă pentru adăugarea unui nou format media (în cazul acesta

compiler) la un curs existent în baza de date. Ea primește ca parametri ID-ul cursului și obiectul media ce urmează a fi adăugat. Funcția găsește mai întâi cursul corespunzător în baza de date utilizând ID-ul său. Apoi adaugă noul obiect media în lista de cursuri (courses) a cursului identificat. După aceea, salvează cursul actualizat în baza de date. În final, returnează indexul la care a fost adăugat noul element media, index care va fi folosit pentru a identifica materialul în frontend. Dacă apare vreo eroare în timpul procesului (de exemplu, dacă nu se găsește cursul), aceasta este propagată mai departe pentru a fi gestionată în mod corespunzător.

```
async addMediaFormat(  
  courseId: Types.ObjectId,  
  media: IVideo | IDocumentFormat | ICompilers,  
) {  
  try {  
    const course: ICourses = await  
this.compilerModel.findById(courseId);  
    course.courses.push(media);  
    await course.save();  
    return course.courses.length - 1;  
  } catch (error) {  
    console.error(error);  
    throw error;}}
```

Endpoint:

- Endpoint-ul newExercices:

Acest endpoint este utilizat pentru a adăuga exerciții noi la un curs specificat. Este protejat de ProfessorGuard, asigurându-se că doar profesorii autentificați pot accesa această funcționalitate. Endpoint-ul primește numele cursului din parametrii URL și detaliile exercițiului din corpul cererii HTTP. Folosind serviciul compilerService, obține ID-ul cursului pe baza numelui său. Apoi, apelează funcția addMediaFormat pentru a adăuga noul exercițiu la curs și returnează indexul la care a fost adăugat. Dacă apare vreo eroare în timpul acestui proces, aceasta este înregistrată în consolă, iar un mesaj de eroare generic este returnat utilizatorului.

```
@Post('/:courseName/new/exercices')
```

```

@UseGuards(ProfessorGuard)
async newExercices(
  @Body() exercices: ICompilers,
  @Param('courseName') courseName: string,
) {
  try {
    const courseId: Types.ObjectId =
      await this.compilerService.takeCoursId(courseName);
    return await this.compilerService.addMediaFormat(courseId,
exercices);
  } catch (error) {
    console.error(error);
    throw new Error('An error occurred while adding new exercises');
  }
}

```

Frontend:

Această componentă React este responsabilă pentru interfața de adăugare/editare a exercițiilor. Ea primește ca props o funcție setDialog (pentru gestionarea unui dialog modal), numele cursului, o variabilă isUpdated care indică dacă exercițiul este nou sau este editat, numele exercițiului (în cazul editării) și adresa de email a profesorului.

Componenta gestionează starea locală pentru diverse aspecte ale exercițiului, cum ar fi exemple, intrări, parametri ai funcției, precum și o variabilă isAllRight pentru a indica validitatea datelor și un mesaj de avertizare warning. De asemenea, are un obiect exercices care stochează toate detaliile exercițiului.

Funcțiile handleExemples și handleInputs sunt folosite pentru a adăuga noi exemple și intrări, respectiv, la exercițiu. Componenta utilizează o instanță a clasei ExelHandle pentru a gestiona încărcarea datelor din fișiere Excel.

Funcția finalUpdate pregătește datele finale ale exercițiului pentru a fi trimise către backend, combinând parametrii, intrările și ieșirile.

Funcțiile send și sendUpdate sunt responsabile pentru trimiterea datelor către backend prin intermediul unor cereri HTTP fetch. send este folosită pentru a adăuga un nou exercițiu, în timp

ce `sendUpdate` este utilizată pentru a actualiza un exercițiu existent. După trimiterea cu succes, utilizatorul este redirecționat către pagina de vizualizare a exercițiului.

Componenta include, de asemenea, elemente de interfață utilizator, cum ar fi câmpuri de input pentru detaliile exercițiului, meniuri dropdown pentru selectarea tipurilor de date, un buton pentru încărcarea fișierelor Excel și un buton pentru trimiterea datelor. În cazul în care există erori de validare, se afișează un mesaj de avertizare.

```
export const ExercicesComponens: FC<{
  setDialog: Dispatch<SetStateAction<JSX.Element | undefined>> |
undefined;
  courseName: string;
  isUpdated: boolean;
  exercicesName: string;
  professorEmail: string;
}> = ({ setDialog, courseName, isUpdated, exercicesName,
professorEmail }) => {
  const [exemples, setExemples] = useState(["Input:\nOutput:"]);
  const [inputs, setInputs] = useState<string[] | null>([]);
  const [funtionInputs, setFuntionInputs] = useState<string[]>([]);
  const [combineParams, setCombineParams] = useState<string[]>([]);
  const [isAllRight, setIsAllRight] = useState(true);
  const [warning, setWarning] = useState("");
  const [exercices, setExercices] = useState<ICompiler>({
    title: "",problemRequire: "",
problemExemples: [...exemples],
problemInputs: [],
problemOutputs: [],
funtionProblemModel: "",
problemParameter: "",
urlName: "",
format: "Compiler",});
  const handleExemples = () => {
    const newExemples = [...exemples, "Input:\nOutput:"];
    setExemples(newExemples);
    setExercices({
      ...exercices,
      problemExemples: newExemples,});};
```

```

const handleInputs = () => {
  if (inputs === null) {
    setInputs(["Input"]);
  } else {
    setInputs([...inputs, "Input"]);
  }
};
const [items, setItems] = useState([]);
const hndleExel = new ExelHandle();
const finalUpdate = () => {
  let s = "";
  combineParams.forEach((e: string) => {
    s += "," + e;
  });
  let output: string[] = [];
  let input: string[] = [];
  items.forEach((o: any) => {
    let out = [];
    let inp = [];
    for (let i = 0; i < o.length; i++) {
      if (i === 0) {
        out.push(o[i]);
      } else {
        inp.push(o[i]);
      }
    }
    output.push(`Output(${out})`);
    input.push(`Input(${inp})`);
  });
  const problemToSed = { ...exercices };
  problemToSed.problemParameter = s.substring(1);
  problemToSed.urlName = courseName;
  problemToSed.problemOutputs = output;
  problemToSed.problemInputs = input;
  problemToSed.funtionProblemModel =
    funtionInputs[0] !== undefined ? funtionInputs[0] : "";
  return problemToSed;
};
const rout = useRouter();
const send = async () => {

```

```

const final = finalUpdate();
if (final.title &&
    final.problemRequire &&
    final.problemInputs.length &&
    final.problemOutputs.length &&
    final.funtionProblemModel
) {const id = getCookie("id")?.toString();
    const api = await fetch(
        "/api/handleNewExercicesApi",
        sendToServerCookies(JSON.stringify(final), id)
    );
    const { text, ok } = await api.json();
    if (ok) {
rout.push(`/CoursView/${professorEmail}/${courseName}/${text}/view`);
        } else {
            notFound();
        }
    } else {
        setIsAllRight(false);
        setWarning("Please fill all the fields correctly.");});
const sendUpdate = async () => {
    if (!isAllRight) {
        return;
    }
    const id = getCookie("id")?.toString();
    const exercicesUpdate: any = finalUpdate();
    delete exercicesUpdate.urlName;
    exercicesUpdate["courseName"] = courseName;
    exercicesUpdate["oldTitle"] = exercicesName;
    const api = await fetch(
        "/api/handleUpdateCourseApi/handleUpdateExercicesApi",
        sendToServerCookies(JSON.stringify(exercicesUpdate), id)
    );
    const { ok } = await api.json();
    if (!ok) notFound();
    if (setDialog !== undefined) {
        setDialog(undefined);});
const handleInputChange = (event: { target: { name: any; value: any

```

```

} }) => {
  setExercices({
    ...exercices,
    [event.target.name]: event.target.value,}));};
const handleTextareaChange = (index: any) => (event: any) => {
  const newExemples = [...exemples];
  newExemples[index] = event.target.value;
  setExemples(newExemples);
  setExercices({
    ...exercices,
    problemExemples: newExemples,}));};
return (<div className="flex overflow-auto flex-col w-full max-w-2xl
p-8 bg-white border rounded-lg shadow sm:p-12 md:p-16">
  <h5 className="text-2xl font-medium text-gray-900">Upload
video</h5>
  <ProblemsInput
    handleInputChange={handleInputChange}
    nameOfInputs="Problems name" name="title"/>
  <ProblemsInput
    handleInputChange={handleInputChange}
    nameOfInputs="Problems Requere"
    name="problemRequire"/>
  {exemples.map((exemple, index) => (
    <textarea key={index} placeholder="Exemples"
      value={exemple}
      onChange={handleTextareaChange(index)}
      cols={10} rows={5}
      className="block p-2.5 w-full text-sm text-gray-900 bg-gray-
50 rounded-lg border border-gray-300 focus:ring-blue-500 focus:border-
blue-500 dark:bg-gray-700 dark:border-gray-600 dark:placeholder-gray-
400 dark:text-white dark:focus:ring-blue-500 dark:focus:border-blue-
500 mt-4"
    >
      {exemple}
    </textarea>)))}
  <Button color="blue" onClick={handleExemples} className="mt-4">
    Add exemples
  </Button>

```

```

<DropDownSearch
  index={0}
  combineParams={funtionInputs}
  setCombineParams={setFuntionInputs}
  nameOfSearch="Funtion name"
  options=["int", "float", "string", "double", "char"]}
/><div>
  {inputs &&
    inputs.map((_, index: number) => (
      <DropDownSearch key={index} index={index}
        combineParams={combineParams}
        setCombineParams={setCombineParams}
        nameOfSearch="Parameters"
        options=["int","float","string","char","double",
          "int[]","float[]","string[]","char[]","double[]",,]}/>
    )))
</div>
<Button color="blue" onClick={handleInputs} className="mt-4">
  Add Parameters</Button>
<div>
  <FileInput
    id="file-upload"
    className="mt-4"
    onChange={(e: any) => {
      const file = e.target.files[0];
      if (file.type ===
        "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet" ||
        file.type === "application/vnd.ms-excel"
      ) {setIsAllRight(true);
        hndleExel.readExcel(file, setItems);
      } else {
        setIsAllRight(false);
        setWarning("Please upload a file with excel format.");
      }}}/></div>
  {!isAllRight ? (
    <Alert color="failure" icon={HiInformationCircle}
      className="mt-4">

```

```

        <span className="font-medium">{warning}</span>
      </Alert>) : (<></>))}
    <Link
      href={` /ExercicesTutorial` }
      className="mt-4 inline-block font-medium text-blue-600
dark:text-blue-500 hover:underline hover:text-red-600"
    >Tutorial</Link>
    <Button color="blue"
      onClick={!isUpdated ? send : sendUpdate}className="mt-4"
    >
      Send</Button></div>));};

```

- Clasa ExelHandle

Este o clasă utilitară care facilitează citirea datelor din fișiere Excel în cadrul componentei ExercicesComponent. Aceasta are două metode principale: readExcel și readExcelEmail.

Metoda readExcel este utilizată pentru a citi datele dintr-un fișier Excel și a le stoca într-un format adecvat pentru a fi utilizat în componenta React. Datele sunt citite și transformate într-o matrice, unde fiecare element al matricei reprezintă o coloană din fișierul Excel. Această metodă folosește FileReader pentru a citi conținutul fișierului Excel ca un buffer de date, apoi utilizează biblioteca XLSX pentru a parsa buffer-ul de date și a-l transforma într-un obiect workbook.

```

export class ExelHandle {
  public readExcel = (
    file: any,
    setItems: Dispatch<SetStateAction<never[]>>
  ) => {
    const promise = new Promise((resolve, reject) => {
      const fileReader = new FileReader();
      fileReader.readAsArrayBuffer(file);
      fileReader.onload = (e: any) => {
        const bufferArray = e.target.result;
        const wb = XLSX.read(bufferArray, {
          type: "buffer",
        });
        const wsname = wb.SheetNames[0];
        const ws = wb.Sheets[wsname];
        let data: any = XLSX.utils.sheet_to_json(ws, { header: 1 });

```



```

        data = data[0].map((_: any, i: any) => data.map((row: any) =>
row[i]));
        resolve(data);
    };
    fileReader.onerror = (error) => {
        reject(error);
    };
});
promise.then((d: any) => {
    setItems(d);
});
};
public readExcelEmail = (
    file: any,
    setItems: Dispatch<SetStateAction<never[][]>>
) => {
    const promise = new Promise((resolve, reject) => {
        const fileReader = new FileReader();
        fileReader.readAsArrayBuffer(file);
        fileReader.onload = (e: any) => {
            const bufferArray = e.target.result;
            const wb = XLSX.read(bufferArray, { type: "buffer" });
            const wsname = wb.SheetNames[0];
            const ws = wb.Sheets[wsname];
            const data = XLSX.utils.sheet_to_json(ws, { header: 1 });

            const firstColumn = data.map((row: any) => row[0]); // Extract
the first column
            const c = firstColumn.filter((value: any) => value !==
undefined);
            resolve(c);
        };
        fileReader.onerror = (error) => {
            reject(error);});});
    promise.then((d: any) => {
        const c = d.filter((value: any) => value !== undefined);
        setItems(c);});});
}

```

4.2.5 Modulul Exercises

Backend:

- Functia chooseCompiler:

Această funcție primește datele utilizatorului (userData) care conțin informații despre limbajul de programare selectat. Funcția creează o instanță a clasei HandleProgrammingLanguage, care se ocupă de gestionarea specifică a limbajului. Apoi, apelează metoda chooseLanguage() din această instanță pentru a obține scriptul inițial pentru compilare. În caz de eroare, se înregistrează eroarea în consolă și se aruncă mai departe.

```
async chooseCompiler(userData: ICompilerUser) {
  try {
    const handleProgrammingLanguage: IHandleProgrammingLanguage =
      new HandleProgrammingLanguage(userData);
    return handleProgrammingLanguage.chooseLanguage();
  } catch (error) {
    console.error(error);
    throw error; }}}
```

- Functia executeScripts:

Această funcție primește datele utilizatorului (userData) și inputul (input) pentru codul ce urmează a fi executat. Similar cu funcția anterioară, creează o instanță a clasei HandleProgrammingLanguage. Setează inputul și apoi apelează metoda executeScripts() pentru a executa scriptul și a obține rezultatul. Erorile sunt înregistrate în consolă și aruncate mai departe.

```
async executeScripts(userData: ICompilerUser, input: string) {
  try {
    const handleProgrammingLanguage: IHandleProgrammingLanguage =
      new HandleProgrammingLanguage(userData);
    handleProgrammingLanguage.setInputOutputs(input);
    return handleProgrammingLanguage.executeScripts();
  } catch (error) {
    console.error(error);
    throw error; }}}
```

Enpoint-urii:

- Endpoint-ul chooseCompilerEndPoint:

Acest endpoint GET este responsabil pentru alegerea compilatorului potrivit în funcție de limbajul de programare selectat. Extrage informațiile despre curs, profesor și limbaj din parametrii URL. Apoi, utilizează serviciul compilerService pentru a obține detaliile cursului de la profesorul specificat și apelează funcția chooseCompiler pentru a obține scriptul inițial corespunzător limbajului. Returnează scriptul către frontend. În caz de eroare, afișează un mesaj de eroare corespunzător.

```
@Get('/:professor/:courseName/:language/:id/compile')
async chooseCompilerEndPoint(
  @Param('language') language: string,
  @Param('id') id: string,
  @Param('professor') professor: string,
  @Param('courseName') courseName: string,
): Promise<string> {
  try {
    const course: ICompilers =
      await this.compilerService.getCoursFromProfessor(
        professor,
        courseName,
        id,
      );

    const choose = await this.compilerService.chooseCompiler(
      this.makeCompilerDto(course, language, ''),
    );
    return choose;
  } catch (error) {
    console.error(error);
    throw new Error('An error occurred while choosing the
compiler');
  }
}
```

- Endpoint-ul getCompilerFormat:

Acest endpoint GET returnează formatul compilatorului pentru un exercițiu specific. Extrage

informațiile despre curs și profesor din parametrii URL. Folosește compilerService pentru a obține detaliile cursului și returnează un obiect conținând titlul, cerința, exemplele și formatul exercițiului. În caz de eroare, afișează un mesaj de eroare corespunzător.

```
@Get('/:professor/:courseName/:id/get/exercices/format')
async getCompilerFormat(
  @Param('courseName') courseName: string,
  @Param('professor') professor: string,
  @Param('id') id: string,
) {
  try {
    const course: ICompilers =
      await this.compilerService.getCoursFromProfessor(
        professor,
        courseName,
        id,
      );
    return {
      title: course.title,
      problemRequire: course.problemRequire,
      problemExemples: course.problemExemples,
      format: course.format,
    };
  } catch (error) {
    console.error(error);
    throw new Error('An error occurred while getting the compiler
format');}}}
```

- Endpoint-ul executeScripts:

Acest endpoint POST execută scriptul trimis de utilizator și verifică dacă rezultatul este corect. Extrage informațiile despre curs, profesor, limbaj și script din parametrii URL și corpul cererii. Obține intrările și ieșirile așteptate din detaliile cursului. Execută scriptul pentru fiecare intrare și compară rezultatul cu ieșirea așteptată. Returnează un obiect care indică dacă algoritmul este corect și detalii despre intrare, ieșire proprie și ieșire așteptată în caz de eroare. În caz de eroare la execuție, afișează un mesaj de eroare corespunzător.

```
@Post('/:professor/:courseName/:language/:id/execute/script')
```

```

async executeScripts(
  @Body() scripts: { script: string },
  @Param('language') language: string,
  @Param('id') id: string,
  @Param('professor') professor: string,
  @Param('courseName') courseName: string,
) {
  try {
    const course: ICompilers =
      await this.compilerService.getCoursFromProfessor(
        professor,
        courseName,
        id,
      );
    const output = this.takeParameterValue(course.problemOutputs,
'Output');

    const inputs = this.takeParameterValue(course.problemInputs,
'Input');
    for (let index = 0; index < inputs.length; index++) {
      const input = inputs[index];
      const exec = await this.compilerService.executeScripts(
        this.makeCompilerDto(course, language, scripts.script),
        input,
      );
      const e = exec.toString().replace('\r\n', '');
      if (e.replace('\n', '') !== output[index]) {
        return {
          isAlgorithmOk: false,
          input: input,
          yourOutput: e,
          expected: output[index],};}}
    return {
      isAlgorithmOk: true,
      input: '',yourOutput: '',
      expected: '',};} catch (error) {
      console.error(error);
      throw new Error('An error occurred while executing the

```

```
scripts');}}
```

Clasa HandleProgrammingLanguage

Această clasă este responsabilă de manipularea specifică a limbajelor de programare, folosind un model de design strategy cu clase precum PythonCompier, CppCompier, JavaScriptCompier și JavaCompier. Funcția chooseLanguage selectează o implementare specifică limbajului de programare ales și generează un script inițial. Funcția executeScripts apelează metoda execute a implementării specifice pentru a executa codul și a obține rezultatul.

```
export class HandleProgrammingLanguage implements
IHandleProgrammingLanguage {
  private userData: ICompilerUser;
  public constructor(userData) {
    this.userData = userData;
  }
  private input;
  setInputOutputs(input: string): void {
    this.input = input;
  }
  public async chooseLanguage() {
    let programs: IPythonCompier;
    switch (this.userData.programmingLanguage) {
      case 'python':
        programs = new PythonCompier();
        programs.setPatern(
          this.userData.functionName,
          this.userData.parameterWithType,);
        programs.generatePythonFuntion();
        return programs.getScripts();
      case 'cpp':
        programs = new CppCompier();
        programs.setPatern(
          this.userData.functionName,
          this.userData.parameterWithType,);
        programs.generatePythonFuntion();
        return programs.getScripts();
      case 'javaScript':
        programs = new JavaScriptCompier();
```

```

        programs.setPatern(
            this.userData.functionName,
            this.userData.parameterWithType,);

        programs.generatePythonFuntion();
        return programs.getScripts();
    case 'java':
        programs = new JavaCompier();
        programs.setPatern(
            this.userData.functionName,
            this.userData.parameterWithType,);
        programs.generatePythonFuntion();
        return programs.getScripts();}}
public async executeScripts(): Promise<unknown> {
    let programs:
        | IPythonCompier| ICppCompier| IJavaScriptCompier
        | IJavaCompier;
    switch (this.userData.programmingLanguage) {
        case 'python':
            programs = new PythonCompier();
            programs.setPatern(
                this.userData.functionName,
                this.userData.parameterWithType,);
            programs.setInputOutputs(this.input);
            programs.setScripts(this.userData.scripts);
            return programs.execute();
        case 'cpp':
            programs = new CppCompier();
            programs.setPatern(
                this.userData.functionName,
                this.userData.parameterWithType,);
            programs.setInputOutputs(this.input);
            programs.setScripts(this.userData.scripts);
            return programs.execute();
        case 'javaScript':
            programs = new JavaScriptCompier();
            programs.setPatern(
                this.userData.functionName,

```

```

        this.userData.parameterWithType,);
    programs.setInputOutputs(this.input);
    programs.setScripts(this.userData.scripts);
    return programs.execute();
case 'java':
    programs = new JavaCompier();
    programs.setPatern(
        this.userData.functionName,
        this.userData.parameterWithType,);
    programs.setInputOutputs(this.input);
    programs.setScripts(this.userData.scripts);
    return programs.execute();}}}

```

Clasa CompilerHandler

Această clasă este responsabilă pentru executarea codului într-un limbaj de programare specific. Are metode separate pentru fiecare limbaj (executeCppCode, executeJavaCode, executeJavaScriptCode, executePythonCode). Metodele care folosesc limbaj de programare interpretat cum sunt python si javascript vor executa codul direct in masina virtuala. Dar limbaje de programare ca C++ si java vor scrie intr-un fisier codul il vor trimite catre masinile lor virtuale si vor returna raspunsul.

```

export class CompilerHandler {
    private programingLanguage: string;
    private process: ChildProcessWithoutNullStreams;
    private script: string;
    public constructor(programingLanguage: string, script: string) {
        this.programingLanguage = programingLanguage;
        this.script = script;}
    public executeCppCode = async () => {
        writeFileSync(
            'E:\\Licenta-Platforma-de-invatare-
programare\\backend\\src\\Compiler\\CppFile.cpp',
            this.script,);
        return new Promise((resolve) => {
            const compile = spawn('g++', [
                '-o', 'output',

```



```

        'E:\\Licenta-Platforma-de-invatare-
programare\\backend\\src\\Compiler\\CppFile.cpp',
    ]);
    let result = '';
    compile.on('close', (compileExitStatus) => {
        if (compileExitStatus !== 0) {
            return;
        }
        const run = spawn('./output');
        run.stdout.on('data', (data) => {
            result += `${data}`;
            resolve(result);
        });
        run.stderr.on('data', (data) => {
            result += `${data}`;
            resolve(result);
        });
        run.on('close', (runExitStatus) => {
            if (runExitStatus !== 0) {
                return;
            }
        });
    });
    public executeJavaCode = async () => {
        this.script = this.script.replace(/string/g, 'String');
        writeFileSync(
            'E:\\Licenta-Platforma-de-invatare-
programare\\backend\\src\\Compiler\\JavaFile.java',
            this.script,);
        return new Promise((resolve) => {
            const child = spawn('javac', [
                'E:\\Licenta-Platforma-de-invatare-
programare\\backend\\src\\Compiler\\JavaFile.java',
            ]);
            let result = '';
            child.stdout.on('data', (data) => {
                result += data;
                resolve(1);
            });
            child.stderr.on('data', (data) => {
                result += data;
                resolve(
                    result.replace(
                        'E:\\Licenta-Platforma-de-invatare-
programare\\backend\\src\\Compiler\\JavaFile.',

```

```

        ' ',),,);});
child.on('close', (code) => {
  if (code === 0) {
    const run = spawn('java', [
      '-cp','E:\\Licenta-Platforma-de-invatare-
programare\\backend\\src\\Compiler',
      'JavaFile',]);
    run.stdout.on('data', (data) => {
      resolve(data);});
    run.stderr.on('data', (data) => {
      resolve(data);
    });});});});
public executeJavaScriptCode = async () => {
  return new Promise((resolve, reject) => {
    const command = `node -e "${this.script.replace(/\n/g, ' ')}"`;
    exec(command, (error, stdout, stderr) => {
      if (error) {
        console.error(`exec error: ${error}`);
        reject(`exec error: ${error}`);
        return;}
      if (stderr) {
        console.error(`stderr: ${stderr}`);
        reject(`stderr: ${stderr}`);
        return;}
      resolve(stdout);});});});
public executePythonCode = async () => {
  return new Promise((resolve) => {
    let result = '';
    this.process = spawn(this.programingLanguage, ['-c',
this.script]);
    this.process.stdout.on('data', (data) => {
      result += data.toString();});
    this.process.stderr.on('data', (data) => {
      console.error(`stderr: ${data}`);
      result += `${data}`;
      resolve(data.toString());});
    this.process.on('close', () => {
      resolve(result);});});});});

```

- Clasa PythonCompier

Această clasă implementează interfața IPythonCompier și gestionează compilarea și execuția codului Python. Atributele sale private stochează limbajul de programare ("python"), numele funcției, parametrii cu tipurile lor, scriptul complet și intrările.

Metodele publice permit setarea intrărilor, a numelui și parametrilor funcției, a scriptului complet și returnarea scriptului. Metoda generatePythonFuntion generează definiția funcției Python pe baza numelui și parametrilor furnizați.

Metoda execute este responsabilă de execuția codului. Adaugă o linie de cod pentru a apela funcția cu intrările specificate și a afișa rezultatul. Apoi, creează o instanță a clasei CompilerHandler pentru a executa codul Python și returnează rezultatul execuției.

```
export class PythonCompier implements IPythonCompier {
  private programingLanguage: string = 'python';
  private funtionName: string;
  private parameterWithType: JSON;
  private script: string;
  private inputs: string;
  setInputOutputs(input: string): void {
    this.inputs = input;}
  public setPatern(funtionName: string, parameterWithType: JSON) {
    this.funtionName = funtionName;
    this.parameterWithType = parameterWithType;
  }
  public setScripts(script: string) {
    this.script = script;}
  getScripts(): string {
    return this.script;}
  public generatePythonFuntion() {
    const keys = Object.keys(this.parameterWithType);
    let stringKey: string = '';
    keys.forEach((k: string) => {
      stringKey += ',' + k;
    });
    this.script = `def
${this.funtionName.split('.')[1]}(${stringKey.substring(
```

```

        1,))):`;};
    public async execute() {
        const nameOfFunction = this.funtionName.split('.')[1].replace('def', '');
        this.script += `\n\nprint(${nameOfFunction}(${this.inputs}))`;
        const compilerHandler: ICompilerHandler = new CompilerHandler(
            this.programingLanguage,
            this.script,);
        return compilerHandler.executePythonCode();}}

```

Frontend:

- Componenta CompilerViewComponents

Această componentă React este responsabilă pentru afișarea interfeței utilizatorului (UI) pentru vizualizarea și compilarea exercițiilor de programare.

`programingLanguageFormat`: Această variabilă stochează codul sursă al exercițiului. Inițial, este setată cu un șablon pentru limbajul Python.

`programmingLanguage`: Această variabilă stochează limbajul de programare selectat de utilizator. Valoarea inițială este "python".

`statusOfCode`: Această variabilă stochează rezultatul compilării și execuției codului. Este un obiect de tipul `IStatutCode` care conține informații despre succesul sau eșecul compilării, precum și intrările, ieșirile și rezultatele așteptate.

`isStarted`: Acest steag boolean indică dacă procesul de compilare este în desfășurare. Valoarea inițială este false.

Funcționalități:

`callProgram()`: Această funcție asincronă este apelată imediat după ce componenta a fost montată în DOM (prin intermediul hook-ului `useEffect` cu un array de dependențe gol). Rolul său este de a obține șablonul inițial de cod pentru limbajul Python de la backend și de a-l seta în variabila de stare `programingLanguageFormat`.

`handleTextareaChange(event)`: Această funcție este apelată ori de câte ori utilizatorul modifică conținutul zonei de text (textarea) în care își scrie codul. Ea actualizează variabila de stare

programingLanguageFormat cu noul conținut al zonei de text.

handleLanguageFormat(language): Această funcție asincronă este apelată atunci când utilizatorul selectează un alt limbaj de programare din interfață. Ea actualizează variabila de stare programmingLanguage cu noul limbaj selectat. Apoi, trimite o cerere către un endpoint API pe backend (/api/handleExercicesApi) pentru a obține șablonul de cod corespunzător noului limbaj. Dacă cererea are succes, șablonul primit de la backend este setat în variabila de stare programingLanguageFormat. În caz contrar, este apelată funcția notFound(), care probabil gestionează cazul în care resursa nu a fost găsită.

handleCompile(): Această funcție asincronă este apelată când utilizatorul apasă butonul de compilare. Ea începe prin a seta variabila isStarted la true pentru a indica faptul că procesul de compilare este în desfășurare și resetează variabila statusOfCode. Apoi, construiește un obiect option care conține codul sursă (programingLanguageFormat), limbajul de programare (programmingLanguage), ID-ul cursului (idCourses), numele profesorului (professor) și numele cursului (courseName). Acest obiect este trimis într-o cerere POST către un alt endpoint API pe backend (/api/handleCompileApi) pentru a fi compilat și executat. După ce primește răspunsul de la backend, funcția actualizează variabila de stare statusOfCode cu rezultatul și setează isStarted la false pentru a indica faptul că procesul de compilare s-a încheiat. În cazul în care apare o eroare în timpul compilării sau execuției, este apelată funcția notFound().

Randarea interfeței:

Componenta returnează un element div care reprezintă structura principală a interfeței. Acest element conține două componente principale:

ExercicesRequerment: Această componentă afișează cerința exercițiului, exemplele și alte informații relevante pentru utilizator.

CompilerTextArea: Această componentă afișează zona de text în care utilizatorul poate scrie codul sursă, butoanele pentru selectarea limbajului de programare și pentru compilare, precum și mesajele de stare sau eroare.

În funcție de valorile variabilelor de stare statusOfCode și isStarted, componenta afișează fie un mesaj de succes (SuccessAlert), un mesaj de eroare (FailAlert), sau un indicator de încărcare (LoadingStatus).

```

export const CompilerViewComponents: FC<{
  title: string;
  problemRequire: string;
  problemExemples: string[];
  idCourses: number;
  format: "Compiler";
  professor: string;
  courseName: string;
}> = ({title,problemRequire,problemExemples,idCourses,professor,
  courseName,
}) => {
  const [programingLanguageFormat, setProgramingLanguageForrmat] =
    useState<string>();
  const [programmingLanguage, setProgrammingLanguage] =
    useState<string>("");
  const callProgram = async () => {
    const porgamm = await handleLanguageFormat("python");
    setProgramingLanguageForrmat(porgamm);};
  useEffect(() => {
    callProgram();}, []);
  const handleTextareaChange = (event: any) => {
    setProgramingLanguageForrmat(event.target.value);};
  const handleLanguageFormat = async (language: string):
    Promise<string> => {
    setProgrammingLanguage(language);
    const option = {
      language: `${language}`,
      id: `${idCourses}`,
      professor: `${professor}`,
      coursName: `${courseName}`,};
    const api = await fetch(
      "/api/handleExercicesApi",
      sendToServerCookies(JSON.stringify(option), undefined));
    const { text, ok } = await api.json();
    if (ok) {
      return text.replace('undefined', '');
    } else {
      notFound();};};

```

```

const [statusOfCode, setStatusOfCode] = useState<IStatutCode>();
const [isStarted, setIsStarted] = useState<boolean>(false);
const handleCompile = async (): Promise<void> => {
  setIsStarted(true);
  setStatusOfCode(undefined);
  const option = {
    language: `${programmingLanguage}`,
    id: `${idCourses}`,
    professor: `${professor}`,
    coursName: `${courseName}`,
    script: `${programingLanguageForrmat}`,
  };
  try {
    const api = await fetch(
      "/api/handleCompileApi",
      sendToServerCookies(JSON.stringify(option), undefined) );
    const response = await api.json();
    setStatusOfCode(response);
  } catch (error) {
    notFound();
  } finally {
    setIsStarted(false);
  }
};
return (
  <div className="flex flex-wrap overflow-hidden flex-row p-8 bg-
white border rounded-lg shadow sm:p-12 md:p-16 w-screen h-screen">
    <div className="flex flex-col overflow-auto w-1/2 h-screen">
      <ExercicesRequerment
        title={title}
        problemRequire={problemRequire}
        problemExemples={problemExemples}/>
    </div>
    <div className="flex flex-col w-1/2 h-screen">
      <CompilerTextArea
        programingLanguageForrmat={programingLanguageForrmat}
        handleTextareaChange={handleTextareaChange}
        setProgramingLanguageForrmat={setProgramingLanguageForrmat}
        handleLanguageFormat={handleLanguageFormat}
        handleCompile={handleCompile}/>
      {statusOfCode !== undefined ? (

```

```

        statusOfCode.isAlgorithmOk === true ? (
            <SuccessAlert />
        ) : (
            <FailAlert statusOfCode={statusOfCode} />)
    ) : isStarted ? (
        <LoadingStatus />
    ) : undefined}
</div>
</div>
);
};

```


5 Concluzii

În concluzie, proiectul a atins obiectivele propuse, realizând o platformă web completă pentru învățarea programării, utilizând cele mai recente tehnologii. Aceasta platformă oferă studenților și profesorilor un mediu eficient și intuitiv pentru educație și colaborare.

Realizarea Obiectivelor

Crearea unei aplicații web full-stack cu MongoDB:

Aplicația utilizează MongoDB pentru stocarea și gestionarea datelor, asigurând o performanță optimă și scalabilitate.

Design simplu și intuitiv:

Interfața utilizatorului este proiectată pentru a fi rapidă și ușor de utilizat, permițând utilizatorilor să navigheze și să îndeplinească sarcini fără dificultăți.

Alegerea precisă a rolurilor:

Platforma definește clar rolurile utilizatorilor (student, profesor, administrator) și implementează controlul accesului în funcție de aceste roluri, asigurând securitatea și personalizarea experienței utilizatorului.

Funcționalități pentru administratori:

Administratorii au posibilitatea de a gestiona utilizatorii, inclusiv adăugarea și eliminarea profesorilor, precum și administrarea cursurilor acestora.

Funcționalități pentru profesori:

Profesorii pot crea și gestiona cursuri, încărca materiale educaționale (videoclipuri, PDF-uri, exerciții), și seta accesul cursurilor (public sau privat). De asemenea, pot adăuga studenți la cursuri private și desemna colaboratori.

Accesul studenților:

Studenții pot accesa materialele de curs, viziona tutoriale video, citi PDF-uri și rezolva exerciții folosind un compilator integrat. Platforma salvează automat soluțiile corecte ale studenților în baza de date.

5.1 Realizarea obiectivelor

În mare parte, în proiect, obiectivele principale au fost atinse, dar câteva lucruri care ar fi putut fi mai bine făcute ar fi:

Modificarea cursurilor de către profesori ar fi fost de preferat să se facă pe baza cheii unice din MongoDB, nu pe baza numelui.

Adăugarea mai multor limbaje de programare ar fi fost benefică pentru diversificarea și îmbunătățirea platformei.

6 Bibliografie

6.1 Webliografie

Nextjs docs:

[WWW01] What is Next.js - https://nextjs.org/docs?utm_source=create-next-app&utm_medium=appdir-template&utm_campaign=create-next-app#:~:text=Next.js%20is,fast%20React%20applications.

[WWW02] Main Features - https://nextjs.org/docs?utm_source=create-next-app&utm_medium=appdir-template&utm_campaign=create-next-app#:~:text=Some%20of%20the,and%20type%20checker.

[WWW03] Routing Fundamentals - [Building Your Application: Routing | Next.js \(nextjs.org\)](https://nextjs.org/docs/app/building-your-application/routing)

[WWW04] Creating Routes - <https://nextjs.org/docs/app/building-your-application/routing/defining-routes#creating-routes:~:text=In%20this%20example,other%20colocated%20files>.

[WWW05] Dynamic Routes <https://nextjs.org/docs/app/building-your-application/routing/dynamic-routes#:~:text=When%20you%20don%27t%20know%20the%20exact%20segment%20names%20ahead%20of%20time%20and%20want%20to%20create%20routes%20from%20dynamic%20data%2C%20you%20can%20use%20Dynamic%20Segments%20that%20are%20filled%20in%20at%20request%20time%20or%20prerendered%20at%20build%20time>.

[WWW06] Api Router - <https://nextjs.org/docs/pages/building-your-application/routing/api-routes#:~:text=API%20routes%20provide,side%20bundle%20size>.

[WWW07] Server Rendering - <https://nextjs.org/docs/app/building-your-application/rendering/server-components#:~:text=React%20Server%20Components%20allow%20you%20to%20write%20UI%20that%20can%20be%20rendered%20and%20optionally%20cached%20on%20the%20server.%20In%20Next.js%2C%20the%20rendering%20work%20is%20further%20split%20by%20route%20segments%20to%20enable%20streaming%20and%20partial%20rendering%2C%20and%20there%20are%20three%20different%20server%20rendering%20strategies%3A>

[WWW08] Client Rendering - <https://nextjs.org/docs/app/building-your-application/rendering/client-components#:~:text=Client%20Components%20allow,might%20use%20them>.

Tailwind:

[WWW01] [Installation - Tailwind CSS](#)

Flowbite:

[WWW01] [Flowbite - Tailwind CSS component library](#)

Flowbite react:

[WWW01] [Flowbite React - UI Component Library \(flowbite-react.com\)](#)

Next Video:

[WWW01] [Docs | next-video](#)

React-pdf-viewer:

[WWW07] [Getting started - React PDF Viewer \(react-pdf-viewer.dev\)](#)

Nestjs:

[WWW08] [Documentation | NestJS - A progressive Node.js framework-](#)

[WWW02] Controllers -

[https://docs.nestjs.com/controllers#:~:text=A%20controller%27s%20purpose,the%20corresponding%20controllers\).](https://docs.nestjs.com/controllers#:~:text=A%20controller%27s%20purpose,the%20corresponding%20controllers).)

[WWW03] Modules-

<https://docs.nestjs.com/modules#:~:text=Each%20application%20has,of%20capabilities.>

[WWW04] Guards-

[https://docs.nestjs.com/guards#:~:text=Guards%20have%20a,and%20its%20metadata\).](https://docs.nestjs.com/guards#:~:text=Guards%20have%20a,and%20its%20metadata).)

JWT:

[WWW01] [Authentication | NestJS - A progressive Node.js framework-](#)

Exceljs:

[WWW01] [exceljs - npm \(npmjs.com\)](https://www.npmjs.com/package/exceljs)

JavaScript:

[WWW01] [JavaScript | MDN \(mozilla.org\)](https://developer.mozilla.org/en-US/docs/Web/JavaScript)

TypeScript:

[WWW01] [TypeScript: The starting point for learning TypeScript \(typescriptlang.org\)](https://www.typescriptlang.org/)-

Node/Express:

[WWW01] [Express/Node introduction - Learn web development | MDN \(mozilla.org\)](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Node/Express_introduction)

Mongodb:

[WWW01] [MongoDB Documentation](https://www.mongodb.com/docs/)