



**Universitatea
Transilvania
din Brașov**

**FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ**

LUCRARE DE LICENȚĂ

Conducător științific: Lector Dr. Vasilescu Anca

Absolvent: Bolba-Mateescu Andrei-Ioan

Brașov

Iunie 2023



**Universitatea
Transilvania
din Brașov**

**FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ**

**UNIVERSITATEA TRANSILVANIA DIN BRAȘOV
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII: INFORMATICĂ**

InternHub - PLAFORMĂ DE MONITORIZARE A EVOLUȚIEI STUDENȚILOR IN STAGIUL DE PRACTICĂ

Coordonator științific: Lector Dr. Vasilescu Anca

Absolvent: Bolba-Mateescu Andrei-Ioan

Brașov

Iunie 2023

Cuprins

1. Introducere	3
1.1. Scurtă descriere a temei	3
1.2. Motivația alegere lucrării	4
1.3. Contribuții personale	5
1.4. Structura lucrării	8
2. Abordarea temei	9
2.1. Scopul aplicației	9
2.2. Alte abordări în domeniu	13
3. Managementul entităților	13
3.1. Modelarea entităților	14
3.2. Faza de implementare a bazei de date	15
3.3. Managementul identităților și al utilizatorilor	17
4. Arhitectura aplicației	19
4.1. Separarea preocupărilor	19
4.2. Arhitectura REST pentru designul unui API	23
4.3. CRUD folosind verbe HTTP	24
4.4. Statusuri de răspuns	26
4.5. Modelul “Clean Architecture” definit de Uncle Bob	27
4.6. Securitate	29
5. Noțiuni teoretice și tehnologii folosite	31
5.1. .NET 7	31
5.2. Angular	36
5.3. Microsoft SQL Server	41
5.4. Securitatea datelor	44
5.5. Cloudinary	44
6. Prezentarea aplicației	47
6.1. Introducere	47
6.2. Structura aplicației	48
6.3. Prezentarea aplicației	49
7. Concluzii și perspective de viitor	65
7.1. Concluzii	65
7.2. Posibilități de extindere	65

1. Introducere

1.1. Scurtă descriere a temei

Primul pas pe care un student îl face pentru a începe o carieră în domeniul IT este, de cele mai multe ori, un internship. Și asta se poate observa foarte ușor pe piața job-urilor în domeniul dezvoltării software. Fie că vorbim de site-uri precum eJobs, BestJobs sau chiar LinkedIn, se pot observa multe astfel de oferte de internship de la companii care caută studenți pe care să îi învețe ce au nevoie pentru a putea lucra în compania respectivă. Evident nu toți studenții ce reușesc să înceapă un internship reușesc să și obțină un post în acea companie. Acest lucru este influențat și de procesul de monitorizare a evoluției studenților, ce este efectuat de un mentor. Acest mentor le este atribuit internilor în prima zi de lucru la noua companie.

Astfel tema aleasă și anume InternHub vine în ajutorul mentorilor și al managerilor pentru o mai bună urmărire a evoluției studenților în practică și pentru a ușura alegerea când vine vorba de oferirea unei oferte în cadrul companiei. InternHub este o aplicație unde mentorii au grupuri cu studenți asigurați lor și pot comunica mai ușor și pot vedea evoluția lor ajutându-i prin a le răspunde la întrebări.

Lucrarea își propune facilitarea procesului de urmărire și totodată înțelegere a nivelului de însușire a noilor informații pentru fiecare student în parte prin oferirea de task-uri periodice, la care se adaugă provocările zilnice pe care le oferă mentorii studenților pe care îi au sub observare. Sistemul de provocări zilnice are la bază de motivare prin procesul de *gamification*. Acest proces utilizează principii precum competiția, recompensele, punctele, clasamentele, provocările și elementele conexe pentru a crește motivația și participarea activă la completarea task-urilor. Astfel, fiecare provocare dusă la bun sfârșit reprezintă un punct câștigat. La finalul luni se face un clasament pentru a vedea cine a fost cel mai activ. Prin acest proces de *gamification* studenții se dezvoltă, iar mentorii văd rezultatele, le interpretează și sunt mai aproape de o decizie obiectivă de ofertă la angajare pentru un student la finalul programului de internship.

În plus, aplicația dispune de un sistem de forum, asemănător lui StackOverflow, unde atât studenți, mentori cât și administratori pot interveni cu întrebări, nelămuriri sau chiar curiozități, iar ceilalți utilizatori, indiferent de rol pot adăuga răspunsuri. Pentru a ști care răspunsuri sunt bune și care sunt mai puțin bune, toate postările și răspunsurile pot fi votate în consecință cu *upvote*, dacă se consideră o întrebare sau un răspuns relevant, respectiv *downvote* dacă nu este considerat relevant. Astfel, postarea are un punctaj dat de diferența: *upvotes* – *downvotes*, iar astfel ne putem da seama de calitatea postării.

Tot prin intermediul aplicației se pot planifica ședințe, atât pentru grupuri, cât și individual. Platforma de susținere a acestor ședințe este la alegerea companiei care va utiliza în

viitor această aplicație, sistemul din cadrul acestui soft are nevoie doar de un link către platforma aleasă, fie ea Google Meet, Zoom sau Teams. Toate au în comun acest avantaj al link-ului pentru a intra într-o conferință.

Un alt feature cheie al aplicației este sistemul de chat, care facilitează comunicarea între utilizatori, putând să fie făcute și grupuri de discuții. Comunicarea este un aspect important în cadrul dezvoltării tinerilor aflați la început de drum în acest domeniu, astfel InternHub vine în ajutorul lor prin aceste sistem de comunicare în timp real între utilizatori.

Toate aceste caracteristici fac din InternHub nu doar o altă aplicație în domeniu, ci una care are un scop direct de legat de evoluția în carieră a studenților și care dispune de uneltele necesare pentru a aduce plus valoare în parteneriatele dintre universități și mediul economic.

1.2. Motivația alegere lucrării

Ideea lucrării a plecat de la stagiul meu de practica, unde ca în multe locuri nu este încă implementat un sistem de genul acesta. Acest lucru este determinat și de faptul că fiecare companie decide utilitatea unui astfel de soft. Lipsa unui astfel de sistem s-a simțit deoarece feedback-ul nu vine așa de ușor cum ne-am imagina, iar existența unui portal intern de genul lui InternHub ar fi adus o mare diferență.

Astfel, motivația alegerii acestei teme se regăsește într-un motiv personal, deoarece simt că m-ar fi ajutat enorm când eram în practică. În plus o aplicație de acest tip are o aplicabilitate reală în toate companiile de profil IT și nu numai care au genul acesta de programe de internship, fie ca sunt pentru stagiile de practică, fie ca sunt programe de internship cu scopul pregătirii noilor angajați să lucreze cu uneltele companiei pe proiecte reale folosind modul de lucru din cadrul companiei.

Utilitatea reală în cadrul unei companii este unul din motivele pentru care am ales să gândesc, proiectez, să și să implementez o astfel de aplicație. În plus este o aplicație relativ singulară, după cum vom prezenta mai multe în subcapitolul 2.2, unde o să vedem ce alte abordări mai sunt pe piața și în ce măsură acestea se folosesc pentru rezolvarea acestei probleme particulare.

Un alt motiv pentru care am ales această temă îl reprezintă faptul că nu este doar o aplicație foarte utilă în practică, ci este o aplicație care poate aduce plus-valoare unei companii prin dezvoltarea concomitentă a tuturor angajaților care o folosesc. Este o aplicație dezvoltată pentru stagiile de practica din domeniul IT, dar ce poate fi ușor adaptată și pentru alte stagii de practică din alte domenii precum contabilitate, inginerie etc. Acest lucru este posibil deoarece aplicația nu are atât de multe caracteristici ce se regăsesc doar în domeniul IT, ci are caracteristici ce se regăsesc prin orice domeniu. Un sistem de chat este universal valabil pentru orice companie indiferent de domeniul de activitate. Sistemul de forum este și el valabil universal, diferența fiind subiectele ce se discută prin intermediul lui. Acest lucru nu ține însă de

cod, ci de ce contextul de discuții. Exact la fel este și sistemul de challenge-uri, în care arhitectura aplicației rămâne aceeași, pentru orice arie în care compania activează.

Având în vedere toate cele menționate mai sus, aplicația InternHub este una cu o utilitate relativ restrânsă la prima vedere, dar după o analiză mai amănunțită ne dăm seama că acesta poate fi folosită în mai multe domenii. Acest lucru face din aplicație una foarte utilă adăugând, așa cum am mai spus, plus-valoare în compania în care este utilizată prin mijloacele prezentate mai sus.

Așadar, motivația principală pentru alegerea acestei teme este utilitatea ei în mai multe domenii și outcome-ul pe care acesta îl poate aduce în cadrul unei companii, dacă este folosită la capacitatea ei adevărată. De asemenea merită menționat și că o parte din motivație este redată de experiența personală în cadrul unui internship.

1.3. Contribuții personale

Aplicația are o structură de tip full-stack, formată din Angular ca front-end și .NET ca back-end. În ceea ce privește stocarea datelor, am folosit Microsoft SQL Server. Mai multe detalii despre structura aplicației vor fi discutate în capitolul 6, la subcapitolul 6.2.

Atât pe partea de Angular, .NET, cât și pe cea de Microsoft SQL Server, am analizat, proiectat și dezvoltat componentele necesare pentru o aplicație de acest tip. Aș fi putut alege să folosesc Firebase, dar nu aș fi avut aceeași libertate în ceea ce privește crearea arhitecturii back-end-ului și manipularea cererilor pentru fiecare componentă importantă.

Acum voi considera fiecare dintre cele trei componente pentru a prezenta și explica contribuțiile personale în acest proiect și pentru a justifica de ce am optat această soluție software, în detrimentul altora.

- **Front-end (Angular)**

În cadrul dezvoltării aplicației InternHub, elaborarea componentei de front-end a implicat mai multe responsabilități și a necesitat utilizarea unei varietăți de abilități și cunoștințe din partea mea.

Legat de elementele de UI/UX, m-am folosit de cunoștințele mele pentru a crea o interfață prietenoasă, intuitivă și eficientă pentru utilizator. Acest pas a făcut parte din procesul de creare a unei aplicații de front-end cât mai plăcută și, în același timp, ușor de utilizat.

Una dintre contribuțiile mele importante a fost crearea și gestionarea componentelor Angular și comunicarea între ele, reușind în multe cazuri să evit apelarea de request-uri care ar fi îngreunat aplicația. Am realizat acest lucru prin manipularea DOM-ului (Document Object Model) și prin transferul datelor între componente cu ajutorul uneltelor specifice Angular. Am

utilizat acest cadru de lucru pentru a structura aplicația într-o serie de componente reutilizabile, ceea ce a îmbunătățit claritatea și manevrabilitatea codului.

Am implementat funcționalități ale utilizatorului, asigurându-mă că acestea sunt funcționale și eficiente. Acest lucru a inclus implementarea sistemelor de autentificare și autorizare și a altor funcționalități esențiale. Legat de partea de autorizare și autentificare, vom discuta mai multe în capitolul 5, subcapitolul 5.4 - Securitatea Datelor. Am încercat, pe cât posibil, să evit duplicarea codului prin utilizarea unei componente pentru două sau mai multe sarcini asemănătoare, precum adăugarea și editarea unei entități în baza de date.

Pentru partea de front-end, contribuțiile au fost notabile deoarece este o componentă complexă din punctul de vedere al structurii și design-ului. Legat de comunicarea cu API-ul, acesta s-a bazat integral pe verbe HTTP, despre care vom discuta mai multe în Capitolul 4 - Arhitectura aplicației, la Subcapitolul 4.4 - CRUD folosind verbe HTTP.

- **API (Framework .NET)**

Pentru partea de back-end, am proiectat și dezvoltat un API de la zero, folosindu-mă de design pattern-ul MVC (Model-View-Controller), împreună cu alte design pattern-uri, care au avut ca rezultat un cod curat, ușor de înțeles și de parcurs.

Legat de acest pattern, și anume MVC, vom discuta mai multe în capitolul 4, subcapitolul 4.1, deoarece are un rol esențial în proiectarea unei arhitecturi back-end eficiente, ușor de înțeles, urmând principiile de Clean Code.

O sarcină importantă în cadrul dezvoltării aplicației a fost gândirea și implementarea API-ului, cu scopul de a crea o cale de comunicare între front-end și baza de date. Un aspect ce merită menționat este că m-am asigurat că transmiterea datelor între aceste două componente se execută într-un mod eficient și securizat.

Securitatea este o parte crucială în crearea unui API și, de aceea am acordat o atenție sporită acestui aspect prin implementarea diverselor strategii de securitate, pentru a mă asigura că datele utilizatorilor sunt protejate și că numai utilizatorii autorizați au acces la informații sensibile. Acest lucru este foarte relevant în cadrul aplicației InternHub, deoarece există mai multe tipuri de utilizatori cu drepturi de acces diferite ce implică drepturi de scriere diferite. Acest lucru a inclus utilizarea de token-uri JWT (JSON Web Token), care permite doar utilizatorilor aplicației să aibă acces la resursele oferite de către back-end. Pentru acest pas a fost necesară și implementarea de politici CORS. Legat de drepturile de acces, acestea se verifică în momentul în care cererea ajunge în back-end și în funcție de rangul utilizatorului, se execută cererea sau se întoarce codul de eroare 401 Unauthorized.

Un alt aspect important de care am ținut cont în crearea back-end-ului a fost gestionarea logicii de business și comunicarea cu baza de date. Acest aspect a implicat crearea

și gestionarea entităților și a relațiilor între ele, precum și implementarea operațiunilor CRUD (Create, Read, Update, Delete) pentru manipularea datelor, detalii pe care le-am realizat prin implementarea design pattern-ului repository. Vom discuta mai multe despre acest pattern în Capitolul 4, Subcapitolul 4.1.

Pentru proiectarea bazei de date am folosit Entity Framework, după modelul database-first, ceea ce a dus la faptul că modelele din cadrul API-ului au fost voluminoase, având informații care nu erau necesare în totalitate. Pentru a rezolva acest lucru, am ales să folosesc obiecte de tip DTO (Data Transfer Object), care conțin doar informațiile necesare în absolut toate request-urile, având câte un DTO pentru fiecare model. Evident, având acest tip de obiecte, este necesară și o mapare, care, datorită tool-ului numit AutoMapper din ecosistemul .NET, s-a realizat cu ușurință, fiind necesară doar declararea relațiilor între obiectele mapate.

Crearea unui API de la zero, în locul utilizării back-end-ului de la Firebase, are mai multe avantaje. Printre acestea se numără flexibilitatea modelării entităților și a modelării request-urilor. Modelarea entităților este un punct important în cadrul unui back-end, deoarece putem modela obiectul trimis către front-end în funcție de nevoile specifice fiecărui request. În plus, un API creat de la zero facilitează testarea eventualelor bug-uri prin procesul de debugging, care nu este la fel de facil pentru un API de tipul Firebase.

- **Baze de date (Microsoft SQL Server)**

Așa cum am specificat mai sus, pentru baza de date am utilizat Entity Framework, împreună cu pattern-ul database-first, folosindu-mă de migrări pentru crearea bazei de date. Acest lucru mă ajută să am un management mai bun al bazei de date și facilitează schimbările din baza de date, dacă acestea sunt necesare, fie pentru adăugarea unui câmp nou, fie pentru adăugarea unui feature nou, care necesită o tabelă sau mai multe în plus.

Așa cum am afirmat și mai sus, am utilizat modelul de creare a bazei de date database-first, cu FluentMigrator, ceea ce a însemnat că schema bazei de date a fost generată în cod folosind migrări pentru a crea baza de date în Microsoft SQL Server. Acest lucru a necesitat evident și definirea atentă a relațiilor între tabele prin chei străine.

De asemenea, am implementat diverse tehnici pentru a asigura securitatea datelor. Acest lucru a inclus criptarea datelor sensibile, implementarea de controale de acces la nivel de date și utilizarea de tranzacții pentru a asigura integritatea datelor. Un exemplu concret este stocarea datelor sensibile precum parola utilizatorului, care este stocată sub formă de două valori: valoarea rezultată în urma criptării și cheia după care s-a realizat criptarea. Mai multe despre acest subiect vom discuta în capitolul 5, la subcapitolul 5.4 - Securitatea Datelor.

Preocuparea personală în crearea bazei de date a fost legată de respectarea pașilor de creare a bazei de date, fiecare pas având cerințe și nevoi speciale. Vom trece pe scurt prin pașii parcurși în crearea bazei de date folosite în aplicația InternHub.

Primul pas a fost identificarea nevoilor care se referă la ce tipuri de date sunt necesare pentru fiecare entitate și am ales cea mai bună variantă pentru fiecare tip. Puțin mai problematică a fost stocarea de imagini, dar până la urmă am găsit o soluție bună și pentru această problemă.

Apoi am ajuns la etapa de proiectare conceptuală a bazei de date. În această etapă am definit structura bazei de date, relațiile între tabele și tipurile de relații între tabele. Aici am utilizat modele conceptuale, precum diagramele Entitate-Relație (ER), pentru a defini entitățile din baza de date și relațiile dintre acestea.

După această etapă, am ajuns la proiectarea bazei de date, unde am transformat modelul conceptual într-un model logic de date. Tot aici am definit structura tabelelor, a câmpurilor, a relațiilor și a restricțiilor.

Următorul pas a fost cel de implementare a bazei de date, unde am creat migrările specifice pentru fiecare entitate, urmând modelul Fluent Migrator, despre care am vorbit mai în detaliu în capitolul 3, la subcapitolul 3.2 - Fazele de implementare ale bazei de date. Schema bazei de date a fost implementată, urmând să fie populată cu date de început pentru a putea trece la etapa următoare, și anume faza de testare și optimizare a bazei de date. Aici am testat performanța, securitatea și funcționalitatea bazei de date

Toate aceste etape reprezintă un ciclu continuu, întrucât baza de date a fost modificată pentru a fi în conformitate cu nevoile aplicației, fiind nevoită să evolueze odată cu acestea. Pentru fiecare modificare adusă bazei de date, s-au urmat aceste etape, și de aceea spunem că pașii prezentați mai sus reprezintă un ciclu continuu.

Consider că aportul meu pentru această aplicație este considerabil, iar acest lucru se datorează complexității aplicației.

1.4. Structura lucrării

Lucrarea este împărțită în 8 capitole, care conțin informațiile necesare pentru a acoperi întregul proces de realizare a aplicației InternHub până în momentul actual. Fiecare capitol conține informații utile care m-au ajutat în elaborarea aplicației sau care facilitează înțelegerea acesteia.

Astfel, mai jos sunt prezentate cele 8 capitole, alături de o scurtă prezentare a informațiilor pe care le găsim în fiecare capitol:

- **Capitolul 1:** Introducere, unde se prezintă în mod succint tema aleasă, motivația pentru tema aleasă, urmată de contribuțiile personale și structura lucrării.
- **Capitolul 2:** Abordarea temei, unde se prezintă scopul aplicației și se discută eventuale alte implementări existente pe piață.
- **Capitolul 3:** Managementul entităților, care prezintă baza de date, fazele de implementare și managementul utilizatorilor.

- **Capitolul 4:** Arhitectura aplicației, care prezintă modul în care aplicația este împărțită prin separarea preocupărilor, tipurile de request-uri folosite și modelul Clean Architecture prezentat de Uncle Bob.
- **Capitolul 5:** Tehnologii folosite, este probabil una dintre cele mai interesante părți ale lucrării, deoarece descrie partea practică.
- **Capitolul 6:** Prezentarea aplicației, unde se prezintă mai în detaliu structura aplicației, comunicarea între framework-uri și persistenta datelor.
- **Capitolul 7:** Concluzii și perspective de viitor, unde se ajunge la o concluzie legată de această aplicație și se prezintă perspectivele de viitor ale acestei aplicații.
- **Capitolul 8:** Bibliografie, unde se regăsesc link-uri utile pe care le-am folosit în elaborarea lucrării și a aplicației.

2. Abordarea temei

2.1. Scopul aplicației

Aplicația are ca scop principal facilitarea monitorizării evoluției studenților aflați în internship la o companie. Majoritatea își caută un internship pentru că așa reușesc să câștige niște bani, să obțină experiență în acest domeniu și să își facă practica de specialitate, care este o materie obligatorie în cadrul facultăților. Astfel, la finalizarea internship-ului, studentul primește o foaie care atestă că a efectuat stagiul de practică și după o discuție cu mentorul, acesta reușește, în cazul în care a avut rezultate bune, să obțină un post de programator ajutor până la finalizarea studiilor.

După finalizarea programului de practică și obținerea caietului de practică completat, studentul merge cu acel caiet la profesor, iar acesta pune nota pentru această materie, fără să știe exact dacă rezultatele de pe acel caiet sunt reale sau nu. Astfel, InternHub își propune să includă și profesorul coordonator al practicii de specialitate în acest proces prin accesul la evoluția studentului. Astfel, acesta își poate da seama dacă studentul a avut într-adevăr rezultate bune sau mai puțin bune.

În plus, aplicația își propune nu doar monitorizarea acestor rezultate, ci și îmbunătățirea studenților care activează în compania respectivă, prin feedback-ul oferit de mentor și prin ajutorul celorlalți din cadrul echipei de interni, prin răspunsul proactiv la discuțiile deschise pe forum de către toată lumea.

Scopul forumului este acela de ajutor reciproc în cadrul companiei, prin răspunsul oamenilor mai experimentați la întrebările celor aflați la început de drum. Astfel, aceștia află informații noi pe care le pot citi toți internii din companie. Totodată, ei pot da răspunsuri care vor fi apreciate de cei mai experimentați sau care vor fi corectate pentru a nu continua cu informații eronate.

Acesta oferă o platformă de comunicare și partajare a cunoștințelor, care poate îmbunătăți eficiența și productivitatea. Iată câteva motive specifice pentru care o companie ar putea dori să implementeze un astfel de forum:

- **Partajarea cunoștințelor:** Un sistem de forum ajută la partajarea cunoștințelor care rămân scrise atât timp cât aplicația rulează în cadrul companiei. Un forum de acest gen permite angajaților să pună întrebări și să primească răspunsuri de la colegii lor. Acesta poate fi un mod eficient de a partaja cunoștințele și de a învăța de la ceilalți.
- **Documentarea soluțiilor:** Odată ce o întrebare a fost răspunsă pe forum, acea informație devine disponibilă pentru toți ceilalți angajați, indiferent de statutul lor în companie, fie că sunt interni sau mentori. Aceasta poate ajuta la prevenirea duplicării eforturilor și la asigurarea că soluțiile la probleme comune sunt ușor accesibile.
- **Implicarea angajaților:** Forumul permite angajaților să arate proactivitate când vine vorba de cei aflați la început de drum, ce le pot deveni colegii după această perioadă. Interesul companiei este să dezvolte acești tineri și să rămână cu ei în companie. De aceea, această proactivitate este esențială, iar forumul facilitează acest pas. Acest lucru poate îmbunătăți satisfacția la locul de muncă și poate încuraja angajații să se dezvolte profesional.
- **Creșterea eficienței:** Prin furnizarea de răspunsuri la întrebări comune, forumul ajută la economisirea timpului folosit pentru căutarea soluțiilor la probleme comune și poate reduce stresul angajaților.
- **Îmbunătățirea comunicării:** Forumul ajută la îmbunătățirea comunicării în cadrul companiei în care este utilizat, oferind un spațiu unde angajații pot discuta și colabora pe diverse subiecte.

Sistemul de chat este un sistem întâlnit în aproape orice aplicație de pe acest pământ, deoarece facilitează comunicarea între oameni. Sistemul implementat în aplicația InternHub își propune eficiența în comunicare, având posibilitatea comunicării de tip 1-la-1, dar și în grupuri de interes. Aceasta își dorește să fie un sistem utilizat nu doar pentru partea de muncă din cadrul companiei, ci și pentru partea de timp liber prin crearea unor grupuri cu interese comune, fie că vorbim de sport, modă, artă culinară sau chiar partea auto.

În plus, acest sistem oferă câteva beneficii cheie care îl ajută în îndeplinirea scopului său final, și anume eficiența. Printre aceste puncte tari ale sistemului de chat se numără:

- **Comunicare rapidă:** Chat-ul permite angajaților să comunice rapid unul cu celălalt, fără a fi nevoie să trimită e-mailuri sau să inițieze apeluri telefonice. Aceasta poate îmbunătăți eficiența, deoarece permite angajaților să rezolve rapid problemele și să răspundă la întrebări.
- **Colaborare:** Chat-ul facilitează colaborarea între echipe sau între membrii echipei. Angajații discută despre proiecte, împărtășesc idei și primesc input în timp real de la toți membrii echipei. Acest lucru este foarte important pentru un student aflat în poziție de internship, deoarece are suport din partea unor oameni cu experiență. Astfel, procesul de asimilare al informațiilor devine mult mai facil.

- **Documentarea conversațiilor:** Spre deosebire de conversațiile telefonice sau față în față, chat-urile se păstrează și se pot accesa ulterior. Acest lucru este util pentru urmărirea procesului de învățare al studentului, dar și pentru procesul de monitorizare a unui proiect.
- **Disponibilitate:** Chat-ul permite angajaților să rămână conectați și să comunice eficient, indiferent de locație sau fus orar. Desigur, acest lucru este posibil și prin telefon, dar acesta vine cu anumite costuri care diferă în funcție de locul de unde este inițiat apelul. Există și varianta aplicațiilor de tip WhatsApp, dar unii oameni preferă să separe viața profesională de cea personală, astfel un sistem de chat poate ajuta la acest lucru. Acest lucru poate fi deosebit de util pentru companiile cu angajați sau echipe la distanță.
- **Îmbunătățirea moralului angajaților:** Chat-ul intern din cadrul companiei contribuie la îmbunătățirea moralului și a culturii companiei. Oferă un loc pentru angajați să socializeze și să creeze o atmosferă mai prietenoasă și mai colaborativă, ceea ce duce la un mediu de lucru sănătos, care ajută în timp eficiența și calitatea proiectelor.
- **Suport tehnic intern:** Chat-ul poate fi utilizat pentru a oferi suport tehnic rapid angajaților, în special celor aflați în primele luni într-un context de genul acesta. Fiind prima interacțiune cu mediul de lucru, întrebările care apar în mintea celor aflați la început sunt inevitabile și au nevoie de un răspuns rapid. Chat-ul reușește să facă acest lucru prin răspunsul scris, care rămâne acolo și poate fi o sursă de documentare pentru o altă problemă. Acesta poate fi un canal eficient pentru rezolvarea problemelor tehnice sau pentru a răspunde la întrebări.

Sistemul de challenge-uri zilnice, având un sistem de gamification în spate, în colaborare cu cel de task-uri mai lungi, duce la creșterea eficienței lucrului la proiecte și totodată duce la dorința de acumulare de informații noi prin finalizarea task-urilor.

Sistemul de challenge-uri zilnice se bazează pe ideea completării lor de către interni și oferirea de puncte de către mentor. Aceste puncte se vor folosi la crearea unui clasament, care se actualizează pe zi ce trece și care, la final, poate oferi un premiu primilor 3 angajați. Acest pas este la alegerea companiei, iar implementarea lui duce la motivarea studenților de a fi cât mai competitivi și de a completa cât mai bine acest challenge pentru acumularea de puncte cât mai multe.

Asemănător lui Jira, sistemul de task-uri își propune monitorizarea evoluției studenților prin completarea treptată a lor. Progresul poate fi văzut atât de către mentor, cât și de către profesorul responsabil de stagiul de practică din facultatea din care face parte studentul.

Cum Jira s-a dovedit a fi eficientă de-a lungul timpului, sistemul de task-uri din cadrul aplicației InternHub își propune cât de eficient se poate raportat la Jira. Acest sistem oferă numeroase beneficii, printre care:

- **Urmărirea progresului:** Sistemul de task-uri permite mentorilor și profesorului responsabil să urmărească progresul studentului în cadrul perioadei de practică. Astfel, se pot sesiza anumite lacune pe care studentul le are și se pot remedia

ușor, atât din partea mentorului din cadrul companiei, cât și din partea profesorului.

- **Transparența:** Sistemul de task-uri oferă transparență asupra a ceea ce se lucrează, cine lucrează la ce și care sunt termenele. Aceasta poate duce la o mai bună coordonare și la o mai bună înțelegere a sarcinilor de către toată echipa.
- **Eficiența:** Prin structurarea și organizarea muncii, un sistem de task-uri poate ajuta la creșterea eficienței și la reducerea riscului de a uita sau a neglija sarcini importante.
- **Comunicarea și colaborarea:** Sistemul de task-uri facilitează comunicarea și colaborarea în cadrul unei echipe. Membrii echipei pot adăuga comentarii la sarcini, pot împărtăși actualizări și pot colabora pentru a rezolva probleme.

Sistemul de feedback este un sistem vital pentru evoluția oricărui angajat în cadrul unei companii, deoarece prin feedback oamenii evoluează și pot observa greșeli pe care altfel nu le-ar fi sesizat la fel de ușor.

Un sistem de feedback are numeroase avantaje atât în cadrul unei echipe de lucru la un proiect, cât și în cadrul unui grup de oameni. Printre avantajele sistemului de feedback din cadrul aplicației InternHub se enumeră:

Îmbunătățirea performanței: Feedback-ul regulat ajută oamenii să înțeleagă ce fac bine și unde au nevoie de îmbunătățire. Astfel, un student care nu știe foarte multe despre acest domeniu, fiind la început, poate evolua ușor prin implementarea feedback-ului primit de la oamenii mai experimentați. Acest lucru le permite să-și îmbunătățească abilitățile și să crească în performanță.

- **Dezvoltare profesională:** Feedback-ul ghidează dezvoltarea profesională a oamenilor, indicându-le zonele în care au nevoie de mai multă instruire sau experiență. Astfel, mentori știu unde trebuie să pună accent, iar profesorul responsabil poate folosi aceste informații pentru a ajuta studentul în dezvoltarea sa cât timp mai este în facultate.
- **Rezolvarea problemelor:** Prin feedback, problemele pot fi identificate și rezolvate rapid. Acest lucru poate duce la o îmbunătățire a eficienței și a productivității.
- **Recunoașterea muncii bine făcute:** Un sistem de feedback oferă oportunități pentru recunoașterea și aprecierea oamenilor pentru munca lor bine făcută, ceea ce poate duce la creșterea moralului și a angajamentului față de companie.
- **Comunicare deschisă:** Un sistem de feedback poate ajuta la crearea unei culturi de comunicare deschisă și onestă în cadrul companiei.

Astfel, aplicația nu are ca scop doar monitorizarea procesului de evoluție al unui student și ușurarea alegerii dacă se continuă colaborarea sau nu, ci și dezvoltarea tuturor angajaților care utilizează această aplicație în cadrul companiei.

2.2. Alte abordări în domeniu

Problema monitorizării evoluției angajaților, fie că vorbim de interni sau de angajați, a mai fost abordată la cererea companiei care utilizează aplicația. Este și normal ca aplicațiile de genul acesta să difere fie prin feature-uri, fie prin modul de gândire al arhitecturii componentelor ce o formează. Având în vedere că domeniul IT devine tot mai atractiv, chiar și pentru reconversii profesionale, companiile își doresc să păstreze cei mai buni angajați. Astfel, se apelează la tipul acesta de soft-uri, care nu pot fi găsite pe piață, deoarece fiecare companie are nevoi diferite ce vin cu cerințe diferite și scenarii de utilizare tot mai diversificate.

Astfel, companiile aleg să angajeze studenți talentați cu dorințe de învățare în detrimentul experienței și să-i învețe pentru a putea lucra pentru proiectele companiei. Astfel, internship-ul este practic ca un curs pe o perioadă stabilită de la început și are ca finalitate simularea unui proiect de demo. În tot acest timp, este necesară o monitorizare a evoluției, iar pentru acest lucru este necesar un soft de monitorizare. Practic, există și alte abordări ale ideii de monitorizare a evoluției, dar specific pentru stagiile de internship, fiind un lucru nisat, nu există mai deloc abordări.

Cu alte cuvinte, aplicația InternHub pleacă de la ideea platformelor precum E-learning, cu aplicabilitate în cadrul unei companii. Diferența majoră este că în platforma InternHub nu există note, există doar feedback pe care cei aflați la început de drum îl pot folosi pentru a se dezvolta și pentru a evolua în cariera pe care și-au ales-o.

Un concept foarte în vogă este acela de "gamification", care este procesul de utilizare a elementelor și tehnicilor de design de jocuri în contexte care nu sunt legate de jocuri, pentru a îmbunătăți implicarea utilizatorului sau învățarea. Aceasta este o tehnică populară în domenii precum educația unde poate ajuta la creșterea motivației, prin stârnirea spiritului competitiv. Astfel, fiecare task făcut are un anumit punctaj, iar la final se creează un clasament general prin care oamenii sunt premiați pentru progresul lor.

Acest concept funcționează deoarece se bazează pe competiție, iar oamenii sunt ființe competitive, ceea ce îi face să își dorească să termine ceea ce au de făcut cât mai repede și cât mai eficient, știind că la final există o recompensă. Și simplul clasament are același efect de stârnire al oamenilor, care ajung în timp să fie din ce în ce mai eficienți.

Astfel, aplicația InternHub este o aplicație originală ce are ca puncte de plecare principiile pe care se bazează o aplicație de monitorizare a evoluției, la care se adaugă principii ce se regăsesc într-o platformă precum E-learning.

3. Managementul entităților

Orice sistem orientat pe obiecte are în componenta sa mecanisme de gestionare a stării obiectelor. Folosind mecanisme precum moștenirea, polimorfismul, abstractizarea și încapsularea, limbajul C# ne permite să scriem un cod orientat pe obiecte eficient și ușor de înțeles.

3.1. Modelarea entităților

Modelarea entităților este un aspect important în cadrul sistemelor orientate pe obiecte. Este un pas important să ne gândim ce tip de obiecte vom avea în aplicație și ce relație există între ele. Un prim pas considerat semnificativ de mulți programatori și omis de alții programatori este crearea unor modele schițate înainte de a scrie cod. Acest lucru este important deoarece ne ajută să avem un model solid, care nu suferă modificări inutile, cauzate de un model incomplet. Un model bine gândit de la început ajută la modelarea codului, iar unul care necesită modificări repetate va duce la un cod greu de înțeles, de o complexitate inutilă. Astfel, sunt afectate procesul de mentenanță și actualizare a codului.

Teoria importantă dezvoltată pentru modelul relațional al entităților (ER) implică conceptul de dependență funcțională (FD). Scopul acestui studiu este de a îmbunătăți înțelegerea relațiilor dintre date și de a oferi un cadru formal pentru proiectarea practică a bazelor de date.

Dependențele funcționale (FD) sunt extrase din semantica domeniului aplicației, la fel ca și constrângerile. Ele descriu relațiile și interdependențele dintre atributele individuale. FD-urile reprezintă constrângeri esențiale pentru proiectarea corectă a structurii relaționale a bazelor de date. Prin urmare, este necesar să le analizăm în detaliu pentru a obține o proiectare corespunzătoare și eficientă a bazelor de date. Faza de implementare a bazei de date. [17]

Redundanța este un concept nedorit când vorbim despre proiectarea unei baze de date, deoarece sunt îngreunate procese precum menținerea coerenței și a actualizării. Cu toate că redundanța este un concept de care multă lume fuge, acesta are câteva cazuri în care mărește performanța. Acest lucru se realizează în momentul în care folosim redundanța în locul unei combinații pentru a conecta date. Când dorim să obținem informații pe baza a două tabele conexe folosim o combinație.

Astfel, există cazuri în care redundanța are beneficii, făcând mai ușoară modificarea datelor. Spre exemplu, să presupunem că avem o listă de conturi bancare asemenea figurii 3.1.1.

accountNo	balance	customer	branch	address	assets
A-101	500	1313131	Downtown	Brooklyn	9000000
A-102	400	1313131	Perryridge	Horseneck	1700000
A-113	600	9876543	Round Hill	Horseneck	8000000
A-201	900	9876543	Brighton	Brooklyn	7100000
A-215	700	1111111	Mianus	Horseneck	400000
A-222	700	1111111	Redwood	Palo Alto	2100000
A-305	350	1234567	Round Hill	Horseneck	8000000

Figura 3.1.1 – Beneficiile redundanței

În cazul de față, se poate observa apariția unor date duplicate, care nu pot fi catalogate ca fiind date redundante, cu toate că există anomalii de ștergere cu tabelul. O soluție pentru

această problemă ar fi crearea unei tabele separate pentru clienți, dar acest lucru duce la alte probleme. Una dintre probleme ar fi declanșată de schimbarea adresei unei sucursale, fapt ce necesită schimbarea acestei adrese în mai multe locuri. Dacă tabela rămâne așa cum este, atunci se elimină necesitatea unui tabel de sucursale, fapt ce nu necesită nicio asociere, iar astfel performanța este îmbunătățită, fiind necesare modificări doar într-un singur loc. [17] Acest lucru este realizabil doar în cazul în care nu folosim aceste date în mai mult de o tabelă. În acest caz, abordarea corectă ar fi să avem o tabelă cu sucursale, iar schimbările efectuate în tabela nou adăugată vor avea efect asupra tuturor tabelor care au relații cu aceasta.

O proiectare precară a bazei de date poate duce la următoarele anomalii:

- **Anomalia de inserție.** Inserarea unor date inconsistente reprezintă o anomalie de inserție.
- **Anomalia de actualizare.** Dacă un câmp își schimbă adresa, atunci ar trebui actualizate toate rândurile care conțin această informație.
- **Anomalia de ștergere.** Ștergerea unor informații care nu ar trebui șterse reprezintă o anomalie de ștergere.

Având în vedere cele prezentate mai sus, se poate vedea importanța creării unui model al entităților solid, care să aibă o eficiență maximă și să fie ușor de menținut, actualizat și chiar îmbunătățit.

3.2. Faza de implementare a bazei de date

Așa cum am specificat în subcapitolul anterior, proiectarea bazei de date este unul dintre cele mai importante procese în realizarea unei aplicații. Motivele pentru acest aspect au fost prezentate puțin mai sus. Scopul acestui subcapitol este să vedem fazele de implementare ale unei baze de date și să le arătăm în procesul de creare al bazei de date din cadrul aplicației prezentate.

Fazele de implementare ale unei baze de date se realizează în mai multe etape. Numărul de etape variază în funcție de complexitatea sistemului din care face parte și de nevoile acestuia. Pentru aplicația InternHub, s-au urmat pașii după cum urmează:

1. **Analiza cerințelor:** Acest pas a fost necesar pentru identificarea cerințelor și nevoilor sistemului. Tot în acest pas s-a stabilit structura datelor, operațiile necesare și interacțiunea cu alte sisteme. După finalizarea acestui pas, avem o imagine a bazei de date, a relațiilor între tabele și a acțiunilor necesare îndeplinirii scopului final. După finalizarea pasului 1, putem trece la pasul al doilea.
2. **Proiectarea conceptuală:** La acest pas este necesară crearea unui model conceptual al bazei de date, care definește entitățile și relațiile între ele. Pentru acest pas, se utilizează diagramele entitate-relație. După finalizarea acestui pas, este definit un model conceptual al bazei de date care respectă cerințele de sistem și care este în conformitate cu cerințele și nevoile aplicației. Astfel, avem modelul conceptual necesar la pasul următor.

3. **Proiectarea logică:** Acest pas reprezintă trecerea modelului conceptual într-un model logic compatibil cu sistemul de gestionare al bazei de date (SGBD). La acest pas se definesc următoarele aspecte: schema bazei de date, tabelele, coloanele, cheile primare și secundare, alături de relațiile între tabele. Relațiile între tabele pot fi de mai multe tipuri, dar cele mai des întâlnite sunt următoarele:
- One-to-one: o înregistrare dintr-o tabelă este asociată cu exact o înregistrare dintr-o altă tabelă și reciproc. Această relație este reprezentată prin intermediul cheilor primare și cheilor străine. Un exemplu din aplicația InternHub este relația între tabelele "Person" și "University", unde un profesor poate fi de la o singură universitate, iar o universitate poate avea un singur profesor reprezentant.
 - One-to-many: o înregistrare dintr-o tabelă este asociată cu una sau mai multe înregistrări dintr-o altă tabelă. Înregistrările din tabela sursă au referințe către înregistrările din tabela destinație prin intermediul cheilor străine. Un exemplu din aplicația InternHub este forum-ul, care este format din postări. Astfel, o postare poate avea mai multe comentarii, dar un comentariu poate fi asociat doar unei singure postări.
 - Many-to-many: în această relație, o înregistrare dintr-o tabelă poate fi asociată cu una sau mai multe înregistrări dintr-o altă tabelă, și reciproc. Această relație necesită o a treia tabelă intermediară, care conține două chei străine pentru fiecare relație cu alte tabele existente. În cadrul aplicației InternHub, aceste tabele sunt denumite sub forma "Nume_tabel1_Nume_tabel2". Un exemplu din aplicația prezentată este sistemul de teste. Un test este format din mai multe întrebări, iar întrebările pot fi asociate cu mai multe teste. Astfel, având tabela "Test" și tabela "Question", tabela intermediară este numită "TestQuestion".

De asemenea, normalizarea este procesul care presupune descompunerea unei tabele relationale alcătuite din mai multe atribute în mai multe tabele, care vor forma baza de date, cu scopul eliminării redundanței și anomaliilor care pot apărea. Acest subiect a fost abordat în subcapitolul anterior, iar conceptele pe care se fundamentează acest proces sunt:

- **Dependenta funcțională:**
 - **Dependenta funcțională parțială**
 - **Dependenta funcțională totală**
- **Dependenta multivaloare**
- **Forme normale**

Formele normale sunt principii de proiectare a bazelor de date care asigură o structură bine definită, fără anomalii și redundanțe. Există mai multe forme normale, denumite în mod obișnuit Prima Formă Normală (1NF), a Doua Formă Normală (2NF), a Treia Formă Normală (3NF), Forma Normală Boyce-Codd (BCNF) și a Patra Formă Normală (4NF). În continuare, voi rezuma fiecare formă normală:

1. **Prima formă normală (1NF):** O bază de date este în 1NF atunci când fiecare coloană din fiecare tabel conține doar valori atomice și nu există duplicări de rânduri.
 2. **A doua formă normală (2NF):** O bază de date este în 2NF atunci când îndeplinește cerințele 1NF și fiecare atribut, cu excepția cheilor primare și cheilor străine, este complet dependent de cheia primară, adică nu există dependențe funcționale parțiale.
 3. **A treia formă normală (3NF):** O bază de date este în 3NF atunci când îndeplinește cerințele 2NF și nu există dependențe funcționale între atribuți.
 4. **Forma normală Boyce-Codd (BCNF):** O bază de date este în BCNF atunci când îndeplinește cerințele 3NF și nu există dependențe funcționale nerealizate. În această formă normală, fiecare dependență funcțională trebuie să implice o cheie candidată.
 5. **A patra formă normală (4NF):** O bază de date este în 4NF dacă îndeplinește toate cerințele BCNF și, în plus, nu există dependențe funcționale multidirecționale între atribute.
-
4. **Proiectarea fizică și implementarea:** În această etapă, se traduce modelul logic în structuri specifice SGBD-ului. Se creează tabelele și se definesc tipurile de date și relațiile dintre ele. Pentru aplicația prezentată, implementarea bazei de date s-a realizat folosind migrări pentru fiecare tabel. În cadrul migrărilor se regăsesc și relațiile între cheia primară și cheia străină din alte tabele. Pentru realizarea acestui pas, s-a utilizat pachetul FluentMigrator disponibil în .NET. Aceste migrări sunt apelate și rulate la fiecare pornire a serverului. Un aspect important al lui FluentMigrator este acela că își generează și o tabelă de migrări, astfel rulând doar migrările care nu au fost încă rulate.
 5. **Popularea datelor:** După implementarea bazei de date, se adaugă date reale sau de test în tabele. Acest lucru poate implica inserția manuală a datelor sau importul lor din alte surse. Pentru a putea accesa aplicația, este necesar cel puțin un cont de administrator pentru a putea avea acces la toate celelalte funcționalități, deoarece doar el poate crea conturi în cadrul aplicației.
 6. **Mentenanță și administrare:** După implementare, baza de date necesită mentenanță și administrare continuă. Aceasta include monitorizarea performanței, backup-urile regulate, optimizarea interogărilor și actualizarea schemei în funcție de evoluția cerințelor și nevoilor aplicației.

Aceștia au fost pașii luați în considerare când a fost proiectată, creată și implementată baza de date. Astfel, baza de date respectă regulile de creare ale unei baze de date, fiind ușor de realizat mentenanța și de adus eventuale îmbunătățiri în funcție de cerințele noi ale aplicației.

3.3. Managementul identităților și al utilizatorilor

Când ne referim la managementul identităților și al utilizatorilor (Identity and Access Management - IAM), ne referim practic la metodele prin care doar utilizatorii autorizați au

acces la anumite resurse. Scopul acestui IAM este de a ne asigura că doar utilizatorii autorizați au acces la resursele disponibile în aplicație, menținând în același timp confidențialitatea datelor.

Un IAM este extrem de important, deoarece asigură stabilitatea aplicației prin accesul autorizat la resursele disponibile. Toate request-urile, cu excepția a două dintre ele, necesită autorizare prin intermediul unui JSON Web Token. Acest lucru va fi abordat și explicat în capitolul 4, subcapitolul 4.6 Securitate.

Componentele cheie ale managementului identităților și al utilizatorilor includ:

- **Autentificare:** Verificarea identității utilizatorului prin furnizarea de credențiale valide. Pentru aplicația prezentată, este necesar email-ul, care este unic, și parola, care trebuie să îndeplinească mai multe criterii, despre care vom discuta la prezentarea aplicației. Pentru o securitate sporită, putem apela la metode precum autentificarea în două pași, care implică trimiterea unui cod pe numărul de telefon sau pe adresa de email pentru a verifica identitatea utilizatorului.
- **Autorizare:** Acordarea de privilegii și drepturi de acces bazate pe roluri sau permisiuni specifice. În cadrul aplicației InternHub, există 4 tipuri de utilizatori: Administrator, Trainer, Intern și Profesor Reprezentant. Toți acești utilizatori au drepturi de scriere și citire diferite. Astfel, fiecare utilizator are un acces limitat, care este gestionat atât în partea de front-end, cât și în partea de back-end. Prin autorizare se stabilește rangul fiecărui utilizator, împreună cu drepturile sale de acces în întreaga aplicație.
- **Administrarea utilizatorilor:** Administrarea conturilor de utilizator, inclusiv crearea, actualizarea și ștergerea acestora. Acest lucru este realizabil doar de către administrator, care poate crea toate tipurile de utilizatori, le poate modifica sau le poate șterge la cerere. Pentru transparență și securitate sporită, administratorul, chiar dacă are acces la crearea, editarea și ștergerea utilizatorilor, nu are acces la parolele acestora. La crearea contului, se generează o parolă automată și se trimite un email prin care utilizatorul este rugat să își seteze o parolă pentru a putea accesa aplicația.
- **Managementul accesului:** Controlul accesului utilizatorilor la resursele și serviciile sistemului prin intermediul politicilor de acces, gestionarea sesiunilor și a cererilor de acces. Un exemplu concret din aplicația prezentată ar fi răspunsul la provocări (challenges), care poate fi realizat doar de către traineri. Astfel, o metodă ar fi de a nu permite internilor să facă asta, dar aceasta nu rezolvă complet problema. Acest lucru nu este total sigur, deoarece am blocat accesul doar din aplicație, dar orice persoană poate accesa resursele prin intermediul request-urilor. Astfel, trebuie verificat în funcțiile care au acces limitat dacă persoana în cauză este sau nu autorizată să aibă acces la acea resursă, iar în cazul în care nu este, să se returneze codul 401 Unauthorized.

- **Monitorizare:** Monitorizarea activității utilizatorilor în scopul detectării și prevenirii accesului neautorizat sau abuziv. Cea mai comună metodă pentru realizarea acestei activități este implementarea unui sistem de logging, în care se poate vedea activitatea tuturor utilizatorilor. Evident, un mic dezavantaj al acestei abordări este faptul că pentru fiecare acțiune ar fi necesară încă o operație pe baza de date, ceea ce teoretic ar putea îngreuna aplicația, însă nu este sesizabil. Până la urmă, securitatea este un aspect important pentru care se pot lua în considerare anumite dezavantaje.

Implementarea unui sistem IAM eficient ajută la protejarea datelor și a resurselor din cadrul aplicației, asigurând un control eficient și corect al accesului și reducând riscul de acces neautorizat sau abuziv.

4. Arhitectura aplicației

4.1. Separarea preocupărilor

Separarea preocupărilor în cadrul unei aplicații este un element fundamental și reprezintă, de asemenea, un principiu important în dezvoltarea software. Acesta se referă la împărțirea codului în module astfel încât fiecare componentă să aibă o singură responsabilitate. Avantajele care vin odată cu această separare sunt testarea mai ușoară și ușurarea întreținerii. În plus, acest principiu facilitează extinderea aplicației, păstrând modularitatea codului. Există mai multe modalități de realizare a acestui principiu. Iată câteva concepte:

1. **Arhitectura Model-View-Controller (MVC).**
2. **Principiul responsabilității unice (Single Responsibility Principle - SRP).**
3. **Injectarea dependențelor (Dependency Inversion Principle - DIP).**
4. **Utilizarea design pattern-urilor.**
5. **Utilizarea modularizării.**
6. **URI-uri semantice.**
7. **Formatul de reprezentare.**

Acum că am prezentat conceptele, putem discuta puțin despre fiecare și să le prezentăm apartenența la API-ul aplicației prezentate.

Legat de arhitectura MVC, aplicația este împărțită în trei module principale, după cum îi spune și numele, și anume:

- **Model:** responsabil pentru gestionarea datelor.
- **View:** responsabil pentru afișarea datelor primite prin intermediul controller-ului. De asemenea, se ocupă de interacțiunea cu utilizatorul final.
- **Controller:** responsabil pentru comunicarea între model și view, precum și gestionarea evenimentelor utilizatorului.

Fiind o aplicație de tip full-stack, doar două dintre cele trei elemente se regăsesc în cadrul server-ului, respectiv Model-ul și Controller-ele. În cazul aplicației InternHub, View-ul este reprezentat de partea de front-end, care preia datele de la Controller și le expune

utilizatorului într-un mod plăcut. În plus, toate acțiunile utilizatorului sunt monitorizate și gestionate de către client și trimise către controller pentru manipularea acestor acțiuni.

Acest principiu este legat de principiul 5, care se referă la împărțirea aplicației în module logice și funcționale care conțin elemente relevante. Aceasta poate fi realizată prin intermediul modulelor în Angular, pachetelor sau bibliotecilor separate în .NET sau prin organizarea fișierelor și directoarelor într-un mod coerent și modular. Acest lucru este vizibil în figurile 4.1.1 pentru Angular și 4.1.2 pentru .NET.

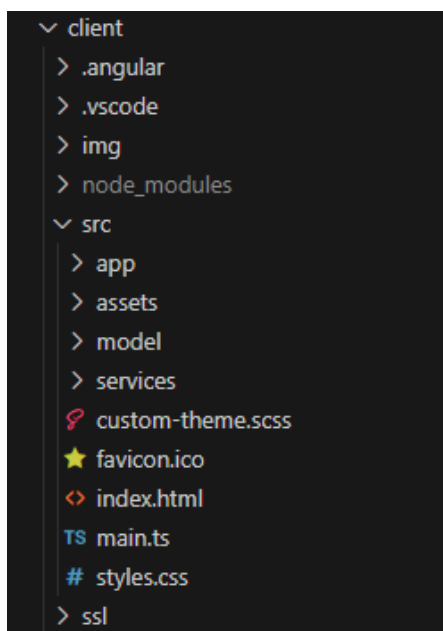


Figura 4.1.1 – Impartirea client-ului pe module logice si funcitonale

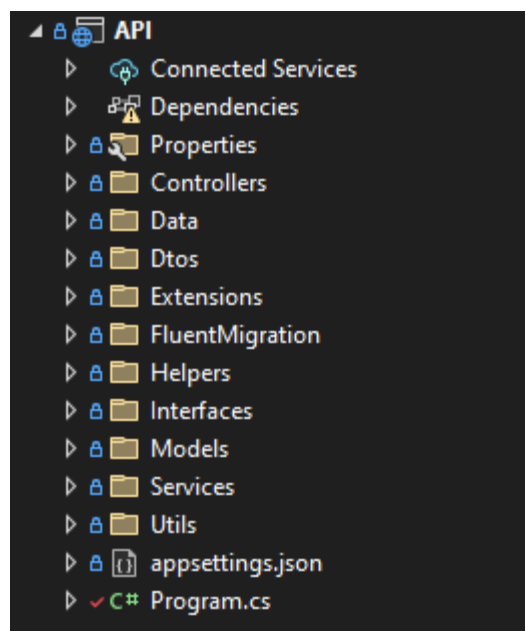


Figura 4.1.2 – Impartirea server-ului pe module logice si funcionale

Astfel, împărțirea pentru partea de server se face prin intermediul organizării fișierelor și directoriilor în conformitate cu principiul 2, iar pentru partea de client acest lucru este posibil datorită pachetelor care facilitează această separare. Am specificat mai sus că sunt realizate în conformitate cu principiul 2, respectiv principiul responsabilității unice.

Acest principiu se referă la faptul că fiecare componentă trebuie să aibă o singură responsabilitate, ceea ce are ca avantaj menținerea unui cod mai ușor și ajută la îmbunătățirea și testarea acestuia. Pentru partea de server, acest principiu este respectat atât la nivelul directoriilor, cât și la nivelul claselor. Totodată, se poate observa împărțirea serverului în conformitate cu arhitectura MVC, având un director atât pentru Model, cât și pentru Controller. Datorită acestei împărțiri modulare a soluției back-end-ului, este mult mai ușor să utilizăm diverse design pattern-uri, conform principiului 4.

Design pattern-ul utilizat, care are un impact major asupra modularității aplicației, este Repository. Acest design pattern acționează ca un intermediar între restul aplicației și logica de acces la date. Practic, acesta izolează toate codurile de acces la date de restul aplicației, având ca avantaj principal ușurința modificării codului când este necesar. Acest lucru este posibil datorită faptului că tot codul este găsit într-un singur loc. De asemenea, testarea controllerelor

devine mai ușoară. Spre exemplu, avem următoarea diagramă:

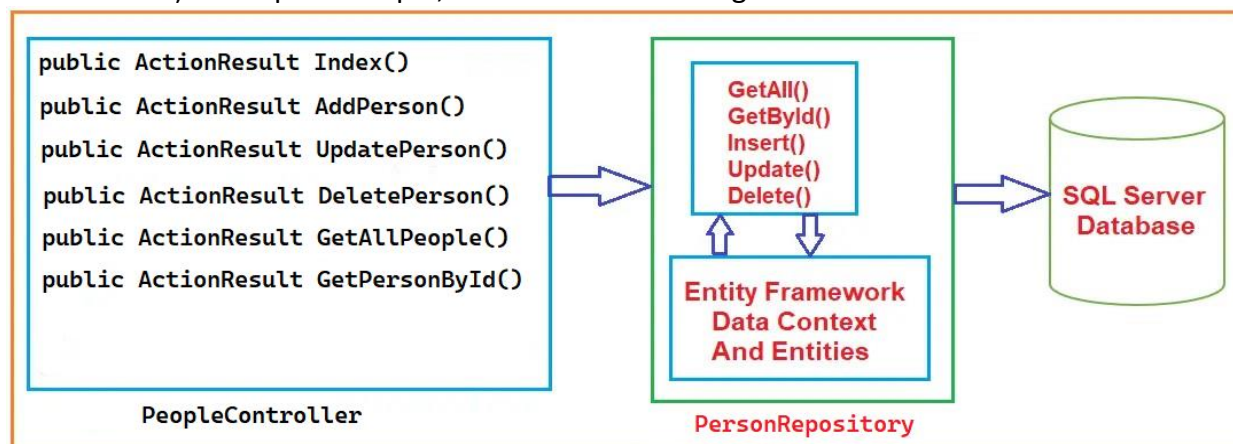


Figura 4.1.3 – Diagrama pentru controller si repository

După cum puteți observa în diagrama de mai sus, acum controllerul People nu va comunica direct cu clasa contextului de date a Entity Framework. De asemenea, acum nu există interogări sau alte coduri de acces la date scrise în metodele de acțiune ale controllerului People, acestea fiind realizate în interiorul repository-ului Person. Avem o diferență de nume între repository și controller, care este cauzată de unul dintre principiile arhitecturii REST, despre care vom discuta mai mult în subcapitolul următor. Repository-ul Person utilizează clasa contextului de date a Entity Framework pentru a efectua operațiile CRUD. După cum puteți vedea, acum repository-ul Person are metode precum `GetAll()`, `GetById()`, `Insert()`, `Update()` și `Delete()`. Aceste metode vor efectua operațiile tipice CRUD asupra bazei de date subiacente. Controllerul Employee utilizează aceste metode pentru a efectua operațiile necesare în baza de date.

Un repository reprezintă pur și simplu o clasă definită pentru o entitate, care conține toate operațiile posibile în baza de date. În plus, în aplicația InternHub, pe partea de server, există o interfață pentru fiecare repository. Toate interfețele sunt numite de tipul `I + nume entitate + Repository`, iar repository-urile sunt de forma `+ nume entitate + Repository`. Acest lucru ajută la o viziune de ansamblu mai bună pentru metodele existente în fiecare repository. De exemplu, un repository pentru o entitate Person va conține operațiile CRUD de bază și orice alte operații posibile legate de entitatea Person.

Un alt motiv pentru care am utilizat această modalitate, și anume implementarea unei interfețe, ține de principiul 3, și anume Injectarea dependențelor (Dependency Inversion Principle – DIP). Astfel, fiecare repository are injectat atât contextul, cât și IMapper-ul, care se ocupă de maparea obiectelor, iar unele mai au injectate și alte repository-uri, acolo unde este cazul. Atât contextul, cât și interfețele sunt de tip private readonly. Pentru partea de controllere, avem injectate toate repository-urile de care avem nevoie. Numărul lor poate să difere în funcție de controller, unele având nevoie de mai multe decât o singură dependență. Acest lucru se poate vedea în figurile 4.1.4 pentru repository doar cu contextul și mapper-ul injectate, 4.1.5 pentru repository-uri care au și alte repository-uri pe lângă context și 4.1.6 care reprezintă controller-ul și un număr variabil de repository-uri injectate.

```
private readonly InternShipAppSystemContext _context;
private readonly IMapper _mapper;

0 references | Bolba-Mateescu Andrei-Ioan, 3 days ago | 1 author, 1 change
public ChallengeRepository(InternShipAppSystemContext context,
    IMapper mapper)
{
    _context = context;
    _mapper = mapper;
}
```

Figura 4.1.4 – Repository cu conext si Mapper injectate

```
private readonly InternShipAppSystemContext _context;
private readonly IPersonRepository _person;
private readonly IMapper _mapper;

0 references | Bolba-Mateescu Andrei-Ioan, 9 days ago | 1 author, 1 change
public GroupChatRepository(InternShipAppSystemContext context,
    IPersonRepository person,
    IMapper mapper)
{
    _context = context;
    _person = person;
    _mapper = mapper;
}
```

Figura 4.1.5 – Repository cu conext, Mapper si alte repository-uri injectate

```
private readonly IChallengeRepository _challengeRepository;
private readonly IChallengeSolutionRepository _challengeSolutionRepository;
private readonly IPersonRepository _personRepository;

0 references | Bolba-Mateescu Andrei-Ioan, 1 hour ago | 1 author, 1 change
public ChallengeController(IChallengeRepository challengeRepository,
    IChallengeSolutionRepository challengeSolutionRepository,
    IPersonRepository personRepository)
{
    _challengeRepository = challengeRepository;
    _challengeSolutionRepository = challengeSolutionRepository;
    _personRepository = personRepository;
}
```

Figura 4.1.6 – Controller cu repository-uri injectate

Un alt principiu prezent în aplicația InternHub este URI semantice, care se referă la faptul că URI-urile ar trebui să fie semantice și să reflecte acțiunile și entitățile pe care le reprezintă. În plus, ele ar trebui să fie descriptive și ușor de înțeles. Avem un exemplu de astfel de URL în figura 4.1.7.

Un ultim principiu al separării preocupărilor este formatul de reprezentare, care se referă la faptul că datele returnate ar trebui să fie într-un format standard, precum JSON sau XML. Pentru aplicația prezentată, formatul ales este JSON și este prezent atât pentru comunicarea de la server la client, cât și pentru comunicarea clientului cu serverul prin intermediul body-ului. Pentru ambele tipuri de comunicare avem reprezentarea în figura 4.1.7.

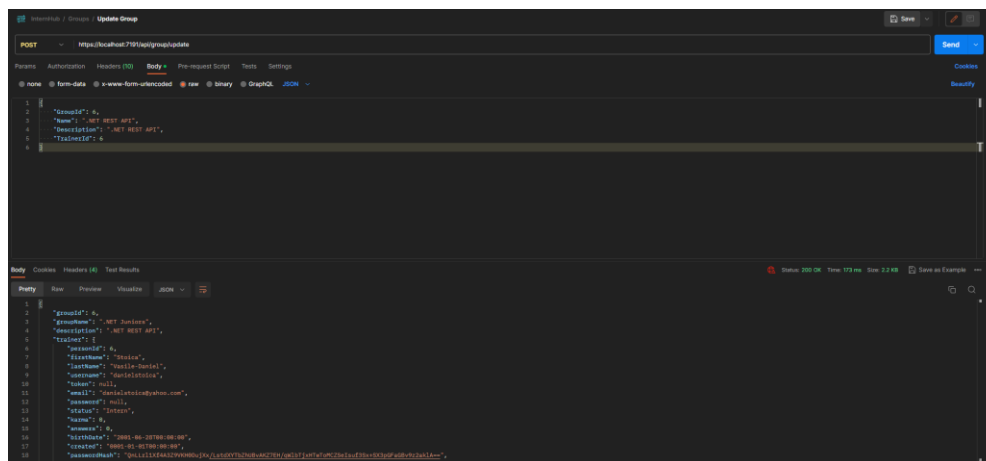


Figura 4.1.7 – Formatul datelor in comunicarea client-server

Conform figurii 4.1.7, URL-ul este: <https://localhost:7191/api/group/update>. Este un URL semantic, ușor de înțeles. Doar uitându-ne la el ne dăm seama că se face update la un grup,

care primește datele ce trebuie actualizate prin intermediul unui body. Este un URL ușor de înțeles atât pentru dezvoltatori, cât și pentru utilizatori.

Legat de formatul de reprezentare, acest lucru se poate observa ușor din figura 4.1.7. Body-ul este de tip JSON, având ca scop trimiterea de informații către server, iar răspunsul este, de asemenea, de tip JSON.

Prin urmare, putem afirma că API-ul aplicației prezentate ține cont de principiile prezentate mai sus și urmează cele mai bune practici pentru a asigura un API eficient, ușor de utilizat și scalabil.

4.2. Arhitectura REST pentru designul unui API

Mijloacele API (Application Programming Interface) și, la fel ca orice altă interfață, permit interacțiuni. În cazul unui API, acesta permite interacțiuni între sisteme, urmând un set de standarde și protocoale pentru a partaja caracteristici, informații și date. Cu alte cuvinte, oferă dezvoltatorilor posibilitatea de a construi și proiecta produse și servicii care vor comunica cu alte produse și servicii.

În programare, termenul de arhitectură software se referă la un set de reguli de proiectare care identifică tipurile de componente și conectori ce pot fi utilizați pentru a compune un sistem sau un subsistem. Pipe and Filter, Layered, Push-Based sunt câteva exemple comune de astfel de arhitecturi software. Unul dintre cele mai întâlnite arhitecturi este arhitectura REST, care are o abordare client-server, fără stare, în care resursele sunt văzute ca fiind servicii web ce pot fi identificate prin intermediul URL-ului. Modul prin care clienții pot utiliza aceste resurse este prin intermediul metodelor HTTP, dar despre asta vom vorbi în capitolul 4, la subcapitolul 4.3 CRUD folosind verbe HTTP. În plus, metodele HTTP pot realiza atât transferul conținutului aplicației, cât și acțiuni asupra resurselor. Un punct forte al arhitecturii REST este acela că interacțiunea între el și protocolul HTTP se realizează fără întreruperi.

REST este un acronim de la REpresentational State Transfer (Transferul Stării Reprezentaționale). Acesta nu este un termen nou. Acest stil arhitectural are ca scop principal centralizarea datelor ce urmează să fie utilizate de către diverse aplicații client. REST este bazat pe un set de principii care descriu modul de definire și adresare al resurselor de rețea. Roy Fielding este cel care a descris inițial aceste principii în anul 2000.

Ca orice altă arhitectură, REST are, de asemenea, reguli și principii de programare care trebuie urmate pentru a obține rezultatele dorite de la aplicație. Printre aceste reguli se enumără:

1. REST se bazează pe substantivul resursei, de aceea este indicat ca URL-ul să se termine cu un substantiv.
2. GET, POST, DELETE și UPDATE sunt verbele folosite pentru a identifica o acțiune. Pe lângă acestea, mai avem: TRACE, OPTIONS, CONNECT și PATCH.
3. Aplicația web trebuie organizată în resurse și verbe.

4. Pentru consecvență, trebuie folosit pluralul în URL-uri.
5. Este necesară trimiterea unui cod corect pentru identificarea succesului sau a erorii.

Mai jos sunt câteva exemple de URL-uri ce respectă regulile expuse mai sus:

- `api/people GET` - Se obțin toate persoanele.
- `api/people POST` - Se adaugă o persoană nouă.
- `api/people/1 PUT` - Se actualizează o persoană cu id = 1. Informațiile referitoare la persoană se transmit prin intermediul body-ului.
- `api/people/1 DELETE` - Se șterge o persoană cu id = 1.
- `api/people/popescumihai GET` - Se obține o persoană cu username = popescumihai.

Acum că am discutat regulile pe care se bazează REST API, trebuie luate în considerare și cele 6 constrângeri arhitecturale tipice REST API:

1. **Interfața uniformă:** Această constrângere se referă la faptul că orice resursă trebuie să fie identificată în mod unic prin intermediul unui URL, folosind doar metode precum GET, POST, PUT sau DELETE. În plus, este necesară prezența unei metode unice de comunicare între client și server, indiferent de tipul de dispozitiv pe care rulează aplicația.
2. **Stateless:** Această constrângere se referă la modul de gestionare a stării, care trebuie făcută întotdeauna pe dispozitivul clientului, ci nu pe server.
3. **Cacheable:** Această constrângere se referă la proiectarea resurselor astfel încât să permită stocarea în cache.
4. **Client-server:** Această constrângere se referă la faptul că trebuie să existe o delimitare clară între client și server. Astfel, fiecare are responsabilitățile sale. Spre exemplu, accesul la date și modificarea acestora este responsabilitatea serverului, iar elementele de UI/UX sunt responsabilitatea clientului.
5. **Sistem stratificat:** Această constrângere se referă la faptul că arhitectura REST permite utilizarea unei structuri stratificate compusă din mai multe niveluri de servere.
6. **Cod la cerere:** Această constrângere se referă la faptul că, de obicei, forma sub care sunt trimise resursele la client este de tip JSON, dar serverul poate trimite, când este necesar, un cod executabil la cerere.

4.3. CRUD folosind verbe HTTP

Modalitatea favorită de comunicare între părțile componente ale unei aplicații full-stack, aici vorbind despre front-end și back-end, este indiscutabil HTTP. Comunicarea client-server se folosește de acest protocol prin trimiterea de cereri și primirea unui răspuns specific HTTP, despre care vom vorbi mai în detaliu în subcapitolul următor. Astfel, există două tipuri de mesaje:

- **Cererea (Request)** - este cea trimisă de la client către server, având ca scop principal declanșarea unei acțiuni.
- **Răspunsul (Response)** - este cel trimis de la server către client, având ca scop trimiterea răspunsului după finalizarea acțiunii declanșate de cerere.

Aceste mesaje sunt sub forma unui text ASCII dispus, de cele mai multe ori, pe mai multe linii. Prima linie conține verbul HTTP, care este urmat de calea către resursa la care dorim să ajungem, în cazul cererii. Pentru răspuns, pe prima linie găsim codul de răspuns și mesajul în sine.

Pentru descrierea acțiunilor necesare efectuării operațiilor de tip CRUD se folosesc metodele HTTP echivalente. Singura excepție este cazul în care dorim o cerere care să aibă în componentă și un body. În cazul acesta, nu vom putea folosi verbul GET deoarece acesta nu suportă un body, cel puțin în cazul unor framework-uri de tip front-end, ci vom fi nevoiți să apelăm la verbul POST pentru a putea face acest lucru. Pentru cazul în care body-ul pe care vrem să-l trimitem are puține câmpuri, este de preferat să folosim GET, dar în cazul în care avem mulți astfel de parametri, însiruirea lui în cadrul URL-ului ar îngreuna atât citirea acestuia în scopul găsirii și rezolvării unei posibile probleme, dar și îngreunarea citirii și urmăririi funcției echivalente.

Spre exemplu, dacă dorim să primim ca răspuns de la API toate persoanele din baza de date care au vârsta minimă 18 ani, vârsta maximă 50, înălțimea minimă 150 cm, maximă 180 cm, din Brașov, să fie bărbați, să fie căsătoriți, să aibă facultatea terminată, să aibă un job, să aibă o mașină și să aibă copii, avem două variante. Figura 4.4.1 reprezintă funcția în cazul în care utilizăm verbul GET, iar figura 4.4.2 reprezintă același request dar realizat cu verbul POST. Se poate observa o diferență notabilă între cele două. În cazul acesta, chiar dacă nu este cea mai corectă alegere, verbul POST este o alegere mai bună în cazul cererilor de acest fel. Este evident că, dacă avem puțini parametri, nu vom opta pentru această abordare, deoarece nu are niciun avantaj real pentru un număr mic de parametri.

```
[HttpGet("people/{minAge:int}/{maxAge:int}/{minHeight:double}/{maxHeight:double}/" +
    "{city}/{gender}/{married:bool}/{hasFinishedUniversity:bool}/{hasAJob:bool}/" +
    "{hasACar:bool}/{hasKids:bool}")]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult GetPeople(int minAge, int maxAge, double minHeight, double maxHeight,
    string city, string gender, bool married, bool hasFinishedUniversity,
    bool hasAJob, bool hasACar, bool hasKids)
```

Figura 4.3.1 – Request realizat cu verbul GET

```
[HttpPost("people")]
0 references | 0 changes | 0 authors, 0 changes
public ActionResult GetPeople([FromBody] Person person)
```

Figura 4.3.2 – Request realizat cu verbul POST

Pentru prima variantă, un URL arată în felul următor:

- <https://localhost:5421/api/people/18/50/150/180/Brasov/masculin/1/1/1/1/1>

Pentru cea de-a doua variantă, un URL arată în felul următor:

- <https://localhost:5421/api/people>, urmat de body-ul în care vom pune informațiile necesare.

Se poate observa o încărcare inutilă a URL-ului în primul caz, față de al doilea unde avem un URL simplu, ușor de urmărit și, totodată, ușor de schimbat valorile care ne interesează pentru acest request. În plus, pentru genul acesta de probleme cu mulți parametrii, nu este necesară scrierea mai multor funcții, deoarece putem modifica body-ul pentru a ne satisface nevoile.

Metodele HTTP, precum GET și POST, sunt utilizate pentru executarea acțiunilor necesare în vederea creării, citirii, actualizării și ștergerii datelor în baza de date. Aceste operații sunt cunoscute sub numele de operații CRUD (Create, Read, Update și Delete). Fiind adesea folosite, este evidentă existența unei analogii cu operațiile în baza de date relațională SQL, care se bazează pe un set comun de verbe prezentat în tabelul de mai jos.[24]

Actiune	SQL	HTTP
Create	Insert	PUT
Read	Select	GET
Update	Update	POST
Delete	Delete	DELETE

Tabelul 1 – Relatie între REST API si protocolul HTTP

Un Webservice RESTful se bazează pe metodele HTTP și pe conceptele arhitecturii REST. Acesta definește în mod tipic un URI de bază pentru resurse și tipurile MIME suportate (XML, Text, JSON, Protocol Buffers, etc.) și setul de operații HTTP. Aceste metode standard sunt[24]:

- **GET** – Definește accesul pentru citirea datelor, evitând efectele secundare și modificarea acestora în urma executării cererii de tip GET.
- **PUT** – Creează o resursă nouă și are ca principală caracteristică idempotența.
- **DELETE** – Similar cu PUT, DELETE este idempotentă și se ocupă de ștergerea resursei.
- **POST** – Este similar cu PUT, având în plus avantajul că poate modifica o resursă, nu doar crea una nouă.

Un aspect important de menționat în ceea ce privește arhitectura HTTP este acela că este independentă față de protocolul HTTP, ceea ce înseamnă că arhitectura REST nu necesită utilizarea obligatorie a protocolului HTTP.

4.4. Statusuri de raspuns

După cum am menționat, fiecare request primește un răspuns, care este sub formă de coduri numerice ce indică dacă acțiunea dorită a fost realizată cu succes sau nu. Nu întotdeauna acțiunea declanșată de request este executată cu succes, iar aceste coduri de status informează clientul cu privire la acțiunile necesare.

Aceste coduri numerice sunt împărțite în categorii principale, fiecare cu un înțeles specific. Acestea sunt împărțite în 5 categorii, pe care le vom vedea în tabelul 2.

Cod numeric	Tip	Mesaj
1xx	Informational	Cerere in curs de procesare sau informatii incomplete
2xx	Succes	Cerere executata cu succes
3xx	Redirectionare	Sunt necesare actiuni suplimentare in vederea finalizarii cererii
4xx	Eroare a clientului	Cerea contine erori sau nu poate fii procesata
5xx	Eroare a serverului	Serverul intampina probleme in rezolvarea cereri

Tabel 2 – Categorii de staturi de raspuns

Acum că știm care sunt categoriile de status-uri de răspuns, putem trece prin câteva exemple de astfel de status-uri de răspuns.

Cod numeric	Mesaj	Explicatie
200	OK	Cerere procesats cu succes
201	Created	Resursa creata cu succes
400	Bad Request	Cererea contine erori de sintaxa sau nu poate fii inteleasa
401	Unauthorized	Accesul restrctionat la resursa
404	Not Found	Resursa solicitata nu a fost gasita
500	Internal Server Error	Serverul intampina probleme in rezolvarea cereri

Tabel 3 – Exemple de statusuri de raspuns

Evident că există și alte coduri de stare HTTP, iar fiecare cod are semnificația sa specifică în cadrul protocolului HTTP, iar fiecare aplicație are modul ei de a le înțelege și a le procesa. Aceste coduri de stare sunt esențiale pentru a comunica rezultatele cererilor între client și server. În cadrul unei aplicații este necesar ca aceste coduri să fie înțelese și interpretate corect pentru a putea diagnostica și remedia eventualele probleme care pot apărea în cadrul comunicării client-server.

4.5. Modelul “Clean Arhitecture” definit de Uncle Bob

Robert C. Martin, cunoscut sub numele de Uncle Bob, ne prezintă în cartea sa "Clean Architecture: A Craftsman's Guide to Software Structure and Design" [20] un model de arhitectură pragmatic, numit simplu "Clean Architecture", ce a evoluat din arhitectura Onion și din cea hexagonală.

Pe lângă cartea publicată în 2017, acesta are și un blog numit "The Clean Code Blog" [22], unde are publicații încă din 2011 până în prezent. Tot în blogul său găsim o postare despre "Clean Architecture", unde practic expune aceleași idei ca în cartea sa, sensibil modificate, având în vedere distanța de timp.

Astfel, în 2012, în cadrul postării despre "Clean Architecture", acesta observă o gamă întreagă de idei legate de arhitectura sistemelor, printre care: arhitectura hexagonală, Onion, Screaming DCI și BCE. Toate acestea au în comun obiectivul de separare a preocupărilor, subiect ce a fost discutat în subcapitolul precedent. Toate au cel puțin un strat pentru regulile de business și un alt strat pentru interfețe[22]. Fiecare dintre aceste arhitecturi produce sisteme care sunt:

- **Independente de framework-uri.** Arhitectura curată nu se bazează pe existența unor biblioteci software complexe și pline de funcționalități. Acest lucru oferă libertatea de a utiliza framework-uri ca instrumente, fără a fi nevoie să adaptați sistemul la constrângerile impuse de acestea.
- **Testabilitatea.** Testarea regulilor de business se poate face fără intervenția interfeței, bazei de date, serverului web sau altor elemente externe.
- **Independente de interfața utilizator.** Interfața utilizator trebuie să poată suferi modificări fără a afecta întregul sistem. O interfață web poate fi înlocuită de o consolă fără a modifica regulile de business.
- **Independente de baza de date.** Același lucru se aplică și pentru baza de date. Aceasta poate fi modificată de la SQL Server la MongoDB fără a afecta regulile de business.

Principiul de bază al acestei arhitecturi este că dependența între straturile superioare și cele inferioare există fără a fi afectate cele inferioare. În plus, straturile inferioare nu știu despre această dependență. Acest lucru facilitează schimbarea componentelor sistemului în tandem cu extinderea individuală a fiecărui strat. Un alt avantaj este acela că datorită acestei separări ni se permite o scriere mai ușoară a testelor unitare, fapt ce ajută la un nivel de calitate superior. Diagrama de mai jos reprezintă o încercare de integrare a tuturor acestor arhitecturi într-o singură idee practică.

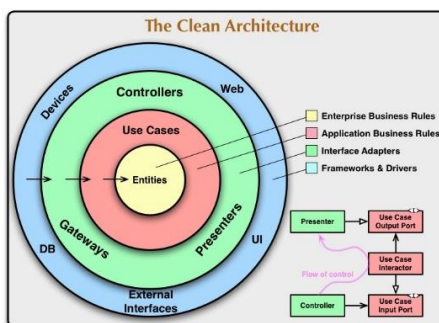


Figura 4.5.1 – The Clean Architecture

Tot Uncle Bob ne prezintă niște principii care s-au dovedit a fi eficiente în crearea unui cod ușor de citit și de menținut. Pe lângă principiile SOLID, există principii legate de denumirea variabilelor, de design și de metode, alături de scrierea de teste automate.

Conform concepției de "Code Smells" a lui Uncle Bob, un cod care "miroase", în sensul rău al cuvântului, este un cod care este greu de modificat, generează erori la cele mai mici schimbări, nu permite refolosirea bucăților de cod și are o complexitate inutilă care face codul greu de înțeles.

Jason Taylor este un programator care a popularizat aplicarea acestui concept de "Clean Code" în cadrul conferinței internaționale NDC. Acesta a creat un template folosind CLI .NET Core folosind următoarea comandă:

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

Apoi pentru a crea o nouă soluție s-a folosit de comanda:

```
dotnet new ca-sln
```

Apoi fiecare nivel este reprezentat de un proiect în Visual Studio.

4.6. Securitate

Știm cu certitudine că securitatea este un aspect important când vine vorba de aplicații. Astfel InternHub are 3 tipuri de utilizatori cu drepturi de acces diferite. Astfel, este necesar să nu lăsăm persoane neautorizate să efectueze modificări nedorite în cadrul aplicației.

Acest subcapitol are ca scop prezentarea securității request-urilor, deoarece securitatea datelor este prezentată în capitolul 5, subcapitolul 5.4 Securitatea datelor.

În cadrul aplicației prezentate există 2 tipuri de request-uri: cele care necesită autorizare și cele care nu necesită autorizare. Cele care nu necesită autorizare sunt acele request-uri care, fie nu au cum să obțină token-ul de autorizare, precum request-ul de resetare a parolei, fie au ca scop generarea unui astfel de token, precum funcția de log-in. În rest, toate celelalte request-uri necesită acest token de autorizare, care este salvat în memoria cache și este eliberat odată cu delogarea utilizatorului sau după expirare.

În cadrul aplicației am ales să folosesc autorizarea cu token JWT (JSON Web Token), care este generat după autentificarea utilizatorului și conține informații precum numele de utilizator.

- **JSON Web Token**

Tokenul JSON Web (JWT) este un standard propus pentru Internet care permite crearea de date cu semnătură opțională și/sau criptare opțională. Conținutul acestui token este reprezentat în format JSON și poate conține diverse afirmații (claims). Tokenurile sunt semnate fie cu o cheie secretă privată, fie cu o pereche de chei publice/privat.[19]

În figura 4.6.1 avem exemplul unui token atât criptat, cât și decriptat.

Astfel, programatorul poate alege ce request-uri necesită autorizare și care nu. Lipsa acelei adnotări reprezintă faptul că request-ul nu necesită autorizare. Un aspect important este că putem avea câteva care necesită autorizare, iar pentru restul să nu fie necesar. Nu suntem obligați să autorizăm tot controller-ul dacă nu este cazul.

5. Noțiuni teoretice și tehnologii folosite

Aplicația InternHub este o aplicație full-stack, din punct de vedere al dezvoltării, formată din două părți: frontend și backend. Persistarea datelor a fost realizată utilizând Microsoft SQL Server.

Pentru partea de backend, a fost folosit framework-ul .NET, cu limbajul de programare C#. Motivația pentru această alegere va fi prezentată în următoarele pagini, începând cu subcapitolul 5.1 pentru partea de backend.

Frontend-ul a fost realizat utilizând Angular, iar ca și limbaj de programare s-a folosit TypeScript. De asemenea, motivarea pentru folosirea acestui framework va fi abordată la subcapitolul 5.2.

Pentru persistența datelor, s-a folosit Microsoft SQL Server împreună cu Entity Framework pentru comunicarea între backend și baza de date. Și pentru aceasta avem motivarea alegerii la subcapitolul 5.3.

5.1. .NET 7

- **.NET Framework**

.NET 7 este succesorul lui .NET 6 și se concentrează pe unificare, modernizare, simplitate și performanță. .NET 7 va beneficia de suport tehnic pentru o perioadă de 18 luni, termen standard în cadrul ecosistemului .NET. A fost lansat pe 8 noiembrie 2022 și va avea suport tehnic până pe 14 mai 2024.

Printre noile caracteristici aduse de la .NET 6 la .NET 7 se numără îmbunătățiri de performanță, serializare JSON, matematică generică, expresii regulate, noi biblioteci .NET, SDK .NET și actualizări pentru rezolvarea unor probleme ale subsistemelor care folosesc ecosistemul .NET.

Pentru a înțelege mai în detaliu semnificația acestor noi caracteristici pentru .NET 7, le vom dezvolta puțin pe cele prezentate mai sus:

- **Performanța:** Performanța este una dintre caracteristicile cheie ale .NET 7, iar toate caracteristicile sale sunt concepute ținând cont de performanță. În plus, .NET 7 include îmbunătățiri destinate exclusiv performanței, precum înlocuirea pe stivă (On-stack replacement), optimizare ghidată pe profil (Profile-guided optimization), îmbunătățirea generării de cod pentru Arm64 și performanță îmbunătățită pentru timpul de rulare Mono.

- **Serializarea JSON:** .NET 7 include îmbunătățiri ale serializării System.Text.Json în domenii precum serializarea polimorfică și suportul pentru membrii necesari.
- **Matematica generică:** .NET 7 și C# 11 includ inovații care permit efectuarea de operații matematice în mod generic. Scrierea unei funcții care are ca scop efectuarea unor operații matematice pentru două numere este simplificată, deoarece nu mai este nevoie de supraîncărcarea metodei pentru fiecare tip dorit. Acum este posibilă scrierea unei metode generice în care parametrul tip este constrâns să fie un tip asemănător unui număr.
- **Expresii regulate:** Biblioteca de expresii regulate a .NET a înregistrat îmbunătățiri semnificative funcționale și de performanță în .NET 7 [1].

Ecosistemul .NET este dezvoltat și menținut de Microsoft și rulează în principal pe Microsoft Windows. Dezvoltarea acestui ecosistem a început la sfârșitul anilor 1990, sub un nume diferit față de cel cunoscut astăzi. Inițial denumit Next Generation Windows Services, a făcut parte din strategia .NET. Începând cu anul 2000, au fost lansate primele versiuni beta ale .NET 1.0, iar în februarie 2002 a fost anunțată și lansată prima versiune, .NET 1.0. În prezent, framework-ul se află la versiunea .NET 8, iar versiunea .NET 8 urmează să fie lansată în noiembrie 2023 [2].

Cel mai mare avantaj al framework-ului .NET este faptul că este compatibil cu platforma Windows. Aproape toată lumea lucrează cu mașini Windows [2].

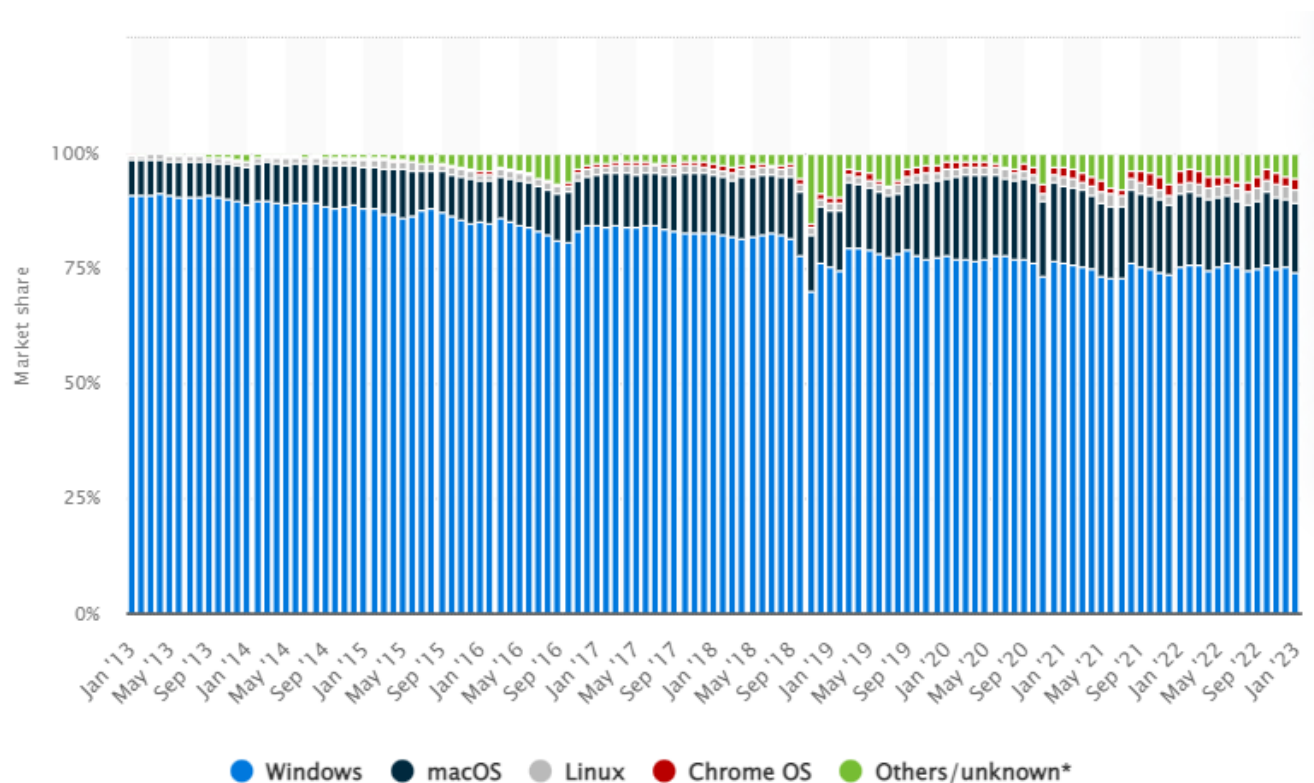


Figura 5.1.1 - Cota de piață globală deținută de sistemele de operare

Conform figurii 5.1.1, se poate observa că sistemul de operare preferat în întreaga lume este Windows, cu un procent constant mai mare de 74% în ultimii 10 ani [3].

Un alt argument în favoarea ecosistemului .NET este lansarea constantă de noi versiuni ale acestui ecosistem și menținerea suportului pentru versiunile anterioare celei curente, pentru a facilita trecerea de la o versiune la alta. În prezent, durata de suport tehnic este de 6 luni după lansarea ultimei versiuni.

Pentru lucrarea în cauză, am ales să lucrez cu .NET 7, deoarece este cea mai nouă versiune și oferă avantaje considerabile față de alte opțiuni. Un alt motiv pentru alegerea .NET 7 este încheierea suportului tehnic pentru .NET 3.1 în decembrie 2022, la scurt timp după lansarea .NET 7, iar tendința generală în companii este de a migra codul de la .NET 3.1 la .NET 6 sau .NET 7. Motivul pentru alegerea .NET 6 este că, la fel ca .NET 3.1, este o versiune LTS (Long-Term Support), care beneficiază de suport până în noiembrie 2024 [4].

Motivul pentru alegerea versiunii curente în detrimentul unei versiuni LTS este, pe lângă motivele prezentate la începutul acestui subcapitol, faptul că am dorit să experimentez ultima versiune.

- **C#**

C# este un limbaj de programare care acceptă mai multe paradigme. Inițial lansat în 2000, acesta a ajuns deja la versiunea 11, lansată odată cu .NET 7 în noiembrie 2022. Asemenea .NET, C# beneficiază constant de actualizări și lansări de noi versiuni, fiind astfel o alegere optimă pentru partea de backend și frontend [5].










Apr 2023	Apr 2022	Change	Programming Language	Ratings	Change
1	1		 Python	14.51%	+0.59%
2	2		 C	14.41%	+1.71%
3	3		 Java	13.23%	+2.41%
4	4		 C++	12.96%	+4.68%
5	5		 C#	8.21%	+1.39%
6	6		 Visual Basic	4.40%	-1.00%
7	7		 JavaScript	2.10%	-0.31%
8	9	▲	 SQL	1.68%	-0.61%
9	10	▲	 PHP	1.36%	-0.28%

Figura 5.1.2 – TIOBE Programming Community Index

Conform figurii 5.1.2, se poate observa că, deși C# nu este cel mai popular limbaj de programare, este în continuă creștere și se situează pe locul al 5-lea ca cel mai utilizat limbaj de programare. În iunie 2001, la scurt timp după lansare, avea o cotă de piață de 0,84%, iar acum se află la 8,21%, înregistrând o creștere de 1,39% în ultimul an [6].

În ceea ce privește API-ul, pentru dezvoltarea aplicației s-a utilizat ASP.NET Core Web API. Potrivit statisticilor realizate de Stack Overflow în 2022, legate de framework-urile de dezvoltare pentru aplicații web, atât ASP.NET Core, cât și ASP.NET sunt destul de populare. Acestea se află imediat după Angular și Vue.js în rândul celor intervieuați, în număr de 58,743, așa cum se poate observa în figura 5.1.3. În rândul programatorilor experimentați, ASP.NET Core ocupă o poziție superioară față de Vue.js, în timp ce ASP.NET rămâne pe aceeași poziție. Numărul programatorilor experimentați intervieuați de către Stack Overflow este puțin mai mic, dar totuși considerabil, fiind de 45,297, conform figurii 5.1.4. În schimb, cei care sunt la început de carieră aleg mai puțin ASP.NET și ASP.NET Core. Numărul celor intervieuați care se află la început de carieră este de 4,932, așa cum se poate observa în figura 5.1.5. Cu toate acestea, primele două opțiuni sunt la fel de populare, indiferent dacă vorbim de utilizatori cu experiență sau de cei la început de carieră [7].



Figura 5.1.3 – Toate răspunsurile la sondaj



Figura 5.1.4 – Răspunsurile programatorilor experimentați la sondaj

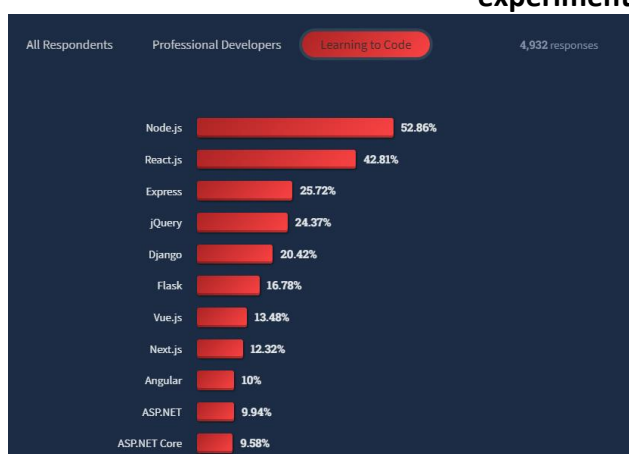


Figura 5.1.5 - Răspunsurile programatorilor aflați la început de carieră la sondaj

- **SignalR**

Sistemul de chat are nevoie de relaționare în timp real în cadrul aplicației, iar pentru aceasta a fost utilizat SignalR, o bibliotecă pentru dezvoltatorii ASP.NET care simplifică procesul

de adăugare a funcționalității web în timp real la aplicații. Funcționalitatea web în timp real implică capacitatea unui server de a transmite instantaneu conținut către clienții conectați imediat ce acesta devine disponibil, fără a aștepta o solicitare explicită pentru date noi de la un client.

Această bibliotecă poate fi folosită pentru adăugarea oricărui tip de funcționalitate web în timp real, chiar dacă cel mai întâlnit exemplu este chat-ul. Sistemul de notificare este un alt exemplu de funcționalitate care funcționează în timp real și care utilizează SignalR. SignalR permite, de asemenea, dezvoltarea de tipuri complet noi de aplicații web care necesită actualizări de înaltă frecvență de la server, cum ar fi jocurile în timp real.

Un alt aspect important al SignalR este capacitatea sa de a gestiona conexiuni (de exemplu, evenimente de conectare și deconectare) și de a grupa conexiunile. Aceasta este esențială, deoarece există momente în care nu toți utilizatorii trebuie să fie afectați de schimbările din aplicație, ci doar un grup predefinit de utilizatori.

În figura 5.1.6 se poate observa o reprezentare grafică a procesului realizat de SignalR în ceea ce privește acțiunile în timp real. [8]

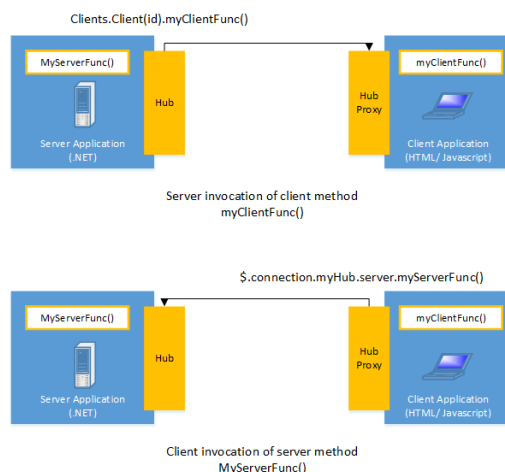


Figura 5.1.6 Modul de functionare a lui SignalR

SignalR este o bibliotecă care gestionează automat conexiunile și permite transmiterea de mesaje către toți clienții conectați simultan, similar unui chat. Poate și trimite mesaje clienților specifici, menținând o conexiune persistentă cu aceștia. Suportă funcționalitatea "server push", care permite codului server să apeleze codul client în browser prin intermediul Apelurilor de Procedură la Distanță (RPC). În plus, SignalR poate scala la mii de clienți folosind provideri de scalare, atât încorporați cât și externi.

Stim cu toți că WebSocket-ul este folosit pentru comunicare pe internet care permite schimbul bidirecțional de date între un client și un server. WebSocket este un protocol de comunicare care diferă de HTTP/HTTPS prin oferirea unui canal de comunicare bidirecțională și continuă. Spre deosebire de modelul solicitare-răspuns al HTTP/HTTPS, WebSocket menține o

conexiune deschisă, ceea ce permite serverului și clientului să transmită date reciproc oricând, până când conexiunea este închisă. Aceasta este o tehnologie extrem de utilă în aplicații care necesită actualizări în timp real, cum ar fi chat-uri, jocuri online, sau alte aplicații interactive.

SignalR folosește transportul WebSocket când este disponibil, dar se poate adapta la tehnologii mai vechi dacă este necesar. Avantajul folosirii SignalR constă în faptul că multe din funcționalitățile suplimentare necesare sunt deja implementate. Prin utilizarea SignalR, aplicația poate beneficia de WebSocket fără a necesita cod separat pentru clienții cu tehnologie mai veche. În plus, SignalR se ocupă de actualizările la WebSocket, asigurând o interfață consistentă pentru aplicație, indiferent de versiunea WebSocket. [8]

5.2. Angular

- **Angular**

Angular este un cadru (framework) de dezvoltare pentru construirea aplicațiilor web. A fost dezvoltat și este întreținut de Google. Lansat în 2010, Angular a devenit o platformă de dezvoltare software extrem de populară și bine primită, cunoscută pentru abilitățile sale puternice în construirea de aplicații de tip single-page (SPA), adică aplicații care se încarcă o singură dată și se actualizează dinamic pe măsură ce utilizatorul interacționează cu aplicația.

Există numeroase motive care favorizează utilizarea lui Angular față de alte framework-uri disponibile pe piață, cum ar fi React.js sau Vue.js. Codul scris în Angular, care poate fi JavaScript sau TypeScript, împreună cu structura sa caracteristică, îl fac intuitiv, ușor de înțeles și ideal pentru dezvoltatorii front-end. De asemenea, Angular oferă elemente HTML/CSS care pot fi utilizate aproape independent. Prin utilizarea Angular, se pot crea soluții scalabile care să răspundă eficient nevoilor clienților.

Un alt avantaj al lui Angular constă în lansarea constantă de noi versiuni și actualizări. De la prima sa versiune lansată în 2010, Angular a ajuns la a 16-a versiune în mai 2023. După Angular 2, cadru de lucru a adoptat versionarea semantică, ceea ce înseamnă că noile lansări:

- Apar în mod regulat, la fiecare șase luni, conform figurii 5.2.1.
- Sunt compatibile invers, adică o nouă versiune poate utiliza fișiere și date create de versiunile anterioare.
- Includ șase luni de suport activ cu actualizări periodice și 12 luni de suport pe termen lung, în timpul cărora sunt lansate doar corecții importante. [9]



ANGULAR VERSIONS OVER THE TIME



Figura 5.2.1 Versiunile de Angular de-a lungul timpului

Astfel, în luna mai a anului 2023, Angular a lansat versiunea 16 și are planificată lansarea versiunii 17 în luna noiembrie a aceluiași an. Acest lucru înseamnă că Angular încearcă să țină pasul cu tehnologia, facilitând utilizarea versiunilor pe perioade mai lungi de 6 luni.

Pe lângă toate acestea, Angular are și alte avantaje, precum:[9]

- **Cross-Platform:** Unul dintre punctele forte ale lui Angular este faptul că este o platformă universală ce permite crearea atât de aplicații web, cât și pentru dispozitivele mobile. Acesta are multe avantaje în comparație cu celelalte platforme menționate anterior și este unul dintre cele mai populare framework-uri.
- **Angular Material:** Angular a creat o bibliotecă open-source de componente UI care facilitează munca programatorilor. Această bibliotecă facilitează crearea de interfețe prietenoase și funcționale în raport cu utilizatorul. Astfel, se facilitează crearea de design-uri mobile atractive folosind componente reutilizabile care oferă un aspect unitar în toate aplicațiile. Angular Material facilitează construcția de aplicații responsive prin utilizarea componentelor și widget-urilor sale preconstruite. Acestea au o caracteristică importantă, și anume faptul că sunt personalizabile, iar programatorul își poate adapta design-ul pentru aplicația la care lucrează.
- **High Speed and Performance:** Aplicațiile care utilizează Angular se încarcă instantaneu, fiind aproape imposibil de egalat. Viteza de încărcare a platformei asigură că aplicațiile la care se lucrează sunt mereu active datorită caracteristicilor lor de încărcare instantanee. O caracteristică remarcabilă este că codul poate fi redat în CSS și HTML, iar aplicația este accesibilă de pe orice platformă, inclusiv .Net, NodeJs și PHP.
- **Command Line Interface:** Interfața de Linie de Comandă (CLI) a Angular respectă practicile standard de dezvoltare front-end din industrie și include capacități unice, cum ar fi rutarea și suportul SCSS. Interfața CLI standard a Angular, cum ar

fi ng-add și ng-new, facilitează dezvoltatorilor găsirea caracteristicilor predefinite.

Figura 5.2.2 de mai jos prezintă toate celelalte beneficii ale utilizării lui Angular, fiind selectate cele mai importante aspecte despre care s-a și discutat în acest context. [9]

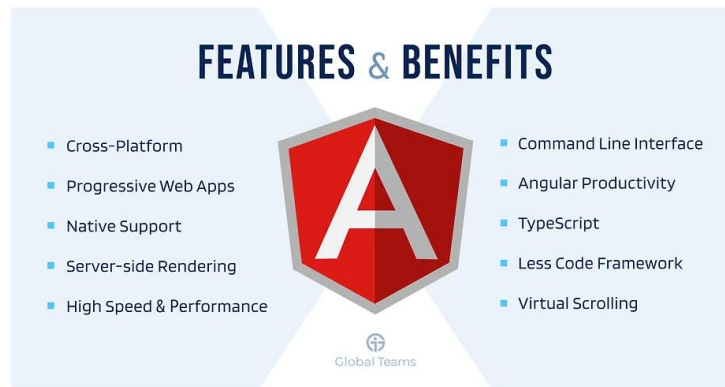


Figura 5.2.2 Beneficiile lui Angular

Conform statisticilor din anul 2022 de pe site-ul StackOverflow, Angular devine din ce în ce mai apreciat de către dezvoltatorii profesioniști, cu o utilizare de 20.39% în rândul acestora. Acest lucru se poate observa în figura 5.2.3. A existat o tendință constantă pe piață pentru cadrul de lucru front-end Angular în ultimii doi ani. [9]

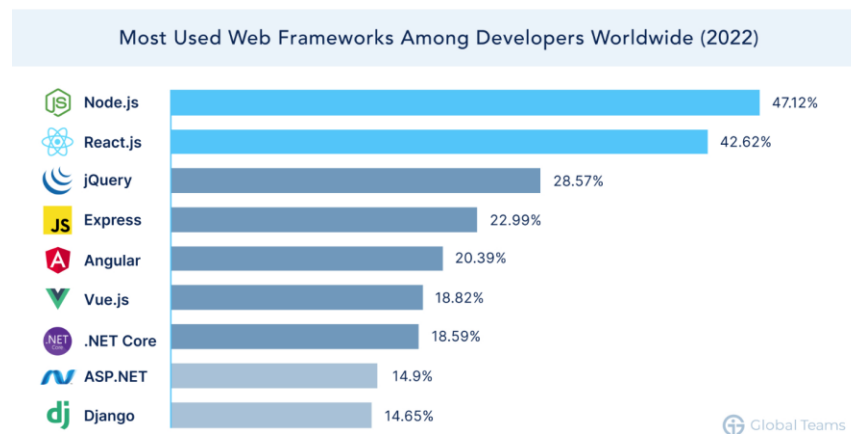


Figura 5.2.3 Cele mai utilizate Framework-uri web in anul 2022

Companii mari precum Microsoft, PayPal și Forbes au trecut pe Angular datorită numeroaselor avantaje pe care Angular le aduce în discuție. Pentru Gmail, trecerea la Angular a însemnat obținerea unei simplități greu de egalat când vine vorba de inbox-urile personale. Evident, și pentru PayPal avantajele nu au întârziat să apară, oferind o construcție a paginilor web structurată, eficientă și care prezintă o evoluție a gestionării datelor tranzacționate.

Astfel, având în vedere toate cele menționate anterior, Angular este un framework ce devine din ce în ce mai utilizat în cadrul companiilor și care oferă versiuni noi constant, ducând la evoluția treptată a framework-ului.

- **Typescript**

TypeScript este un limbaj de programare open-source dezvoltat și menținut de Microsoft. Este un superset strict al JavaScript și adaugă tipizare statică și obiecte orientate pe obiecte la JavaScript. Fiind un limbaj de programare puternic tipizat, TypeScript permite o depanare mai ușoară (la momentul compilării), ceea ce este o abordare mai eficientă pentru codificarea aplicațiilor complexe.

Complexitatea proiectelor bazate pe JavaScript crește în mod exponențial. Inițial, pe partea de client era folosit exclusiv JavaScript ca și limbaj de programare. Însă, cu trecerea timpului s-a observat că JavaScript se poate folosi și pentru partea de server.

Cu toate că JavaScript are numeroase avantaje, acesta poate deveni ușor confuz și implicat pe partea de server, mai ales în cazul aplicațiilor de mari dimensiuni. Un alt minus al lui JavaScript este faptul că a complicat procesul de întreținere al aplicațiilor mai mari și mai complexe.

Perfecționările aduse browserelor și compatibilitatea dintre acestea necesită, de asemenea, modificări în ceea ce privește JavaScript-ul de bază, dar abordarea unui JavaScript pentru viitor nu este fezabilă. Prin urmare, TypeScript a fost introdus pentru a răspunde acestor nevoi specifice ale aplicației. [10]

Cu toate acestea, oare TypeScript este o alegere mai bună când vine vorba de alegerea limbajului de programare pentru partea de front-end? Din felul în care l-am descris, ar părea o alegere mai inspirată, fiind considerat o versiune mai bună. Deși am putea crede că TypeScript va înlocui JavaScript în viitorul apropiat, acest lucru nu se va întâmpla deoarece JavaScript își va avea locul.

Un factor care nu determină trecerea în viitorul apropiat de la JavaScript la TypeScript este complexitatea, care trebuie luată în considerare când vine vorba de o astfel de schimbare.

Când vine vorba de aplicații mai simple, alegerea este la fel de simplă, deoarece JavaScript funcționează foarte ușor pe toate platformele. Pe de altă parte, compilarea codului TypeScript necesită resurse ale CPU-ului considerabil mai mari decât ar necesita JavaScript. Ocazional, poate fi nepotrivit sau excesiv să se folosească TypeScript pentru un anumit proiect.

TypeScript facilitează procesul de refactorizare a codului și pune în evidență tipurile într-un mod mai direct, ceea ce ajută dezvoltatorul să înțeleagă modul în care diferitele componente se conectează. Datorită capacității sale de a depana în timpul compilării, acesta constituie un beneficiu remarcabil pentru entitățile care administrează aplicații extinse și complexe.

Configurarea TypeScript pentru orice proiect este simplă, iar unele framework-uri folosesc TypeScript în mod implicit. Un exemplu concret de framework ce folosește ca limbaj

implicit TypeScript-ul este Angular. Deci, TypeScript are un avantaj considerabil în fața lui JavaScript la acest capitol. [10]

Diferența principală între JavaScript și TypeScript constă în faptul că JavaScript nu dispune de un sistem de tipuri. În JavaScript, variabilele pot schimba forma în mod aleatoriu, în timp ce TypeScript în modul strict interzice acest lucru. Acest lucru face ca TypeScript să fie mai ușor de gestionat și întreținut, în special cu o bază de cod mare. Acest lucru poate avea consecințe destul de mari odată cu implementarea de noi funcționalități care necesită ca datele să fie de un anumit tip.

Următoarele sunt caracteristicile TypeScript:

- TypeScript este un limbaj de programare open-source creat de Microsoft care adaugă tipuri pentru JavaScript. Este mai complex în comparație cu JavaScript, care este un limbaj de programare la nivel înalt pentru scripting.
- TypeScript este un limbaj cu un sistem de tipuri static. Adică, înainte de a utiliza variabilele și obiectele, trebuie să specificați tipul lor. Iar JavaScript nu necesită specificat tipul de date, fapt ce poate duce la anumite bug-uri sau chiar probleme ale datelor sensibile.
- TypeScript extinde JavaScript cu tipuri opționale, clase și module.
- TypeScript include instrumente pentru dezvoltarea de aplicații JavaScript la scară largă care nu depind de browser, gazdă sau sistem de operare.
- TypeScript este compilat în JavaScript lizibil, conform standardelor.
- Deoarece TypeScript extinde sintaxa JavaScript, orice program existent JavaScript va rula în TypeScript fără modificări.
- Puteți instala compilatorul TypeScript pentru linia de comandă ca un pachet Node.js.

În cadrul figurii 5.2.4 se poate observa o tendință de trecere la TypeScript de la lansarea sa până în anul 2022, urcând treptat de pe poziția a 10-a în anul 2014, pe poziția a 7-a în 2018, apoi pe poziția a 5-a în 2019, iar din 2020 s-a menținut pe poziția a 4-a, imediat după Java. Tot în figura 5.2.4 putem observa consistența poziției de lider a lui JavaScript. [11] Tocmai de aceea, TypeScript nu va înlocui complet JavaScript, ci se va alege cel mai potrivit pentru un anumit proiect, luând în considerare cele prezentate mai sus.

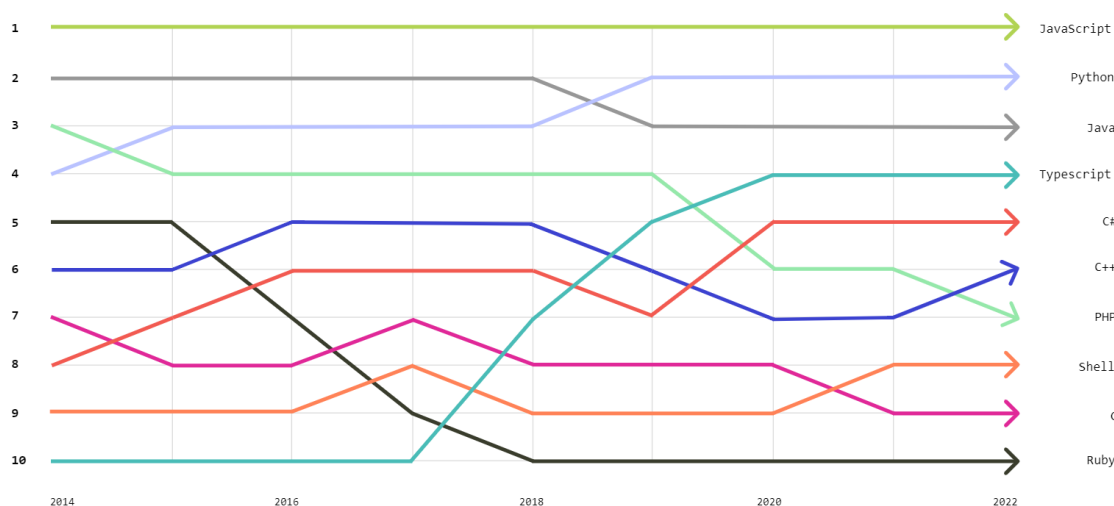


Figura 5.2.4 The top programming languages

Având în vedere cele prezentate pentru proiectul InternHub, am decis să utilizez TypeScript în detrimentul lui JavaScript. Motivul principal este complexitatea proiectului, care poate fi manipulată cu o mai mare ușurință folosind TypeScript. De asemenea, un alt motiv este creșterea popularității sale în rândul programatorilor mai experimentați.

5.3. Microsoft SQL Server

Microsoft SQL Server este un sistem de gestionare de baze de date relaționale (RDBMS) produs de compania americană Microsoft Corp, folosind SQL ca limbaj de interogare, împreună cu extensia T-SQL. Acest sistem de gestionare a bazelor de date este recunoscut în special pentru abilitatea sa de a gestiona baze de date de dimensiuni mari și este considerat potrivit pentru aplicații mai complexe care necesită o stocare mai mare de date. Cu toate acestea, ultimele versiuni ale Microsoft SQL Server beneficiază de licențiere variată, inclusiv o variantă gratuită, ceea ce facilitează lucrul și cu baze de date mai mici.[12]

Un alt plus notabil pentru Microsoft SQL Server este suportul extins, care oferă o soluție fiabilă în gestionarea eficientă a datelor în diverse medii și aplicații. Astfel, Microsoft SQL Server este considerat unul dintre cele mai bune sisteme de gestionare a bazelor de date, având lansări constante de variante care beneficiază de suport pe o perioadă destul de mare.[12]

Microsoft a lansat recent SQL Server 2022, bazându-se pe versiunile anterioare, cu scopul de a crea o platformă cu numeroase opțiuni în ceea ce privește limbajele de dezvoltare, tipurile de date, sistemele de operare și mediile de lucru pe servere, atât fizice cât și în cloud.[13]

Printre numeroasele beneficii se numără:[14]

- **Continuitatea afacerii prin intermediul Azure:** Microsoft SQL Server oferă o gestionare eficientă în cloud, asigurând protecția datelor în fața oricăror dezastre care pot apărea, astfel încât datele utilizatorilor să nu fie pierdute în niciun caz.

- **Analize fără întreruperi asupra datelor operaționale de pe serverele fizice:** Microsoft SQL Server oferă posibilitatea generării în timp real a statisticilor pentru toți utilizatorii prin intermediul lui Spark.
- **Vizibilitate asupra întregului set de date:** Ajută la gestionarea întregului volum de date, prevenind depășirea silozurilor de date prin intermediul lui Microsoft PurView.
- **Cea mai sigură bază de date din ultimii 10 ani:** Microsoft SQL Server este evaluat ca fiind cel mai puțin vulnerabil sistem de gestionare al bazelor de date în ultimii 10 ani, folosind un mediu stabil pentru protecția datelor împotriva falsificării.
- **Performanță și disponibilitate lider în industrie:** Microsoft SQL Server oferă performanță pentru interogări mai rapide, fără a necesita modificarea codului, ceea ce duce la aplicații mai eficiente.

Având în vedere cele prezentate mai sus, este destul de clar de ce am ales să folosesc Microsoft SQL Server în detrimentul altor sisteme de gestionare a bazelor de date, precum PostgreSQL. Evident, ambele au avantaje și dezavantaje, dar pentru InternHub, Microsoft SQL Server s-a potrivit mai bine nevoilor proiectului. Un aspect important este faptul că are un punct comun cu .NET, fiind create de aceeași companie mamă, ceea ce facilitează implementarea și comunicarea între ele.

Într-un studiu realizat de cei de la StackOverflow, care include 63,327 de răspunsuri de la programatori cu diferite nivele de experiență, se poate observa că Microsoft SQL Server nu este la fel de preferat ca PostgreSQL atunci când vine vorba de alegerea unui sistem de gestionare al datelor. Este clasat doar pe poziția a 5-a, cu un procent de 26.78% din toate răspunsurile. Cu toate acestea, în cazul programatorilor experimentați, poziția lui Microsoft SQL Server se schimbă ușor, situându-se pe locul 4, cu un procent de 28.77%. Dacă ne referim la cei care învață să programeze, putem observa că mai mult de jumătate dintre respondenți aleg MySQL, cu un procent de 58.4%. Aici, Microsoft SQL Server revine pe locul 5, cu un procent de 16.34%, iar PostgreSQL se află pe o poziție mai sus, cu 25.54%. [15]

Aceste rezultate pot fi observate în figurile de mai jos, respectiv 5.3.1, 5.3.2 și 5.3.3, unde sunt prezentate toate răspunsurile din studiul realizat de cei de la StackOverflow în anul 2022. În figura 5.3.1 se pot observa toate răspunsurile la sondaj, iar celelalte două figuri sunt împărțite în programatori experimentați (figura 5.3.2) și cei care învață să programeze (figura 5.3.3).

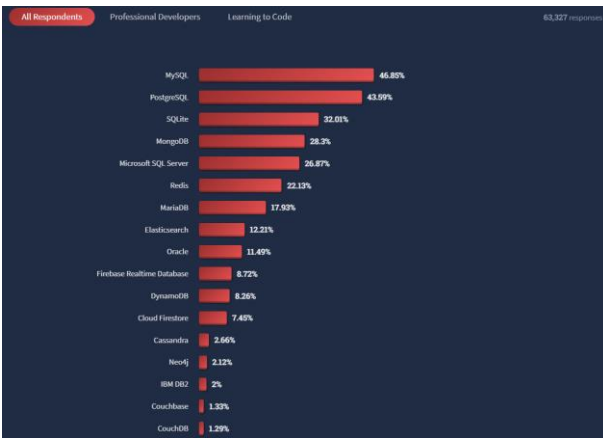


Figura 5.3.1 - Toate răspunsurile la sondaj

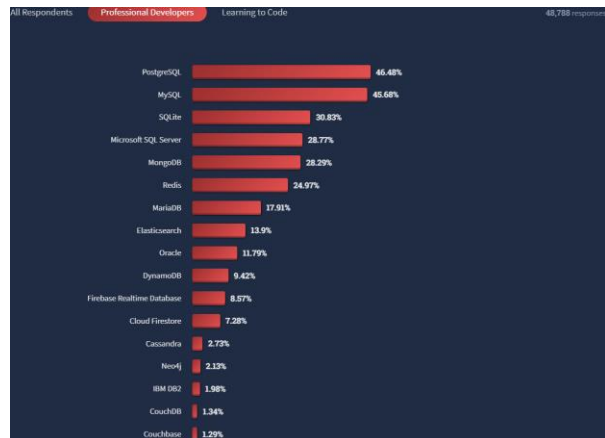


Figura 5.3.2 - Răspunsurile programatorilor experimentați la sondaj

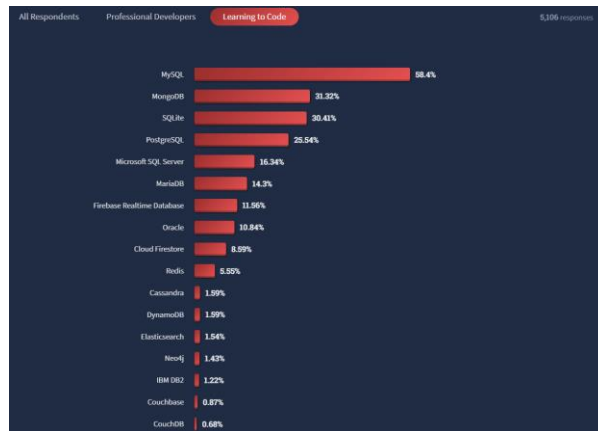


Figura 5.3.3 - Răspunsurile programatorilor aflați la început de carieră la sondaj

Pentru dezvoltatorii profesioniști, PostgreSQL a preluat recent locul întâi de la MySQL. Dezvoltatorii profesioniști sunt mai predispuși decât cei care învață să codeze să utilizeze Redis, PostgreSQL, Microsoft SQL Server și Elasticsearch. MongoDB este utilizat în mod similar atât de către dezvoltatorii profesioniști, cât și de cei care învață să codeze, fiind a doua cea mai populară bază de date pentru cei care învață să codeze (după MySQL). Acest lucru are sens, deoarece suportă un număr mare de limbaje și platforme de dezvoltare a aplicațiilor.[15]

Cu toate că Microsoft SQL Server nu se află în top 3 cele mai utilizate sisteme de gestionare, are multe avantaje care ajută la dezvoltarea aplicației InternHub, având grijă de cea mai importantă parte a oricărei aplicații, și anume securitatea datelor. În plus, așa cum am mai spus, având un producător comun cu framework-ul .NET, comunicarea între API și baza de date este mult mai ușor de realizat și, totodată, sigură.

Prin urmare, având în vedere cele prezentate mai sus, Microsoft SQL Server reprezintă opțiunea cea mai bună când vine vorba de baze de date, având avantaje considerabile în fața celorlalte opțiuni pentru proiectul de față.

5.4. Securitatea datelor

Toți știm că securitatea datelor este un subiect important în cadrul oricărei aplicații. În aplicația prezentată, avem și date sensibile care trebuie cunoscute doar de utilizatorul respectiv, cum ar fi parola contului. Am ales un algoritm care criptează aceste valori folosind o cheie predefinită în proiect. Avantajul algoritmului de criptare utilizat în aplicație este HMACSHA512, care are ca avantaj principal faptul că este destul de greu de decriptat. În acest caz, accesul în contul personal se face prin generarea unui hash și a unui salt cu valoarea transmisă de utilizator. Dacă aceste două valori corespund celor din tabel, atunci utilizatorul este autorizat, iar în caz contrar nu i se permite accesul.

HMACSHA512 este un algoritm de hash bazat pe chei, creat folosind funcția de hash SHA-512 și aplicat ca un Cod de Autentificare a Mesajului bazat pe Hash (HMAC). Procedura HMAC combină o cheie secretă cu datele mesajului și apoi calculează valoarea hash a combinației. Valoarea hash rezultată este apoi amestecată din nou cu cheia secretă și supusă unui al doilea proces de hash. Hash-ul rezultat are o lungime de 512 biți.

HMAC poate fi utilizat pentru a verifica dacă un mesaj trimis prin intermediul unui canal nesigur a fost compromis, atât timp cât expeditorul și destinatarul dețin o cheie secretă comună. Expeditorul calculează valoarea hash pentru datele originale și trimite ambele, datele originale și valoarea hash, ca un singur mesaj. Destinatarul recalculează valoarea hash pe baza mesajului primit și verifică dacă HMAC-ul calculat corespunde cu HMAC-ul transmis.

Dacă valorile hash originale și cele calculate se potrivesc, mesajul este autentificat. Dacă nu se potrivesc, atunci fie datele, fie valoarea hash au fost modificate. HMAC oferă protecție împotriva modificărilor, deoarece pentru a schimba mesajul și a reproduce valoarea hash corectă este necesară cunoașterea cheii secrete.

HMACSHA512 permite utilizarea de chei de orice dimensiune și produce o secvență de hash cu o lungime de 512 biți.[25]

5.5. Cloudinary

Pentru stocarea pozelor în cadrul aplicației InternHub, am optat pentru un serviciu de cloud, deoarece este mai eficient decât stocarea imaginilor în format binar în baza de date. Stocarea imaginilor direct în baza de date poate îngreuna aceasta.

Pentru aceasta funcționalitate, am ales să utilizăm Cloudinary, care este un SDK din ecosistemul .NET ce oferă capacități simple, dar cuprinzătoare de încărcare, transformare, optimizare și livrare a imaginilor și videoclipurilor. Acesta poate fi implementat folosind codul care se integrează perfect cu aplicația .NET existentă. [26]

- Cloudinary oferă o bibliotecă open source .NET pentru a simplifica și mai mult integrarea:
- Construirea URL-urilor pentru transformări de imagini și videoclipuri.
- Metode helper pentru încorporarea și transformarea resurselor.
- Învelișuri de API: încărcarea fișierelor, administrarea, generarea de sprite-uri și altele.

- Încărcarea fișierelor pe server + încărcarea directă de fișiere nesemnate din browser folosind modulul jQuery.

Biblioteca este construită pentru .NET Framework 4.x și .NET Core. Puteți utiliza orice limbaj .NET cu biblioteca, iar biblioteca în sine este scrisă în C#. Documentația noastră include exemple atât în C#, cât și în VB.NET. [26]

Cloudinary oferă funcționalități pentru prelucrarea imaginilor, precum crop, rotate, radius și multe altele. Astfel, prin intermediul Cloudinary, prelucrarea imaginilor este mult mai ușor de realizat. De asemenea, acesta poate prelucra și videoclipuri, dar acest lucru nu este de interes pentru proiectul prezentat.

Pentru partea de implementare, procesul este unul destul de simplu. Primul pas constă în instalarea pachetului specific pentru aceasta, și anume:

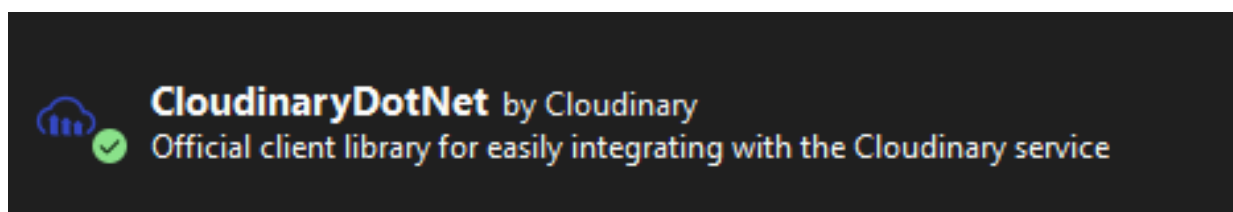


Figura 5.5.1 – Pachetul necesar pentru Cloudinary

Apoi, trebuie să creăm un cont pe platforma Cloudinary, care ne va duce în final la ecranul prezentat mai jos:

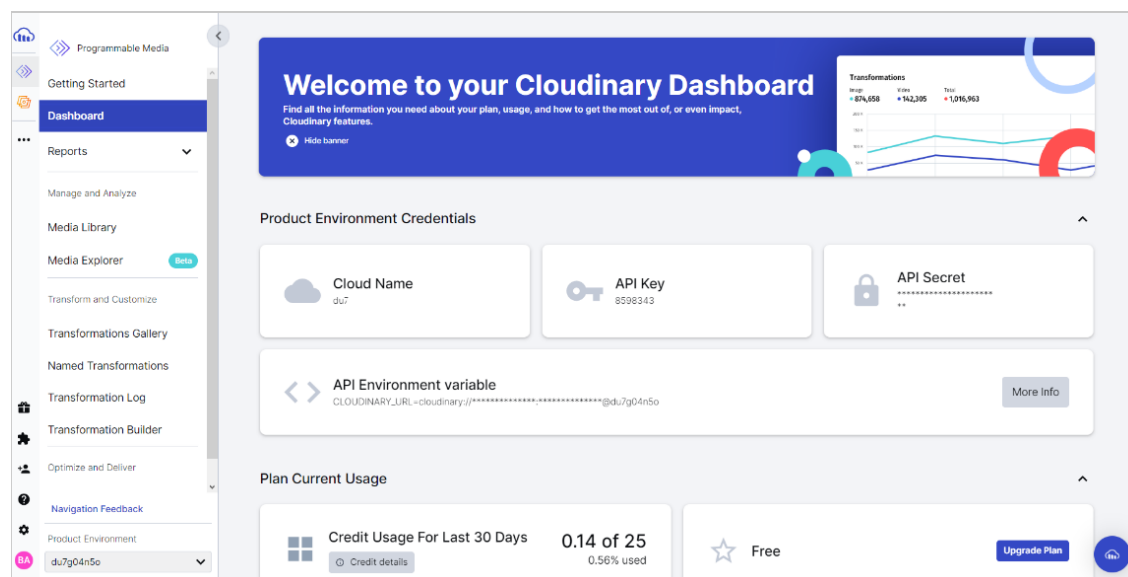
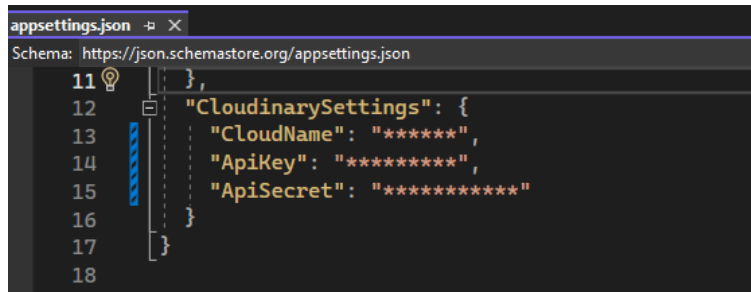


Figura 5.5.2 – Ecran principal Cloudinary

Cu datele vizibile în ecran, configurăm serviciul în API, în fișierul appsettings.json, unde adăugăm cele trei câmpuri: CloudName, APIKey și APISecret.



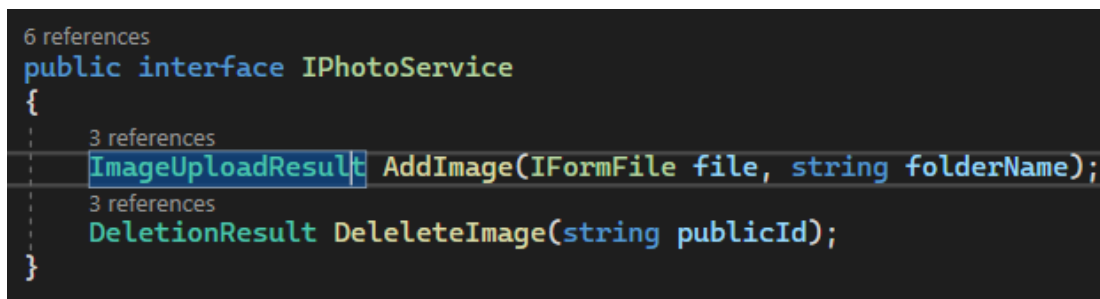
```

appsettings.json
Schema: https://json.schemastore.org/appsettings.json
11  {
12    "CloudinarySettings": {
13      "CloudName": "*****",
14      "ApiKey": "*****",
15      "ApiSecret": "*****"
16    }
17  }
18

```

Figura 5.5.3 – Configurare serviciu în appsettings.json

După acest pas, vom crea o interfață cu două metode: AddImage (de tip ImageUploadResult) care primește ca parametri imaginea și numele directorului în care va trebui să fie creată și DeleteImage (de tip DeletionResult) care primește ca parametru ID-ul imaginii, pentru manipularea imaginilor. Apoi, vom crea o clasă care să implementeze aceste două metode, conform figurilor 5.5.4 și 5.5.5.

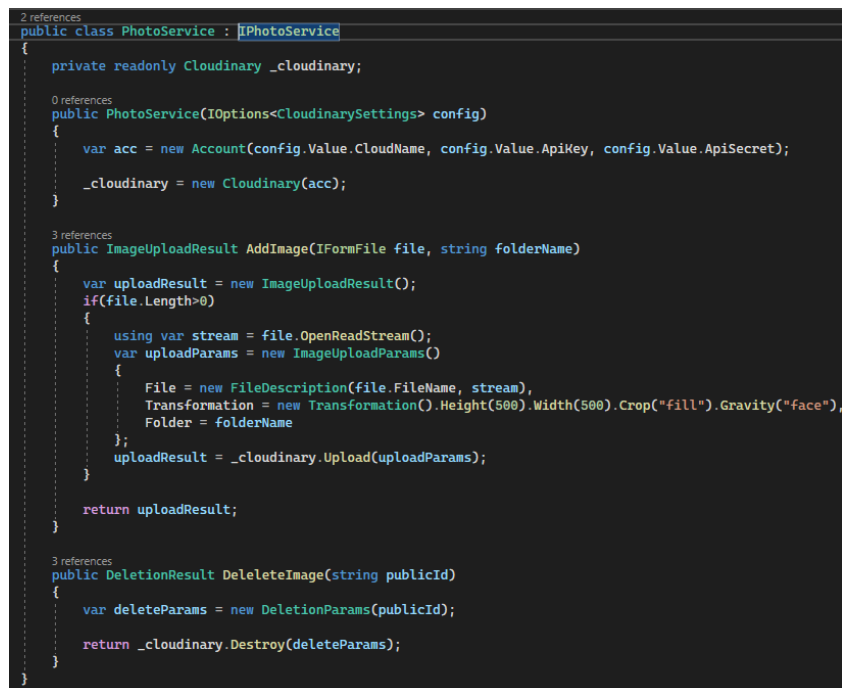


```

6 references
public interface IPhotoService
{
    3 references
    ImageUploadResult AddImage(IFormFile file, string folderName);
    3 references
    DeletionResult DeleteImage(string publicId);
}

```

Figura 5.5.4 – Interfața pentru serviciu



```

2 references
public class PhotoService : IPhotoService
{
    private readonly Cloudinary _cloudinary;

    0 references
    public PhotoService(IOptions<CloudinarySettings> config)
    {
        var acc = new Account(config.Value.CloudName, config.Value.ApiKey, config.Value.ApiSecret);
        _cloudinary = new Cloudinary(acc);
    }

    3 references
    public ImageUploadResult AddImage(IFormFile file, string folderName)
    {
        var uploadResult = new ImageUploadResult();
        if(file.Length>0)
        {
            using var stream = file.OpenReadStream();
            var uploadParams = new ImageUploadParams()
            {
                File = new FileDescription(file.FileName, stream),
                Transformation = new Transformation().Height(500).Width(500).Crop("fill").Gravity("face"),
                Folder = folderName
            };
            uploadResult = _cloudinary.Upload(uploadParams);
        }

        return uploadResult;
    }

    3 references
    public DeletionResult DeleteImage(string publicId)
    {
        var deleteParams = new DeletionParams(publicId);

        return _cloudinary.Destroy(deleteParams);
    }
}

```

Figura 5.5.5 – Implementarea interfeței

În cadrul clasei se realizează și configurarea conexiunii cu serviciul folosind datele de conectare declarate în appsettings.json.

Pentru ultimul pas al configurării, adăugăm atât serviciul, cât și interfața în configuratorul API-ului.

```
services.Configure<CloudinarySettings>(config.GetSection("CloudinarySettings"));
services.AddScoped<IPictureRepository, PictureRepository>();
```

Figura 5.5.6 – Finalizare configurare

Acum că avem serviciul configurat, putem să-l folosim. Astfel, în controller avem o funcție care actualizează poza, primind ca parametru ID-ul obiectului la care dorim să adăugăm poza, iar poza este primită prin body.

Se verifică dacă a fost primită o imagine și dacă este un tip acceptat pentru acest lucru, în cazul de față să fie o imagine. Apoi se verifică dacă obiectul are o imagine stabilită, care este ștearsă pentru a salva cea nouă.

```
IFormFile file = Request.Form.Files[0];

if (file == null || file.Length == 0)
    return BadRequest("No image received!");

if (!file.ContentType.StartsWith("image/"))
    return BadRequest("File is not a valid image!");

if (user.PictureId != null)
{
    var res = _photoService.DeleteImage(_pictureRepository.GetById(user.PictureId.Value).PublicId);
    if (res.Error != null) return BadRequest(res.Error.Message);
}

var result = _photoService.AddImage(file, "InternHub/Users");

if (result.Error != null) return BadRequest(result.Error.Message);
```

Figura 5.5.7 - Actualizarea poza

Astfel se manipulează imaginile în aplicația InternHub, fiind o metodă atât mai ușoară, cât și mai eficientă față de celelalte variante disponibile.

6. Prezentarea aplicației

6.1. Introducere

Aplicația InternHub este un portal creat pentru interni, pentru studenții care se află în stagiul de practică, cu scopul de a-i ajuta pe mentorii care îi au pe studenți în mentorat să le urmărească evoluția, pentru a le putea oferi un calificativ pentru perioada de practică. Apoi, acest calificativ, împreună cu o descriere a activității realizate pe parcursul acestei perioade, este dus la profesorul responsabil de practică de la universitatea la care studenții aparțin.

Pentru a facilita acest proces, este inclus și un proces de vizualizare a procesului de evaluare de către profesor. Acest pas este esențial pentru transparență, deoarece cei care se ocupă de programul de practică nu știu cum a evoluat de fapt studentul în realitate. Astfel, prin implicarea directă a profesorului coordonator de practică, avem parte de o transparență mărită, fiind în măsură și profesorul să ofere un calificativ studentului. Evident, nu știe absolut tot, ceea ce este și imposibil, dar are suficiente informații și este în măsură să ofere, dacă nu o notă, măcar un calificativ.

Aplicația are ca scop principal facilitarea procesului de evaluare după perioada de probă, iar ca scop secundar, aplicația își propune să motiveze resursa umană și să ajute la evoluția acestora în cadrul companiei prin toate funcționalitățile de care dispune. Deoarece tot timpul este loc de învățat, această aplicație își propune să contribuie și la dezvoltarea programatorilor experimentați. Acest lucru se poate realiza fie de la alți colegi cu o experiență similară, fie chiar de la cei aflați în stadiul de practică.

6.2. Structura aplicației

Din punct de vedere structural, aplicația este una full-stack, folosind Angular ca front-end și .NET ca back-end. Sistemul de gestionare a bazelor de date ales este Microsoft SQL Server. Comunicarea între client și server se realizează prin intermediul request-urilor, care au formatul JSON, iar pentru securizarea lor s-a utilizat Bearer token împreună cu JWT. Toate aceste informații au fost deja discutate și prezentate mai sus în capitolele anterioare.

Din punct de vedere al utilizatorilor, avem 4 tipuri de utilizatori: Interni, Traineri, Administratori și Profesori. Fiecare dintre aceștia au drepturi de acces diferite și au ecrane diferite în aplicație. De asemenea, au și operații comune, după cum vom vedea.

Până să vedem ce poate face fiecare în mod separat, vom trece prin atribuțiile comune pe care le au toți acești utilizatori, și anume:

- Crearea/Editarea/Stergerea grupurilor de discuții
- Inițierea discuțiilor 1-la-1 cu ceilalți utilizatori
- Crearea/Editarea/Stergerea postărilor pe forum
- Crearea/Editarea/Stergerea răspunsurilor la postări de pe forum
- Reacționarea la postări și comentarii prin upvote sau downvote
- Crearea/Editarea/Stergerea anunțurilor de pe tab-ul "Notes"

Administratorul are drepturi depline asupra întregii aplicații. El se ocupă de gestionarea utilizatorilor și a grupurilor. El mai poate face:

- Crearea/Editarea/Stergerea utilizatorilor
- Crearea/Editarea/Stergerea grupurilor, la care se poate adăuga/elimina studenți din grupe

Trainerul are drepturi de acces pe aproape toate datele, cu excepția gestionării utilizatorilor și a grupurilor. El mai poate face:

- Crearea/Editarea/Ștergerea challenge-urilor zilnice
- Aprobarea/Respingerea răspunsurilor încărcate de studenți
- Oferirea de puncte pentru fiecare challenge finalizat cu succes
- Crearea/Editarea/Ștergerea feedback-ului oferit studenților
- Crearea/Editarea/Ștergerea întrebărilor pentru teste
- Crearea/Editarea/Ștergerea testelor
- Asignarea testelor
- Vizualizarea rezultatelor studenților la teste postate de ei
- Crearea/Editarea/Ștergerea meeting-urilor, unde poate alege fie persoane individuale, fie grupuri
- Crearea/Editarea/Ștergerea task-urilor și subtask-urilor
- Asignarea de task-uri

Internii au și ei roluri limitate, având următoarele drepturi:

- Încărcarea soluțiilor pentru challenge-uri de tip .zip sau .rar
- Pot finaliza task-urile
- Pot parcurge testele atribuite
- Pot participa la meeting-urile atribuite
- Pot vedea soluțiile încărcate de ei

Profesorii au și ei drepturi limitate, având posibilitatea doar de a vizualiza datele, fără a avea acces de scriere/editare. Astfel, ei pot:

- Vizualiza soluțiile încărcate de studenți
- Vizualiza evoluția task-urilor
- Pot oferi feedback studenților

Aceasta este structura aplicației la nivel de utilizatori. Drepturile de acces pentru resursele de pe server sunt gestionate pe server, dar despre acest subiect am discutat mai mult în capitolul 4, subcapitolul 4.6.

La nivel de ecrane, avem mai multe ecrane pentru fiecare tip de utilizator, iar fiecare utilizator are un număr diferit de tab-uri. Singura excepție este pentru profesor și student, care au aceleași tab-uri. Despre acest aspect al ecranelor o să vorbim în subcapitolul următor.

6.3. Prezentarea aplicației

Aplicația este hostată local pe portul <https://localhost:4200>, care este standard pentru orice aplicație. Pentru aplicația InternHub, acestui link îi este atribuit componenta de autentificare. Astfel, pentru orice tip de utilizator, aceasta este prima pagină pe care o vizualizează. Acest ecran este prezentat în figura 6.3.1.

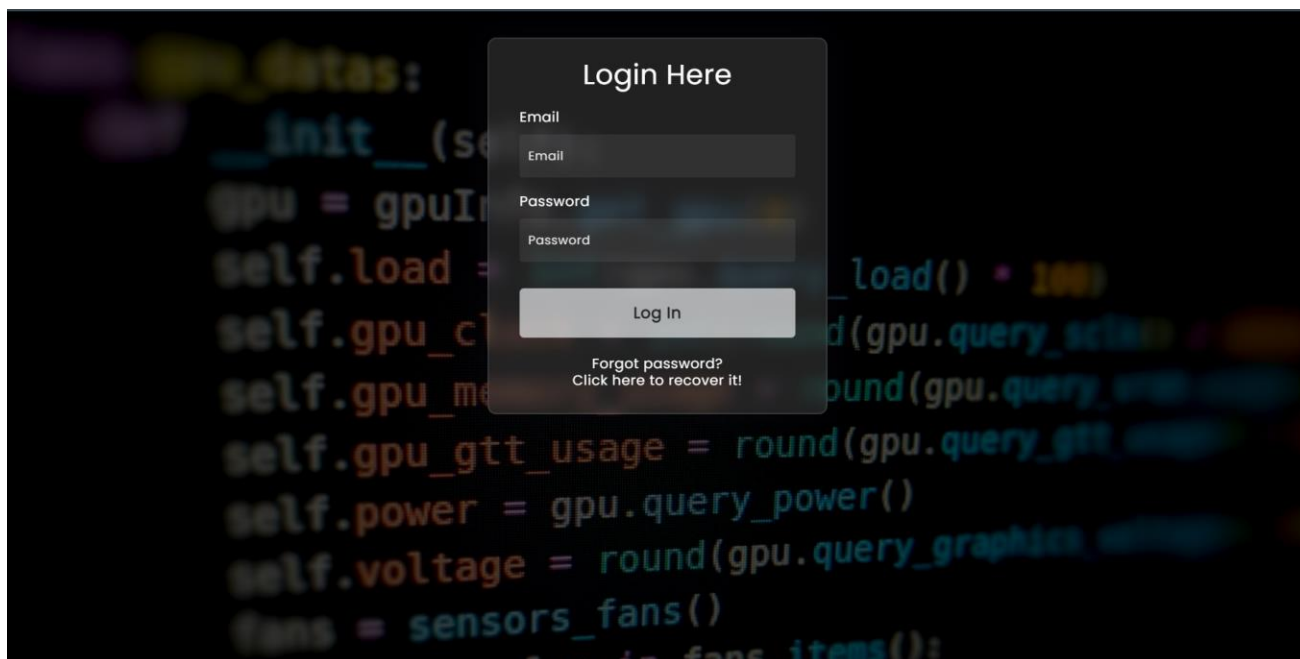


Figura 6.3.1 – Pagina de autentificare

Prima dintre cele două posibile acțiuni este să ne autentificăm cu datele contului. În cazul în care acestea nu sunt corecte, vom fi informați cu privire la această problemă. În cazul în care datele introduse sunt corecte, vom fi redirecționați către ecranul principal al aplicației. Ecranul principal este alcătuit din componente care vin în trei variante: una pentru administrator, una pentru interni și profesori și încă una pentru traineri. Acest lucru este ilustrat în figurile 6.3.2 pentru administrator, 6.3.3 pentru interni și profesori și 6.3.4 pentru traineri.

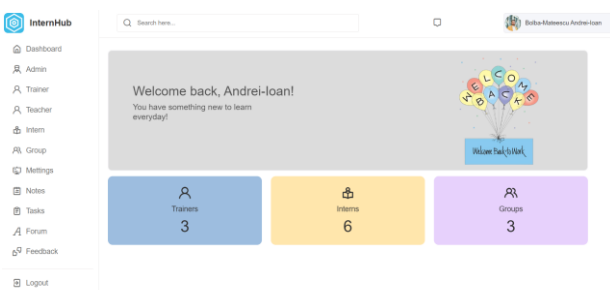


Figura 6.3.2 – Ecran principal admin

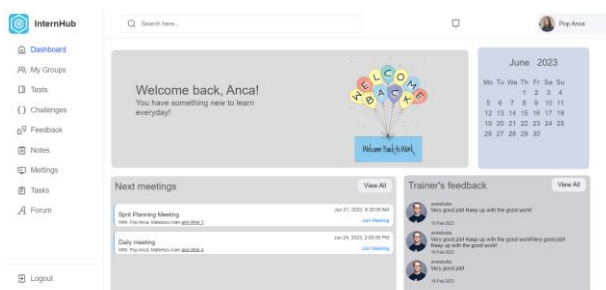


Figura 6.3.3 – Ecran principal intern/profesor

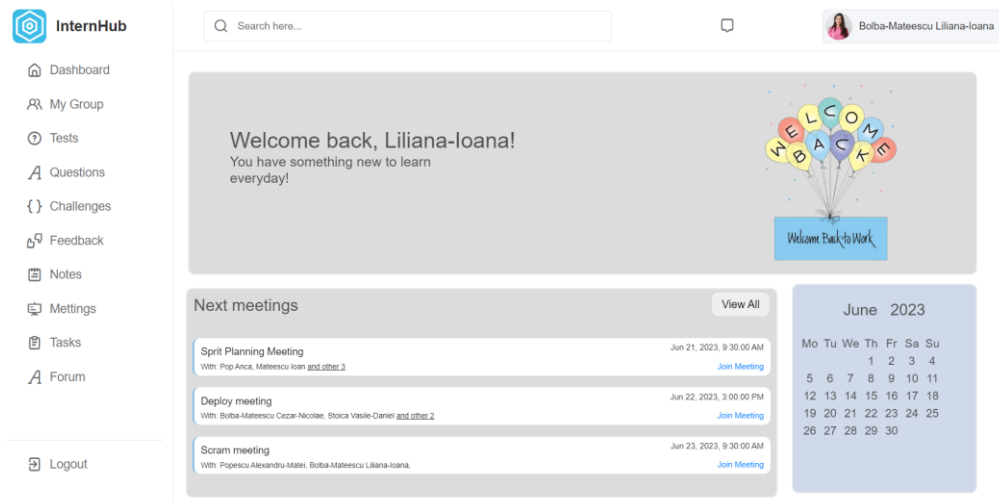


Figura 6.3.4 – Ecran principal trainerii

A doua posibilitate este cea de recuperare a parolei, care ne va duce către un ecran similar unde vom introduce adresa de email. Acest ecran este prezentat conform figurii 6.3.5. Apoi vom primi un email de resetare a parolei care ne va duce către un ecran unde va trebui să setăm o nouă parolă. Email-ul arată conform figurii 6.3.6, iar link-ul este valabil timp de o oră. În cazul în care link-ul a expirat, vom fi redirecționați către ecranul de autentificare conform figurii 6.3.1, cu un mesaj de tip pop-up cu mesajul "Link is expired". Ecranul în care vom reseta parola este prezentat în figura 6.3.7. După setarea noii parole, ne vom întoarce la ecranul de autentificare și vom proceda la logare. Noua parolă va trebui să îndeplinească un set de caracteristici care au ca scop crearea unei parole puternice din punct de vedere al securității:

- Minim 8 caractere
- Minim o literă mică
- Minim o literă mare
- Minim o cifră
- Minim un caracter special

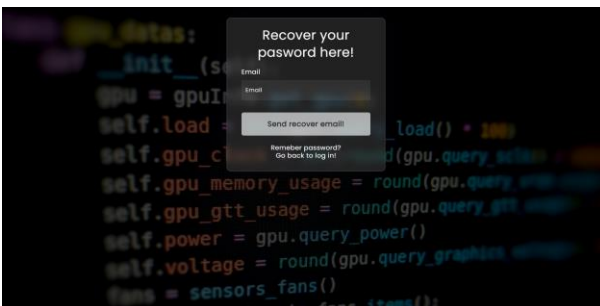


Figura 6.3.5 – Resetarea parolei

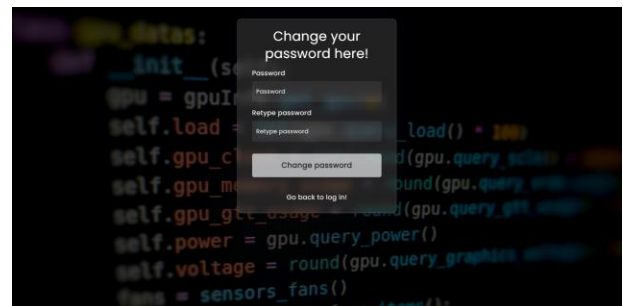


Figura 6.3.7 – Ecran de resetare al parolei

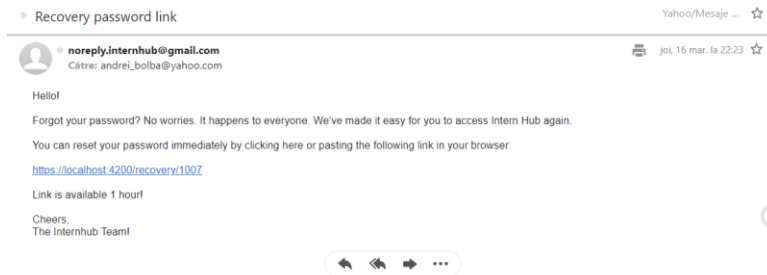


Figura 6.3.6 – Mail de resetare

Odată ajunși în ecranul principal, așa cum se poate observa, fiecare tip de utilizator are drepturi diferite, incluzând și unele comune sau parțial comune. Acestea includ chat-ul, forum-ul, pagina personală și anunțurile.

Partea de chat este disponibilă atât pentru conversații individuale (1-la-1), cât și pentru conversații în grup. La prima accesare a acestui ecran, vor apărea persoanele cu care utilizatorul a convenit să converseze, precum și grupurile din care acesta face parte. Prin selectarea unui chat, se deschide întreaga conversație și persoanele pot începe să converseze. Pentru grupuri, procedeul este identic. Acest lucru poate fi observat în figura 6.3.8 pentru primul clic, iar figura 6.3.9 pentru vizualizarea conversației.

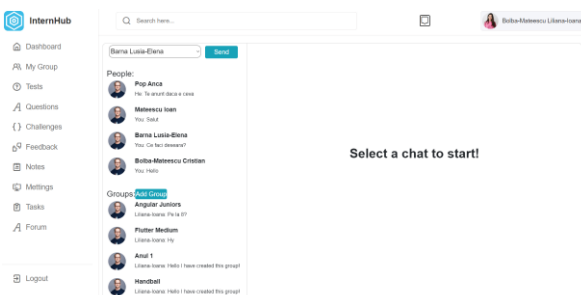


Figura 6.3.8 – Ecran principal pentru chat

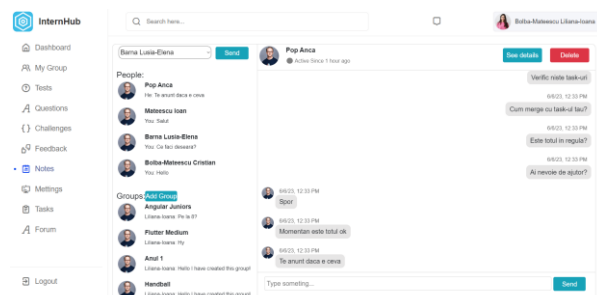


Figura 6.3.9 – Ecranul unei conversații

Pentru partea de grupuri, utilizatorii pot crea grupuri prin specificarea unui nume și, opțional, a unei descrieri, precum și prin adăugarea de membri. Persoana care creează grupul devine automat administrator și nu poate părăsi grupul fără a numi un alt membru ca administrator înainte. Modificări privind numele sau descrierea grupului pot fi realizate de către toți membrii grupului. Administratorul are puterea de a adăuga sau elimina membri din grup. Acest proces este exemplificat în figurile 6.3.10 pentru setarea numelui și descrierii grupului, 6.3.11 pentru adăugarea inițială a persoanelor în grup, și 6.3.12 pentru eliminarea acestora din grup, alături de ecranul unde se pot vedea informații referitoare la grup. În cadrul adăugării de membri, se utilizează ecranul din figura 6.3.11, cu opțiunea de a selecta persoanele deja existente în grup.

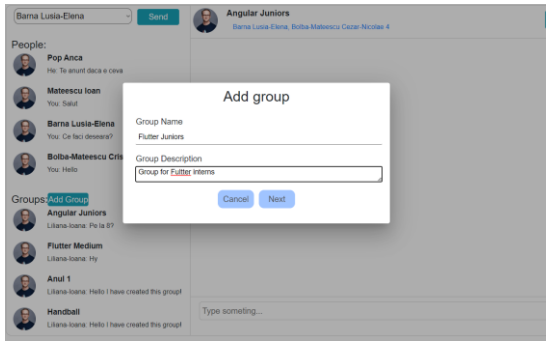


Figura 6.3.10 – Setarea numelui si a descrierii

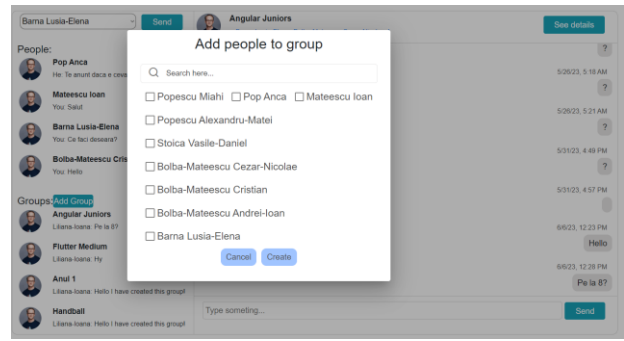


Figura 6.3.11 – Adaugarea initiala a persoanelor

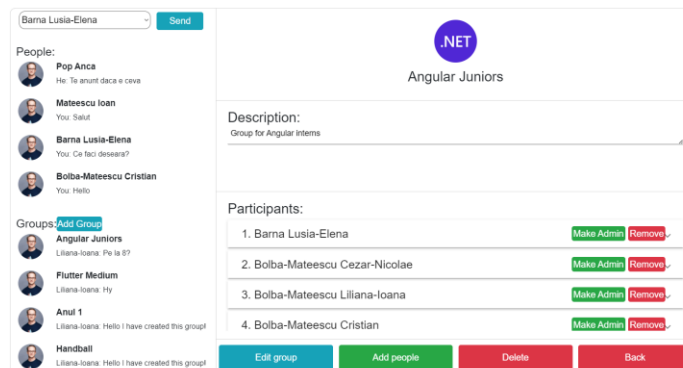


Figura 6.3.12 – Informatii privind grupul

Un alt administrator poate fi numit din rândul persoanelor existente în grup, prin apăsarea butonului "Make admin". Eliminarea se face prin intermediul butonului "Remove", iar adăugarea presupune selectarea lor după apariția butonului "Add people".

Pentru partea de anunțuri, avem un ecran unde se pot vizualiza toate anunțurile, conform figurii 6.3.13. Orice utilizator poate adăuga un anunț (figura 6.3.14), edita anunțuri (identic cu figura 6.3.14, având câmpurile completate) și șterge anunțuri prin apăsarea butonului "delete note".

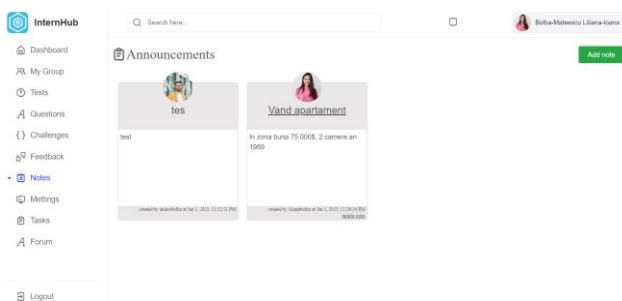


Figura 6.3.13 – Ecran principal anunturi

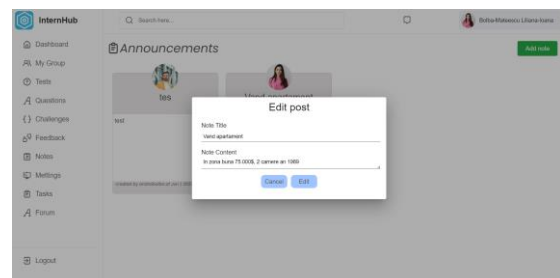


Figura 6.3.14 – Adaugare/editare anunt

Un alt feature comun este vizualizare profilul personal, care are are informatii legate de persoana in cauza. Aceste este un ecran read-only, deoarece doar administartorul are acces la aceste operatii. Singurul lucru pe care îl poate face utilizatorul este de a-și selecta poza de profil care la început este una standard. Acest ecran de actualizare este în figura 6.3.15.2, iar ecranul profilul personal este vizibil in figura 6.3.15.

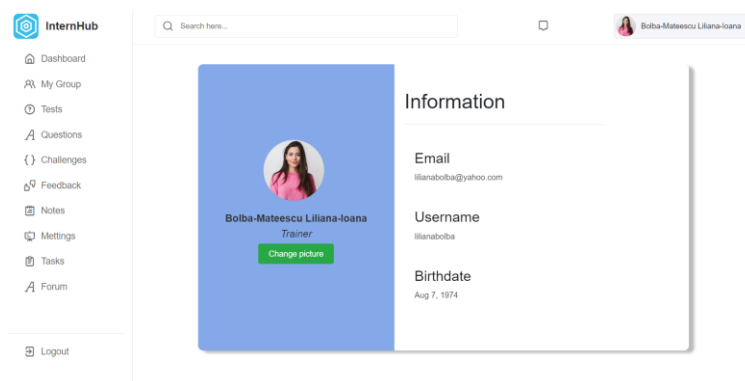


Figura 6.3.15 - Ecranul profilul personal

Ultimul feature comun pentru toți utilizatorii este forumul. Ecranul principal al forumului prezintă toate postările de pe forum în ordine cronologică, de la cea mai veche la cea mai nouă, după cum putem vedea în figura 6.3.16. Orice utilizator poate crea o postare pe forum, poate răspunde la o postare și poate acorda upvote, în cazul în care este de acord cu ce spune persoana respectivă, sau downvote în caz contrar. Adăugarea și editarea utilizează același ecran, diferența fiind popularea datelor în cazul editării, după cum putem observa și în figura 6.3.18. O postare arată similar cu figura 6.3.17, iar un comentariu similar cu figura 6.3.19. În cazul comentariilor, se aplică același sistem de adăugare și editare. De asemenea, o postare sau un comentariu pot fi șterse prin apăsarea butonului "Delete".

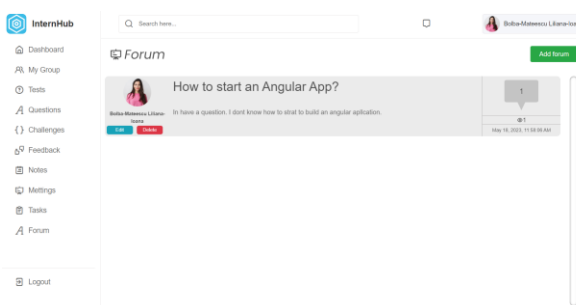


Figura 6.3.16 – Ecran principal forum

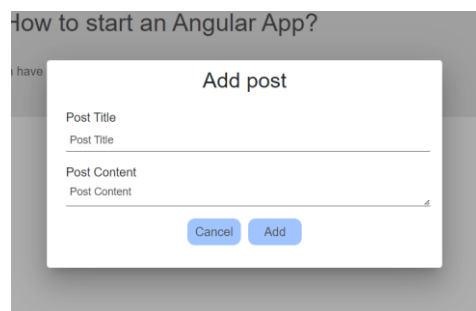


Figura 6.3.18 – Adugare.editare postare

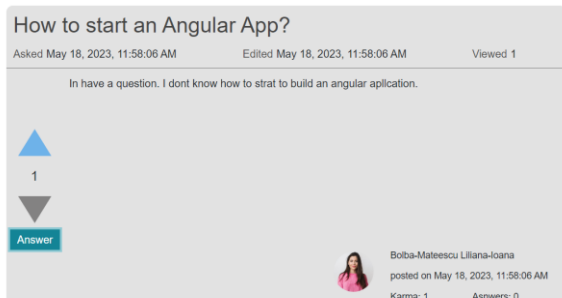


Figura 6.3.17 – Model de postare



Figura 6.3.19 – Model de comentariu

Acestea au fost feature-urile comune pentru toți utilizatorii. Acum o să trecem prin cele mai relevante feature-uri pe care le are fiecare utilizator.

Administratorul mai poate adăuga/edita/șterge conturile utilizatorilor. Din punct de vedere al ecranelor, avem un ecran comun pentru acțiunea de adăugare/editare pentru toți utilizatorii. Acest lucru este posibil datorită faptului că singura diferență între utilizatori la nivel de tabelă este statusul, care este setat când este apelat request-ul.

Astfel, tab-urile Admin, Trainer, Intern și Teacher au aceeași componentă, și anume un tabel cu toți utilizatorii cu același status, de unde se pot realiza operații CRUD pe conturile acestora. Ecranul principal este asemănător cu figura 6.3.20, iar ecranul de adăugare/editare este asemănător figurii 6.3.21.

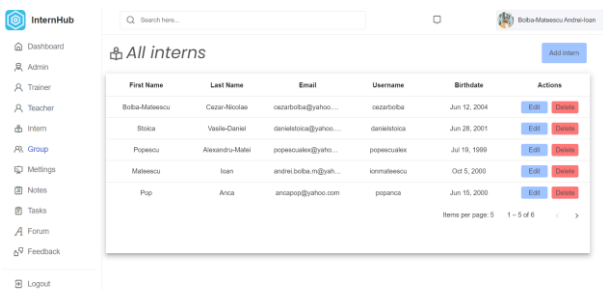


Figura 6.3.20 – Tabelul perosanelor cu acelasi status

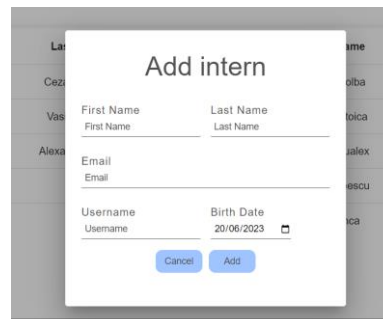


Figura 6.3.21 – Adaugare/editare conturi

Împreună cu managementul conturilor, un administrator mai poate crea/ modifica/ șterge un grup. De asemenea, poate înscrie studenți în grupuri. Ecranul principal se bazează pe aceeași idee ca și ecranul de gestionare al conturilor. Astfel, avem un tabel cu toate grupurile create (figura 6.3.22), de unde un administrator poate adăuga/edita un grup (figura 6.3.23) și, de asemenea, poate adăuga/elimina studenți dintr-o grupă (figura 6.3.24).

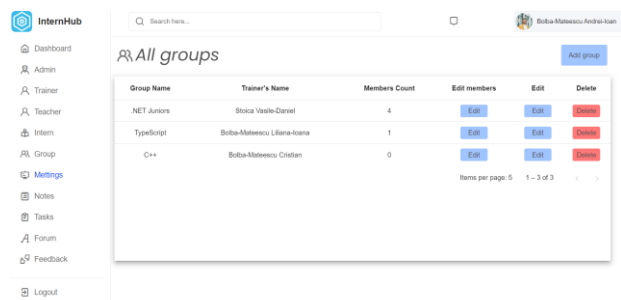


Figura 6.3.22 – Ecran principal pentru grupuri

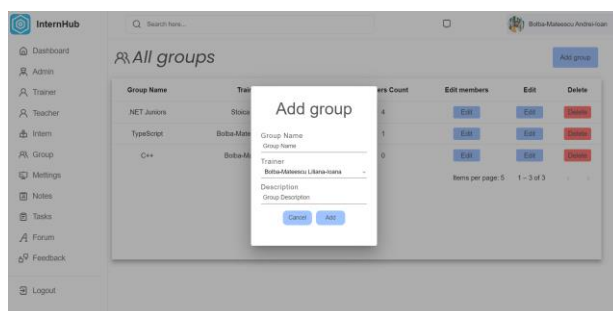


Figura 6.3.23 – Adaugare/editare grup

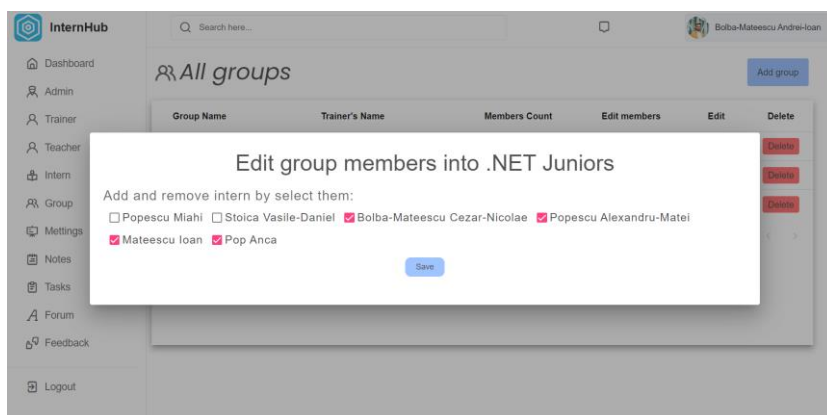


Figura 6.3.24 – Adaugare/editare studenti in/din grup

Pe lângă feature-urile comune în totalitate, mai avem feature-uri comune parțial, adică pentru 2 sau 3 tipuri de utilizatori. Aceste feature-uri sunt testele, challenge-urile și meeting-urile.

Pentru teste avem două părți. Una este partea de întrebări din care sunt create testele, iar cealaltă este partea de teste propriu-zise. Doar trainerii pot vedea toate întrebările și pot crea/edita sau șterge întrebări.

O întrebare este formată din textul întrebării și din răspunsurile posibile, care pot fi între 2 și 6. Astfel, un trainer poate adăuga o întrebare, fiind obligat să aibă cel puțin 2 răspunsuri. De asemenea, la acest pas, este obligat să ofere răspunsul corect, împreună cu numărul de puncte atribuite pentru această întrebare. Ecranul principal pentru acest pas este reprezentat în figura 4.3.25, iar ecranul de adăugare/editare este prezentat în figura 4.3.26.

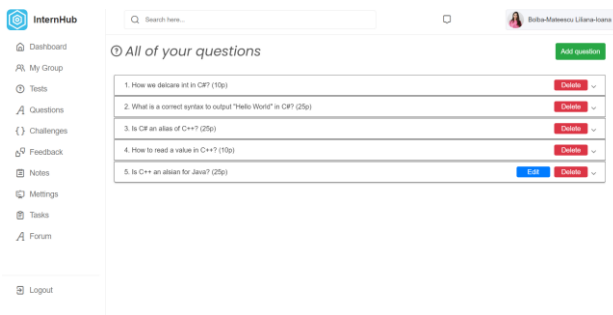


Figura 6.3.25 – Tabelul tuturor intrebărilor

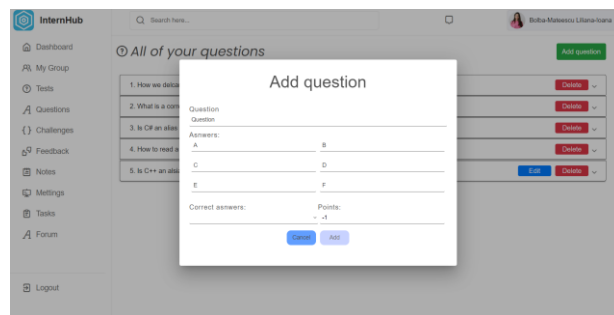


Figura 6.3.26 – Adaugare/editare întrebare

Având partea de întrebări finalizată, un test este compus din întrebări. Acesta are nevoie de un titlu, o limită de timp și de întrebări. Întrebările sunt selectate din cele existente. Apoi, următorul pas este publicarea testului și asignarea lui către persoane sau grupe. De aici, un intern îl va rezolva, iar ulterior un trainer poate vedea rezolvarea făcută de intern. Testul poate fi, de asemenea, editat sau șters, dar doar dacă nu a fost publicat încă. Astfel, nu există probleme referitoare la rezultate.

În figura 4.3.27 avem ecranul cu teste văzut de un trainer, cu teste în diferite stadii: create, publicate și teste pentru care s-a trecut de termenul limită, iar în figura 4.3.28 avem perspectiva internului.

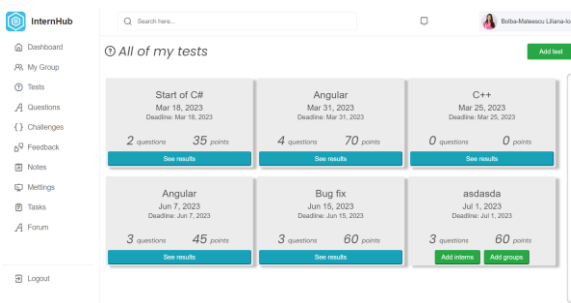


Figura 6.3.27 – Tabelul tuturor testelor din perspectiva unui trainer

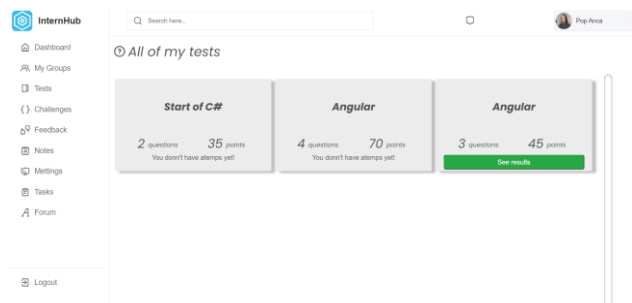


Figura 6.3.28 – Tabelul tuturor testelor din perspectiva unui intern

Asemenea celorlalte componente, partea de creare și editare a unui test se face prin intermediul unui modal, cu datele completate în cazul editării. Acest lucru este reprezentat în figura 4.3.29.

Figura 4.3.29 – Adaugarea/editarea unui test

Adăugarea întrebărilor se face prin apăsarea butonului "Add", iar întrebarea trece în partea de sus, având acum în componentă un buton cu textul "Remove". Pentru partea de publicare a testului, se apasă butonul "Publish", iar pentru asignare se deschide un modal din care se selectează persoanele sau grupurile pentru care este atribuit.

Pentru partea de trainer, acesta este turul pe care trebuie să-l facă în cadrul acestei funcționalități. Mai există partea de vizualizare a rezultatelor, dar aceasta necesită parcurgerea testelor de către interni.

Pentru a începe testul, studentul apasă pe butonul "Start test", iar îi apar întrebările la care trebuie să răspundă, conform figurii 4.3.30. Când selectează un răspuns, acesta primește direct și varianta corectă, care se face cu verde. Dacă răspunde greșit, îi apare varianta aleasă de el cu roșu, iar cea corectă cu verde.

Figura 6.3.31 – Intrebare din test

Figura 6.3.31 – Raspuns intrebare din test

După finalizarea testului, studentul poate să-și vadă rezultatul. Acest lucru poate fi văzut și de către trainer și de către profesor. Modul de vizualizare este asemănător cu cel de parcurgere a testului, diferența fiind faptul că se poate doar trece prin răspunsuri, fără posibilitatea de a răspunde la întrebare.

Pentru traineri și profesori, va apărea o listă a celor care au parcurs testul, putând să vadă evoluția fiecărui student.

Pentru partea de meeting-uri, lucrurile sunt destul de simple. Un trainer le poate crea și asigura către studenți sau către grupe, iar aceștia pot vedea participanții și link-ul către platforma aleasă pentru acest lucru.

Astfel, în figura 6.3.32 avem ecranul pentru meeting-uri din perspectiva unui trainer. Pentru un student, este la fel, având câteva elemente lipsă, precum butonul de "Add meeting" și link-urile de "Edit meeting" și "Delete meeting". Legat de partea de creare și editare, se abordează aceeași tactică ca până acum, cu un ecran comun pentru acestea, iar partea de editare are datele completate. Trainerul trebuie să specifice ziua ședinței, ora de început și o oră estimativă de final.

Tot din acest ecran se poate stabili și cine vede meeting-ul prin selectarea studenților sau a grupurilor direct.

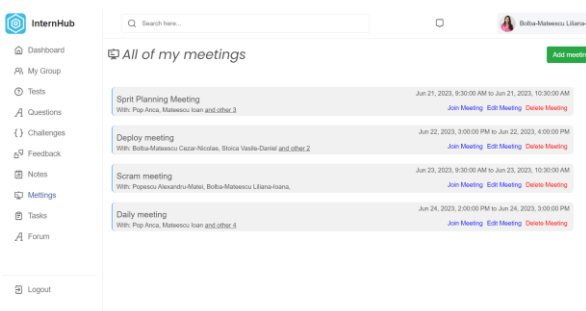


Figura 6.3.32 – Tabelul tuturor meeting-urilor

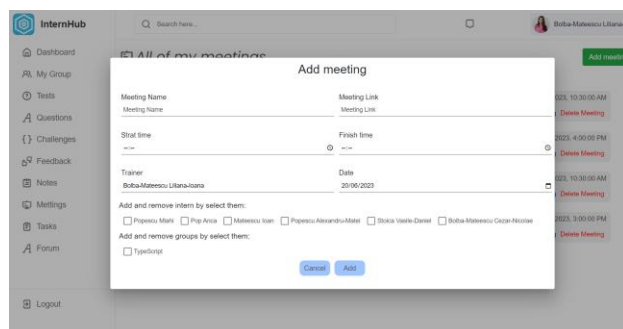


Figura 6.3.33 – Adaugare/editare meeting

Pe lângă partea de teste, mai este disponibilă și varianta de challenge-uri, care are și un sistem de gamificare în interiorul său. Un trainer adaugă un challenge pe zi, iar internii trebuie să le rezolve încărcând soluția printr-un fișier cu extensia .zip sau .rar. Challenge-ul are în componența sa și un număr maxim de puncte pe care un student le poate primi pentru acel challenge. Odată încărcată, trainer-ul trebuie să aprobe sau să respingă soluția propusă de student. În cazul respingerii, studentul mai are timp până la finalul zilei să încarce o altă variantă. În cazul aprobării, trainer-ul trebuie să atribuie un număr de puncte pentru acea soluție, care nu trebuie să depășească numărul maxim permis pentru acel challenge. Odată aprobată soluția, aceasta nu se mai poate modifica.

În figura 6.3.34 avem reprezentat ecranul cu toate challenge-urile disponibile. Viziunea internului este similară, neavând acces la operațiile de adăugare/editare sau aprobare/respingere soluție.

În figura 6.3.35 avem toate soluțiile disponibile pentru un task selectat. Fiecare soluție are trei stări în care poate să fie. Una este aceea în care trebuie acceptată sau respinsă, având două butoane pe care scrie "Approve", respectiv "Decline". Cea de-a doua variantă este cea pe care o regăsim în figura 6.3.35 și anume cea de respins, care apare cu roșu.

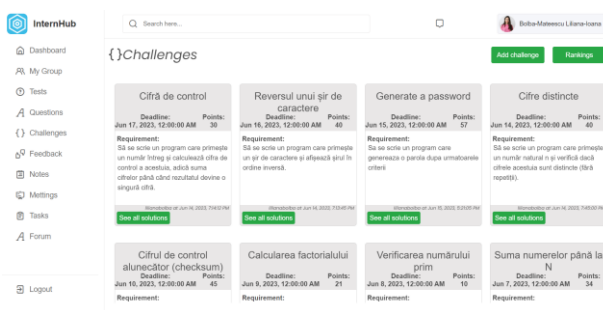


Figura 6.3.34 – Toate challenge-urile

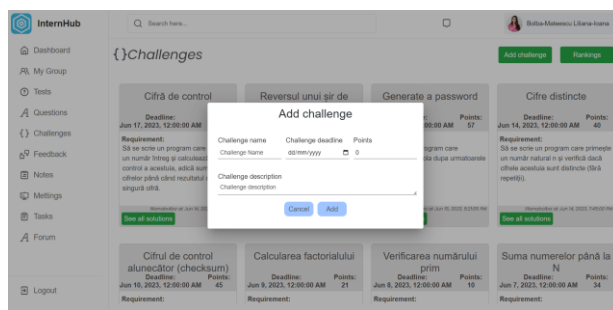


Figura 6.3.35 – Toate solutiile

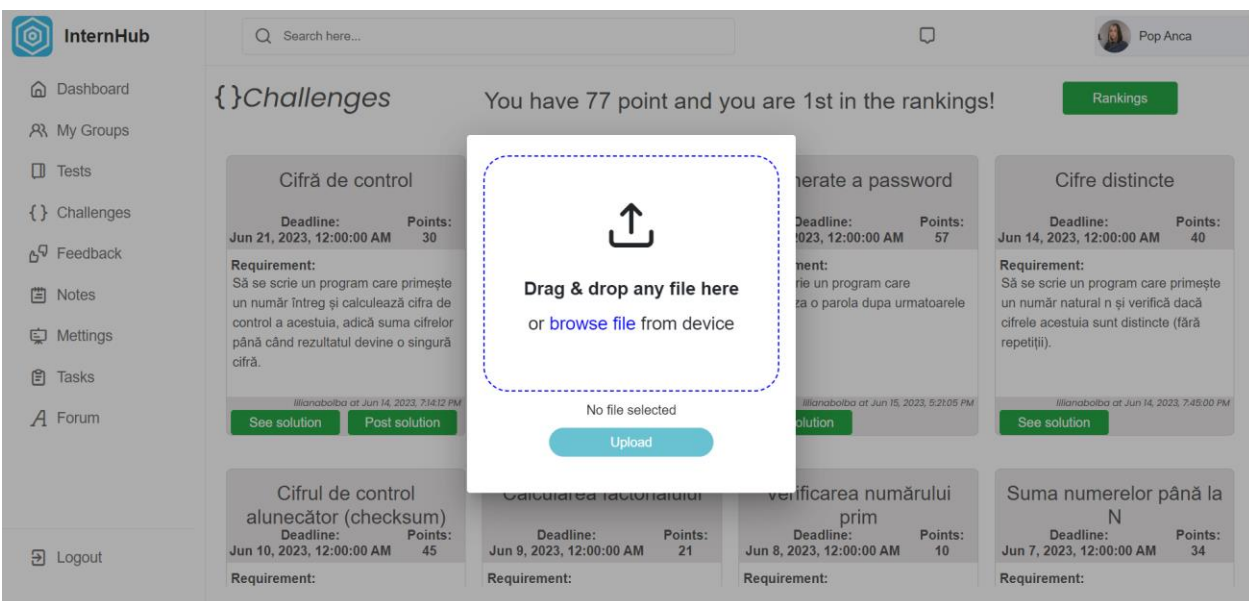


Figura 6.3.36 – Incararea solutiilor

Încărcarea soluțiilor se face prin intermediul fișierelor .zip sau .rar pentru a putea fi stocate mai ușor în baza de date. În plus, accesul la soluție se face prin apăsarea numelui de utilizator, prin intermediul căruia se descarcă soluția încărcată de către utilizator.

Sistemul de puncte este relevant deoarece, având un sistem de gamificare, putem crea un clasament pentru a vedea cei mai activi studenți. Acest lucru este vizibil atât de către profesori, studenți, cât și de către traineri. Sistemul de gamificare se bazează pe premiarea celor mai activi utilizatori prin premii mai mult sau mai puțin "valoroase", în funcție de fiecare companie.

Astfel, după cum se poate observa în figura 6.3.34, avem un buton numit "Ranking" care afișează clasamentul până în momentul actual. În figura 6.3.37 este reprezentat acest aspect.

Position	Intern	Points
1	Pop Anca	77
2	Bolba-Mateescu Cezar-Nicolae	40
3	Stoica Vasile-Daniel	0
4	Popescu Alexandru-Matei	0
5	Mateescu Ioan	0

Figura 6.3.37 – Sistemul de ranking

În componența sa, aplicația dispune și de un sistem de feedback pe care studenții îl pot implementa în munca lor pentru o eficiență ridicată. În plus și studenții la rândul lor oferă feedback trainerilor. Astfel toți pot implementa feedback-ul și se pot dezvolta.

Ecranul principal este comun pentru toți având primele feedback-urile pe care le-au oferit utilizatorii, iar apoi feedback-ul pe care l-au primit. Acest lucru este vizibil în figura 6.3.38.

Figura 6.3.38 – Sistemul de feedback

Când dorim să adăugăm un feedback, putem selecta dacă acesta este pentru un task, test sau challenge specific, prin intermediul dropdown-ului, sau feedback general, iar acest lucru se realizează prin neselectarea oricărui câmp de acest fel.

În plus, trebuie oferită și o notă care să nu fie mai mică de 1 sau mai mare de 10. Tot din acest câmp se selectează persoana căreia dorim să îi oferim feedback. Acest lucru este vizibil în figura 6.3.39.

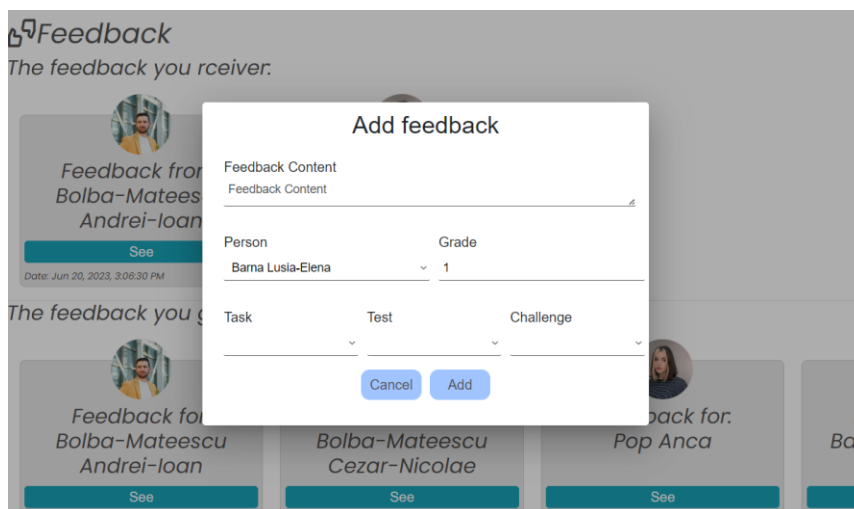


Figura 6.3.39 – Adaugare de feedback

Odată adăugat feedback-ul, acesta poate fi văzut de persoana căreia i-a fost oferit prin selectarea butonului "See". Astfel, se deschide o fereastră în care sunt prezentate toate detaliile acelui feedback. Acest lucru este ilustrat în figura 6.3.40 și figura 6.3.41.

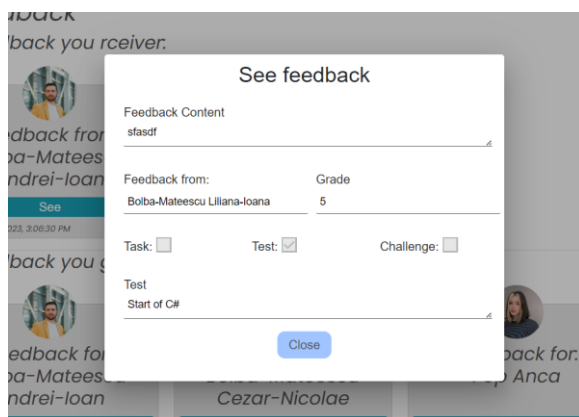


Figura 6.3.40 – Vizualizare feedback

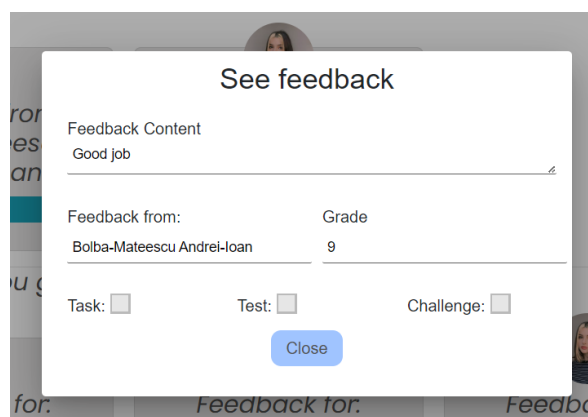


Figura 6.3.41 – Vizualizare feedback general

În figura 6.3.41 se poate observa că nu este selectat niciun camp, fiind un feedback general, iar în figura 6.3.42 este selectat campul de test, iar mai jos este afișat testul pentru care s-a oferit feedback.

Pe partea funcțională, sistemul de feedback funcționează exact ca celelalte elemente prezentate, singura diferență fiind faptul că, odată transmis feedback-ul, acesta nu mai poate fi modificat.

În cadrul aplicației, există și un sistem de task-uri care poate fi atribuit unui student sau unui grup de studenți. Un task este format din mai multe subtask-uri. În ecranul principal, prezentat în figura 6.3.42, utilizatorul poate vedea toate task-urile create de un trainer (în cazul în care utilizatorul este de tip trainer) sau task-urile atribuite persoanei, împreună cu task-urile atribuite grupurilor din care face parte. Pentru a vedea și subtask-urile, trebuie să apăsăm pe task, iar acestea vor apărea sub task-ul principal.

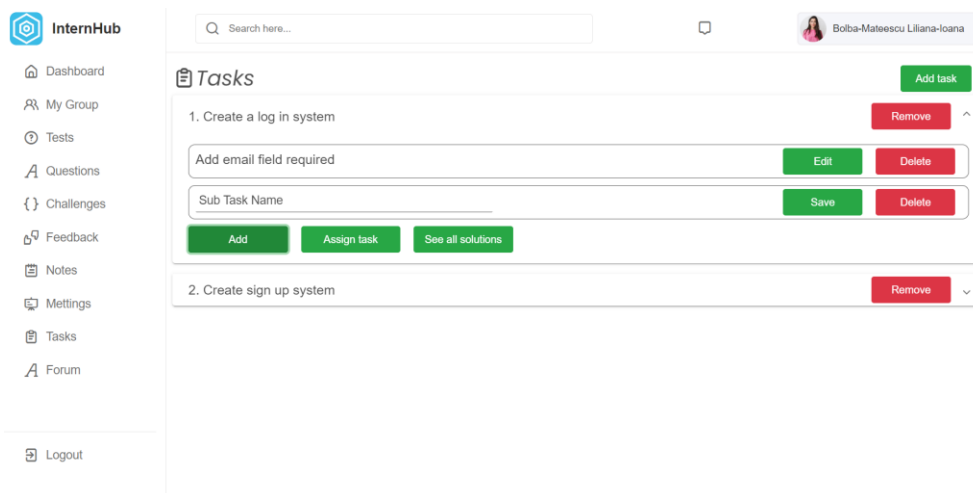


Figura 6.3.42 - Vizualizare task-uri

Un trainer poate adăuga, modifica sau șterge un task. Fiecare acțiune are propriul său buton. Astfel, pentru adăugare avem butonul "Add task", pentru modificare avem butonul "Edit task", iar pentru ștergere avem butonul "Delete". Pentru adăugarea sau modificarea unui task, folosim aceeași componentă ca până acum, având o componentă comună pentru aceste acțiuni. Această componentă poate fi observată în figura 6.3.43.

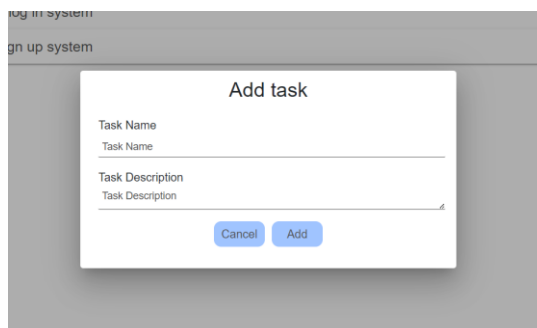


Figura 6.3.44 - Adaugare/ editare task

Pentru partea de subtask-uri, avem adăugare și editare inline. Practic, nu avem o componentă specială pentru acest lucru. Astfel, la apăsarea butonului "Add", se va adăuga un nou element în listă și de acolo se poate modifica. Salvarea elementului se va face la apăsarea butonului "Save". Ștergerea se face la apăsarea butonului "Remove".

Pentru partea de intern, acesta poate vedea task-urile care i-au fost atribuite. Ei se pot încărca pentru a vedea soluțiile adăugate. Acest lucru poate fi observat în figura 6.3.45. Partea de încărcare se folosește de aceeași componentă ca până acum pentru încărcarea de fișiere în baza de date.

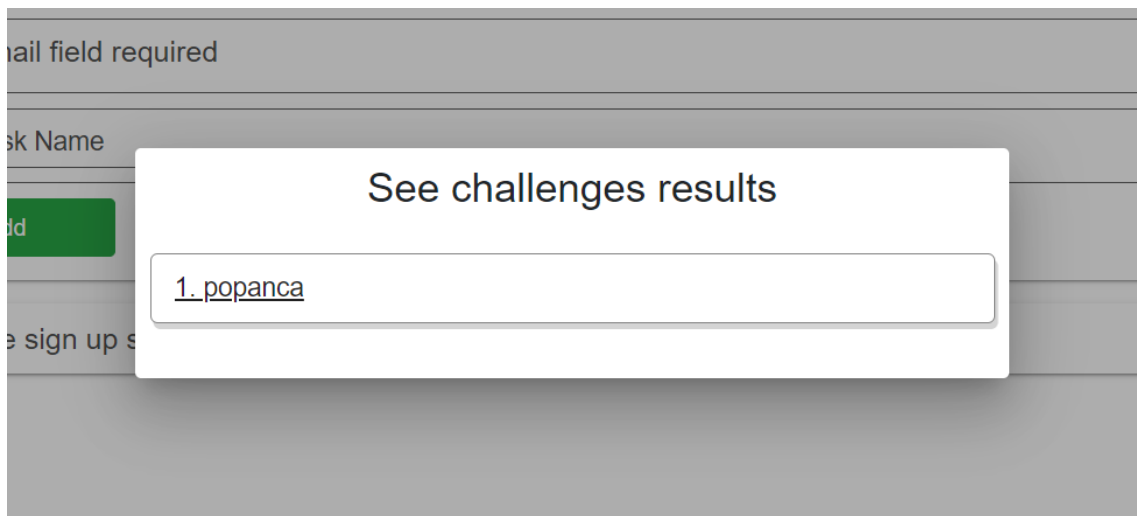


Figura 6.3.45 – Vizualizarea solutiilor

Pentru atribuire, ne vom folosi de componenta prezentată în imaginea 6.3.46, unde putem selecta atât grupuri, cât și studenți individuali.

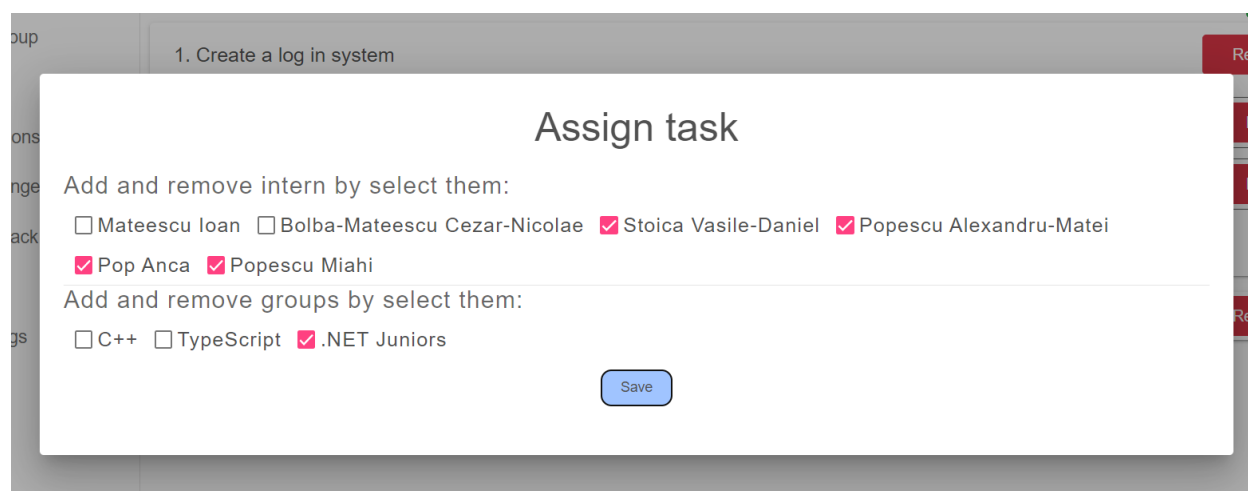


Figura 6.3.45 – Atribuirea de task-uri

Acesta este sistemul de task-uri, creat cu scopul de a monitoriza evoluția studenților și de a identifica punctele în care aceștia mai au nevoie de îmbunătățiri. Prin intermediul acestui sistem, se poate urmări progresul fiecărui student în rezolvarea task-urilor și se pot oferi feedback-uri și îndrumări pentru a-i sprijini în dezvoltarea lor. Acest instrument ajută la îmbunătățirea performanțelor și la obținerea unor rezultate mai bune în parcursul de învățare.

Tot în cadrul aplicației există un tab despre care nu am discutat în detaliu, denumit „My Groups”, care prezintă toate elementele grupului din care un student face parte, având împărțite pe categorii toate funcționalitățile care se bazează pe grup.

Aceasta a fost structura aplicației InternHub, care are ca scop dezvoltarea și monitorizarea studenților care fac stagiul de practică în cadrul unei companii.

7. Concluzii și perspective de viitor

7.1. Concluzii

Având în vedere faptul că aplicația InternHub nu este o aplicație deloc simplă, evident că au apărut provocări. Cea mai mare provocare a fost gestionarea pozelor, care presupun căutări lungi. Am ajuns la concluzia că este mult mai eficientă utilizarea unui serviciu de stocare în cloud pentru aceste informații. O altă provocare a fost stocarea încercărilor de rezolvare a challenge-urilor, deoarece acestea au fost stocate în baza de date ca fișiere .zip sau .rar și trebuiau descărcate pe computerul personal.

Fiind primul proiect de tip full-stack și a doua oară când creez o aplicație de front-end de la zero, prima fiind o aplicație de demo de la un curs de pe Udemy, pot spune că partea de front-end a reprezentat o reală provocare. Chiar dacă aveam cunoștințe de HTML/CSS și JavaScript, a fost o provocare să finalizez această aplicație pe partea de front-end. În schimb, pe partea de back-end a fost mult mai ușor, deoarece am mai creat API-uri, iar aici pot spune că singura provocare a fost legată de implementarea funcționalității real-time, în special pentru sistemul de chat. Crearea unui sistem de chat funcțional nu este în sine o acțiune foarte complexă, dar partea pe cât de complexă, pe atât de interesantă, este utilizarea bibliotecii SignalR pentru realizarea unui sistem de chat în timp real. Evident, provocarea a fost pentru întreaga implementare în timp real, dar totul a început de la sistemul de chat.

Astfel, aplicația a ajuns la forma ei actuală, având funcționalități care facilitează evoluția companiei în care va fi utilizată, respectând atât principiile de REST API, cât și principiile unei „Arhitecturi curate” (Clean Architecture).

7.2. Posibilități de extindere

Dezvoltarea unei aplicații nu se termină niciodată. Fie că vorbim de trecerea la o nouă versiune, fie că vorbim de un nou feature, o aplicație este în continuă dezvoltare și îmbunătățire. Aplicația InternHub nu face excepție de la această regulă. Putem lua în considerare următoarele îmbunătățiri:

- Trecerea la Angular 16, care a apărut în timpul dezvoltării aplicației.
- Trecerea la .NET 8 când va fi lansat, adică în noiembrie 2023.
- Crearea de teste unitare pentru creșterea calității și stabilității.
- Implementarea pattern-ului Repository în tandem cu un code clean-up pe partea de client.
- Implementarea standardului Microsoft OData pentru a îmbunătăți constrângerile REST.
- Securitatea aplicației, deoarece este destul de vulnerabilă în ciuda folosirii token-ului de autorizare.

Acestea sunt îmbunătățiri ce pot fi aduse pe viitor aplicației. Acum să vorbim puțin despre cum mai poate fi dezvoltată aplicația prin adăugarea unor module suplimentare:

- Implementarea unui sistem de tip two-way authentication.

- Un sistem de notificări pentru fiecare eveniment relevant în cadrul aplicației.
- Un sistem de logging pentru o urmărire mai bună a codului și pentru o identificare mai ușoară a erorilor.
- Implementarea unui sistem de evenimente gestionate prin calendar pentru utilizatorii aplicației.
- Introducerea unui modul de rapoarte pentru a putea vedea mai bine evoluția fiecărui intern.
- Afișarea clasamentului pe perioade de timp, pentru a avea o varietate de premii.

8. Bibliografie

- [1] What's new in .NET 7, <https://github.com/dotnet/docs/blob/main/docs/core/whats-new/dotnet-7.md>
- [2] .NET, <https://en.wikipedia.org/wiki/.NET>
- [3] Global market share held by operating systems for desktop PCs, from January 2013 to January 2023, <https://www.statista.com/statistics/218089/global-market-share-of-windows-7/>
- [4] .NET and .NET Core Support Policy, <https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core>
- [5] C Sharp (programming language), [https://en.wikipedia.org/wiki/C_Sharp_\(programming_language\)](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))
- [6] C# in 2023: The MOST POPULAR Programming Language?, <https://www.bytehide.com/blog/c-wants-to-become-the-most-popular-programming-language-in-2022>
- [7] 2022 developer survey, <https://survey.stackoverflow.co/2022/#most-popular-technologies-webframe>
- [8] Introduction to SignalR, <https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>
- [9] What is the Angular Framework?, <https://www.blog.globalteams.ltd/what-is-the-angular-framework-new-features-and-updates-2023>
- [10] Diferențele dintre Javascript și Typescript, <https://hitechglitz.com/romania/diferentele-dintre-javascript-si-typescript/>
- [11] The top programming languages, <https://octoverse.github.com/2022/top-programming-languages>
- [12] Microsoft SQL Server, https://ro.wikipedia.org/wiki/Microsoft_SQL_Server
- [13] What's new in SQL Server 2022, <https://learn.microsoft.com/ro-ro/sql/sql-server/what-s-new-in-sql-server-2022?view=sql-server-ver16>

- [14] SQL Server 2022, <https://www.microsoft.com/en-us/sql-server/sql-server-2022>
- [15] 2022 developer survey, <https://survey.stackoverflow.co/2022/#most-popular-technologies-database-learn>
- [16] BAZE DATE Normalizarea, <https://www.studocu.com/ro/document/universitatea-petrol-gaze-din-ploiesti/baze-de-date/baze-date-normalizarea/7514728>
- [17] Modelarea relațională a entităților în bazele de date, <https://www.telework.ro/ro/modelarea-relationala-a-entitatilor-in-bazele-de-date/>
- [18] Bearer Authentication, <https://swagger.io/docs/specification/authentication/bearer-authentication/>
- [19] JSON Web Token, https://en.wikipedia.org/wiki/JSON_Web_Token
- [20] Robert C.Martin, Clean Architecture: A Craftsman's Guide to Software Stucture and design, Editura Pearson, USA, 2007
- [21] Repository Design Pattern in C#, <https://dotnettutorials.net/lesson/repository-design-pattern-csharp/>
- [22] The Clean Architecture, <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>
- [23] O introducere în arhitectura REST, <https://www.opti.ro/ro/post/about-rest-apis>
- [24] RESTful Web Services, <https://www.oracle.com/technical-resources/articles/javase/restful.html>
- [25] HMACSHA512 Class, <https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.hmacsha512?view=net-7.0>
- [26] .NET SDK, https://cloudinary.com/documentation/dotnet_integration