**SISTEM DE GESTIUNE A CONVENȚIILOR DE PRACTICĂ**

**Candidat:Jurca Daniel**

**Coordonator științific: Conf. dr.ing. Ciprian-Bogdan CHIRILĂ**

Sesiunea: Iulie 2025

**REZUMAT**

Această aplicație reprezintă un sistem informatic pentru gestionarea convențiilor de practică studențească în cadrul Facultății de Automatică și Calculatoare din cadrul Universității Politehnica Timișoara. Sistemul oferă o platformă digitală centralizată prin care studenți, tutori, prodecani și prorectori își pot gestiona întregul proces de creare, gestionare, aprobare, semnare și arhivare a convențiilor de practică. Platforma facilitează întregul flux de documente necesar pentru desfășurarea stagiilor de practică, de la inițierea convenției de către student până la aprobarea finală de către autoritățile academice competente.

Sistemul automatizează fluxul de lucru, reduce semnificativ timpul de procesare și elimină potențialele erori umane din procesul de gestiune a convențiilor de practică. Aplicația implementează un sistem robust de autentificare și autorizare bazat pe roluri, asigurând că fiecare utilizator are acces doar la funcționalitățile și datele relevante pentru rolul său în cadrul procesului. Platforma oferă funcționalități precum generarea automată a documentelor în multiple formate (HTML, PDF, Word), sistem de notificări pentru actualizări de status și un tablou de bord personalizat pentru fiecare tip de utilizator.

Implementarea folosește tehnologii moderne precum Spring Boot pentru partea de backend și Thymeleaf și Bootstrap pentru partea de frontend, oferind o soluție eficientă și scalabilă.

**ABSTRACT**

This application represents an information system for managing student internship agreements within the Faculty of Automation and Computers at the Politehnica University of Timișoara. The system provides a centralized digital platform through which students, tutors, vice-deans, and vice-rectors can manage the entire process of creating, managing, approving, signing, and archiving internship agreements. The platform facilitates the complete document workflow necessary for conducting internships, from the initial creation of the agreement by the student to the final approval by the competent academic authorities. The system automates the workflow, significantly reduces processing time, and eliminates potential human errors from the internship agreement management process. The application implements a robust role-based authentication and authorization system, ensuring that each user has access only to the functionalities and data relevant to their role in the process. The platform provides features such as automatic document generation in multiple formats (HTML, PDF, Word), a status update notification system, and a customized dashboard for each type of user. The implementation utilizes modern technologies such as Spring Boot for the backend and Thymeleaf and Bootstrap for the frontend, providing an efficient and scalable solution.

**1.Introducere. Punerea problemei**

Gestionarea documentelor reprezintă un proces esențial în cadrul oricărei organizații, iar în cazul unei instituții de învățământ superior, devine o componentă critică în administrarea situației academice a studenților și în dezvoltarea lor profesională viitoare. În contextul digitalizării accelerate și al necesității de eficientizare a proceselor administrative la Facultatea de Automatică și Calculatoare din cadrul Universității Politehnica Timișoara, am identificat o nevoie stringentă pentru implementarea unui sistem informatic modern dedicat acestor procese, cu un focus specific pe gestiunea convențiilor de practică.

* 1. **Contextul și Justificarea Proiectului**

Sistemul actual de gestionare a convențiilor de practică, predominant bazat pe documente fizice și procese manuale, se confruntă cu numeroase limitări și provocări semnificative. Printre acestea, se numără timpul îndelungat necesar procesării documentelor, un risc ridicat de erori umane în completarea și manipularea acestora, dificultăți în urmărirea statusului convențiilor în timp real, precum și un consum considerabil de resurse umane și materiale. Mai mult, absența unei platforme digitale centralizate îngreunează monitorizarea eficientă a stagiilor de practică și generarea de rapoarte agregate la nivel instituțional. Aceste deficiențe subliniază importanța și urgența dezvoltării unei soluții informatice moderne.

Proiectul de față, intitulat "SISTEM DE GESTIUNE A CONVENȚIILOR DE PRACTICĂ", își propune să răspundă acestor provocări prin dezvoltarea unei platforme web centralizate. Această platformă va digitaliza și optimiza întregul flux de management al convențiilor de practică studențească. Aplicația se adresează în mod specific Facultății de Automatică și Calculatoare din cadrul Universității Politehnica Timișoara, dar este concepută cu un grad de flexibilitate care permite adaptarea și pentru alte structuri academice. Scopul principal este de a transforma modul tradițional de lucru, adesea greoi și ineficient, într-un proces digitalizat, transparent și eficient pentru toți utilizatorii implicați: studenți, tutori de practică din companii, cadre didactice universitare, personal administrativ și personalul de conducere al facultății și universității.

* 1. **Obiectivele Proiectului**

Dezvoltarea acestui sistem informatic vizează atingerea următoarelor obiective principale:

Digitalizarea completă a procesului de creare și gestionare a convențiilor de practică, eliminând dependența de documentele fizice și reducând considerabil timpul de procesare

Implementarea unui flux de lucru automatizat pentru inițierea, aprobarea, semnarea și arhivarea convențiilor, integrând un sistem de notificări pentru a informa toate părțile implicate despre stadiul procesului.

Asigurarea unui mediu securizat pentru gestionarea datelor personale și a informațiilor specifice convențiilor, respectând normele moderne de protecție a datelor și cerințele de securitate informatică. Acest aspect include un sistem robust de autentificare și autorizare bazat pe roluri.

Reducerea semnificativă a timpului de procesare a documentelor prin automatizarea sarcinilor repetitive și eliminarea pașilor redundanți.

Oferirea de funcționalități avansate, precum generarea automată a documentelor în multiple formate (HTML, PDF, Word) și un tablou de bord personalizat pentru fiecare tip de utilizator.

Implementarea unui sistem de raportare și analiză care să permită monitorizarea în timp real a statusului convențiilor și generarea de rapoarte statistice comprehensive.

Asigurarea scalabilității și adaptabilității sistemului pentru a permite dezvoltări și îmbunătățiri viitoare

Prin atingerea acestor obiective, sistemul propus nu doar că va moderniza un proces administrativ esențial, ci va contribui și la creșterea eficienței, reducerea erorilor și îmbunătățirea experienței tuturor utilizatorilor implicați în procesul de practică studențească.

* 1. **Structura Lucrării**

Prezenta lucrare este structurată după cum urmează:

* **Capitolul 1 (Introducere):** Prezintă contextul, justificarea și obiectivele proiectului, alături de structura generală a lucrării.
* **Capitolul 2 (Tehnologii Utilizate):** Descrie arhitecturile software și tehnologiile web care stau la baza dezvoltării sistemului informatic.
* **Capitolul 3 (Analiza Cerințelor și Specificațiile Proiectului):** Detaliază cerințele funcționale și non-funcționale ale sistemului, precum și actorii implicați și rolurile acestora.
* **Capitolul 4 (Proiectarea Sistemului):** Prezintă arhitectura generală a aplicației, modelul conceptual al bazei de date și designul interfețelor utilizator.
* **Capitolul 5 (Implementarea Sistemului):** Descrie implementarea efectivă a modulelor sistemului, detaliind clasele, metodele și structurile de date utilizate.
* **Capitolul 6 (Testarea și Instalarea Sistemului):** Expune strategiile de testare aplicate pentru validarea funcționalităților și corectitudinii sistemului, precum și pașii necesari pentru instalarea și configurarea aplicației.
* **Capitolul 7 (Concluzii și Dezvoltări Viitoare):** Sintetizează rezultatele obținute, evidențiază contribuțiile personale și propune direcții pentru dezvoltări ulterioare ale sistemului.
* **Bibliografia:** Listează sursele documentare utilizate în elaborarea lucrării.

1. **Tehnologii web și embedded utilizate**

Acest capitol detaliază platformele, framework-urile, bibliotecile și limbajele de programare care au constituit fundamentul tehnic al Sistemului de Gestiune a Convențiilor de Practică. Selecția acestor tehnologii a fost realizată în urma unei analize atente, având ca scop principal dezvoltarea unei aplicații web performante, securizate, scalabile și ușor de întreținut, care să răspundă eficient tuturor cerințelor funcționale și non-funcționale identificate în etapa de analiză. S-a optat pentru o combinație de tehnologii recunoscute pentru robustețea și flexibilitatea lor în dezvoltarea aplicațiilor web moderne, facilitând totodată un proces de dezvoltare agil și eficient.

**2.1. Spring Framework și Spring Boot**

Spring Framework este o platformă Java open-source complexă și versatilă, care furnizează un suport infrastructurál extensiv pentru dezvoltarea aplicațiilor Java, în special a celor de tip enterprise. Prin intermediul modulelor sale multiple, Spring simplifică dezvoltarea, promovând bune practici precum programarea orientată pe aspecte (AOP) și inversarea controlului (IoC).

**Inversion of Control (IoC) și Dependency Injection (DI):** Nucleul Spring Framework este containerul IoC, responsabil pentru gestionarea ciclului de viață al obiectelor (denumite "beans") și a dependențelor dintre acestea. În loc ca obiectele să își creeze propriile dependențe, acestea sunt "injectate" de către container, ceea ce duce la o cuplare slabă între componente și la o mai mare modularitate a aplicației. În cadrul sistemului de gestiune a convențiilor, acest principiu a permis o separare clară a responsabilităților între diferitele servicii (de exemplu, ConventionService, UserService) și repository-uri, facilitând testarea și mentenanța.

**Arhitectura Modulară:** Spring este constituit din aproximativ 20 de module, grupate funcțional (Core Container, Data Access/Integration, Web, AOP, Instrumentation, Test), permițând dezvoltatorilor să utilizeze doar componentele necesare proiectului. Pentru acest proiect, modulele Spring Web MVC, Spring Data JPA și Spring Security au fost esențiale.

A diagram of software

Description automatically generated with medium confidence

Figura 1 - Arhitectura Spring Framework Runtime

**Spring Boot:** Pentru a accelera și simplifica procesul de dezvoltare și configurare, s-a utilizat Spring Boot. Acesta oferă un set de opinii predefinite pentru configurarea aplicațiilor Spring, auto-configurare, servere web embedded (cum ar fi Tomcat, utilizat implicit) și un sistem robust de gestionare a dependențelor. Utilizarea Spring Boot a redus semnificativ efortul de configurare inițială și a permis concentrarea pe logica de business a aplicației, contribuind la realizarea unei soluții eficiente și scalabile.

Printre avantajele concrete aduse de Spring Framework în acest proiect se numără:

* **Simplificarea accesului la date:** Prin Spring Data JPA, interacțiunea cu baza de date MySQL a fost abstractizată, permițând definirea facilă a repository-urilor și a operațiilor CRUD.
* **Dezvoltare web robustă:** Modulul Spring Web MVC a furnizat o arhitectură solidă pentru implementarea controller-elor care gestionează cererile HTTP și interacționează cu logica de business.
* **Securitate integrată:** Prin Spring Security, s-a implementat un sistem de autentificare și autorizare granular, esențial pentru a proteja datele și funcționalitățile specifice fiecărui rol de utilizator (student, tutore, prodecan etc.).

**2.2. Thymeleaf**

Thymeleaf este un motor modern de template-uri Java, orientat către server, capabil să proceseze și să genereze HTML, XML, JavaScript, CSS și text simplu. Caracteristica sa distinctivă este conceptul de "Natural Templating", care permite șabloanelor Thymeleaf să fie fișiere HTML valide, care pot fi vizualizate corect în browsere ca prototipuri statice, dar care sunt transformate în pagini dinamice la runtime. Această dualitate facilitează colaborarea între designeri și dezvoltatori.

În cadrul sistemului de gestiune a convențiilor de practică, Thymeleaf a fost utilizat pentru a construi interfața utilizator dinamică și interactivă. Funcționalitățile cheie exploatate includ:

* **Integrarea dinamică a datelor:** Afișarea informațiilor specifice convențiilor (datele studentului, ale companiei, statusul convenției etc.) în paginile HTML, prin legarea modelului de date trimis de controllerele Spring la atributele Thymeleaf.
* **Structuri de control:** Implementarea logicii condiționale (ex: afișarea anumitor butoane sau secțiuni doar pentru anumite roluri de utilizatori sau în funcție de statusul convenției) și iterative (ex: listarea convențiilor într-un tabel) direct în șabloanele HTML.
* **Procesarea formularelor:** Facilitarea legării datelor din formularele HTML (ex: formularul de creare sau editare a unei convenții) la obiectele Java din backend.
* **Internaționalizare și localizare:** Deși nu a fost un focus principal în versiunea curentă, Thymeleaf oferă suport nativ pentru aceste aspecte, ceea ce reprezintă un avantaj pentru dezvoltările viitoare ale aplicației.
* **Integrarea cu Spring Security:** Utilizarea atributelor specifice Thymeleaf pentru a controla accesul la diferite părți ale interfeței utilizator în funcție de rolurile și permisiunile utilizatorului autentificat.

**Spring Security**

În contextul sistemelor informaționale contemporane, securitatea reprezintă un aspect fundamental care necesită o abordare metodică și comprehensivă. Spring Security, implementat în cadrul prezentului sistem, constituie soluția tehnologică predominantă pentru securizarea aplicațiilor Java, oferind un ecosistem extensiv pentru autentificare, autorizare și protecție împotriva vulnerabilităților comune.

Arhitectura de securitate implementată prin Spring Security în cadrul sistemului de management al convențiilor de practică cuprinde următoarele componente esențiale:

* Mecanisme robuste de autentificare, configurabile pentru diverse strategii de verificare a identității utilizatorilor, inclusiv autentificare multifactor și integrare cu sisteme externe;
* Sisteme de autorizare granulară bazate pe roluri și permisiuni, asigurând astfel că utilizatorii pot accesa exclusiv funcționalitățile și datele corespunzătoare nivelului lor de autorizare;
* Protecție implementată la nivel de aplicație împotriva vulnerabilităților de securitate comune, cum ar fi Cross-Site Request Forgery (CSRF), Cross-Site Scripting (XSS) și tentativele de fixare a sesiunii;
* Implementarea algoritmilor criptografici avansați pentru stocarea parolelor utilizând funcții de hashing unidirecționale cu adăugare de salt, prevenind astfel compromiterea credențialelor în cazul unei potențiale breșe de securitate.

**2.3. Spring Security**

Securitatea este un pilon fundamental al oricărei aplicații web, cu atât mai mult într-un sistem care gestionează date personale și documente oficiale. Spring Security este un framework puternic și extrem de configurabil care oferă capabilități complete de autentificare, autorizare și protecție împotriva diverselor atacuri pentru aplicațiile Java.

Pentru sistemul de gestiune a convențiilor de practică, Spring Security a fost esențial pentru a asigura:

* **Autentificare Robustă:** Implementarea unui mecanism de login securizat, unde utilizatorii (studenți, tutori, prodecani, prorectori) se autentifică folosind un identificator unic (email) și o parolă. Parolele sunt stocate în baza de date într-un format criptat (hashed) pentru a preveni accesul neautorizat la acestea în cazul unei breșe de securitate.
* **Autorizare Granulară Bazată pe Roluri:** Definirea și aplicarea strictă a permisiunilor pentru fiecare rol de utilizator. De exemplu, un student poate crea și edita propriile convenții, în timp ce un prodecan are drepturi de aprobare și de management al altor utilizatori. Acest lucru asigură că fiecare utilizator accesează doar funcționalitățile și datele relevante pentru activitatea sa în cadrul platformei.
* **Protecție Împotriva Atacurilor Comune:** Spring Security oferă protecție implicită împotriva unor vulnerabilități web frecvente, cum ar fi Cross-Site Request Forgery (CSRF) – prevenind acțiuni neautorizate executate în numele utilizatorului, Cross-Site Scripting (XSS) – prin mecanisme de sanitizare a input-ului, și Session Fixation.
* **Gestionarea Sesiunilor:** Managementul securizat al sesiunilor utilizatorilor.
* **Criptarea Parolelor:** Utilizarea funcțiilor de hashing unidirecționale, cu adăugarea unui "salt" (o valoare aleatorie unică pentru fiecare parolă), pentru stocarea securizată a parolelor, făcând extrem de dificilă recuperarea acestora chiar și în cazul compromiterii bazei de date.

**2.4. Tehnologii Frontend (HTML5, CSS3, JavaScript, Bootstrap, DataTables)**

Interfața cu utilizatorul (frontend-ul) sistemului a fost dezvoltată utilizând o combinație de tehnologii standard și framework-uri moderne pentru a asigura o experiență interactivă, intuitivă, accesibilă și adaptabilă la diverse dispozitive (responsive design).

**HTML5 (HyperText Markup Language 5):** Reprezintă limbajul de marcare standard pentru crearea structurii paginilor web. Utilizarea elementelor semantice noi introduse în HTML5, precum <header>, <nav>, <main>, <article>, <section> și <footer>, a contribuit la o mai bună structurare logică a conținutului paginilor, îmbunătățind atât lizibilitatea codului pentru dezvoltatori.

**CSS3 (Cascading Style Sheets 3):** A fost utilizat pentru stilizarea elementelor HTML, definind aspectul vizual al aplicației – culori, fonturi, layout, animații etc.. S-au folosit tehnici moderne de layout, precum Flexbox și Grid, pentru a crea aranjamente complexe și responsive ale elementelor pe pagină.

**Bootstrap 4.6:** Este un framework frontend popular, open-source, care oferă o colecție vastă de componente UI reutilizabile (meniuri de navigație, formulare, butoane, alerte, modale etc.), un sistem de grid responsiv și utilitare CSS. Integrarea Bootstrap a accelerat procesul de dezvoltare a interfeței, asigurând un design consistent, modern și adaptabil pe toate dispozitivele, fără a necesita scrierea unui volum mare de cod CSS custom.

**JavaScript și jQuery:** JavaScript, împreună cu biblioteca jQuery, a fost utilizat pentru a adăuga interactivitate dinamică la nivel de client. Aceasta include:

* Validarea formularelor direct în browser înainte de trimiterea datelor către server, oferind feedback imediat utilizatorului.
* Manipularea Document Object Model (DOM) pentru a actualiza dinamic conținutul paginilor fără a necesita reîncărcarea completă (de exemplu, afișarea/ascunderea unor elemente, actualizarea unor liste).
* Realizarea de cereri asincrone către server (AJAX) pentru a încărca date sau a efectua acțiuni în fundal, îmbunătățind fluiditatea experienței utilizatorului.

**DataTables:** Este un plug-in jQuery avansat care transformă tabelele HTML standard în controale interactive și puternice. În cadrul aplicației, DataTables a fost folosit pentru a afișa listele de convenții, studenți sau alte entități, oferind funcționalități precum sortare pe multiple coloane, căutare/filtrare avansată, paginare și exportul datelor în formate precum CSV sau PDF, facilitând astfel analiza și managementul unui volum mare de informații.

**2.5. Sistemul de Gestiune a Bazelor de Date (MySQL și phpMyAdmin)**

Alegerea unui sistem de gestiune a bazelor de date (SGBD) adecvat este crucială pentru performanța, integritatea și scalabilitatea oricărei aplicații care manipulează date.

**MySQL:** Pentru stocarea și gestionarea datelor sistemului de convenții de practică, s-a optat pentru MySQL, unul dintre cele mai populare SGBD-uri relaționale (RDBMS) open-source. Motivele alegerii MySQL includ:

* + **Robustețe și Fiabilitate:** MySQL este recunoscut pentru stabilitatea și capacitatea sa de a gestiona volume mari de date și tranzacții concurente.
  + **Performanță:** Oferă mecanisme eficiente de indexare și optimizare a interogărilor, esențiale pentru un timp de răspuns rapid al aplicației.
  + **Model Relațional:** Permite structurarea logică și eficientă a datelor referitoare la utilizatori, convenții, companii, tutori și alte entități ale sistemului, definind relații clare între acestea.
  + **Integritate Referențială:** Suportă constrângeri de cheie străină, asigurând consistența datelor și prevenind anomaliile care ar putea apărea în urma operațiilor de modificare sau ștergere.
  + **Securitate Granulară:** Permite definirea precisă a drepturilor de acces la nivel de utilizator, tabelă sau chiar coloană individuală, contribuind la securitatea datelor.
  + **Scalabilitate:** MySQL poate fi scalat pentru a gestiona un număr mare de utilizatori și un volum crescut de date.
  + **Comunitate și Suport:** Beneficiază de o comunitate mare de dezvoltatori și de o documentație vastă.

**phpMyAdmin:** Pentru administrarea și mentenanța bazei de date MySQL, s-a utilizat phpMyAdmin, o interfață web grafică populară și intuitivă. Aceasta simplifică sarcinile administrative, eliminând necesitatea de a scrie comenzi SQL complexe pentru operațiuni comune.

Principalele funcționalități oferite de phpMyAdmin și utilizate în cadrul proiectului sunt:

* + **Crearea și Modificarea Structurii Bazei de Date:** Definirea tabelelor, a coloanelor, a tipurilor de date, a indecșilor și a relațiilor referențiale într-un mod vizual.
  + **Manipularea Datelor:** Interfețe pentru inserarea, vizualizarea, editarea și ștergerea înregistrărilor din tabele.
  + **Executarea Interogărilor SQL:** O consolă SQL pentru executarea directă a interogărilor complexe care nu sunt acoperite de interfața grafică.
  + **Importul și Exportul Datelor:** Facilități pentru importul datelor (de exemplu, dintr-un fișier CSV sau SQL) și exportul bazei de date sau a tabelelor individuale în diverse formate (SQL, CSV, XML, PDF), esențiale pentru backup și migrare.
  + **Administrarea Utilizatorilor și a Privilegiilor:** Gestionarea conturilor de utilizatori MySQL și a drepturilor de acces ale acestora la baza de date.

**3. Analiza cerințelor și specificațiile funcționale ale sistemului informațional**

Acest capitol definește cu precizie scopul și funcționalitățile implementate în cadrul Sistemului de Gestiune a Convențiilor de Practică. Sunt descrise capabilitățile sistemului și modul în care acesta răspunde nevoilor identificate, precum și rolurile utilizatorilor și interacțiunile acestora cu platforma.

**3.1. Obiectivul Principal al Proiectului**

Obiectivul primordial care a stat la baza inițierii și dezvoltării acestui proiect software este crearea și implementarea unei platforme web centralizate, moderne și intuitive, destinate digitalizării complete și optimizării procesului complex de management al convențiilor de practică studențească. Aplicația se adresează în mod specific nevoilor Facultății de Automatică și Calculatoare din cadrul Universității Politehnica Timișoara, având însă potențialul de a fi adaptată și pentru alte structuri academice similare.

Sistemul revoluționează modul tradițional de gestionare a acestor acorduri, adesea bazat pe documente fizice și multiple intervenții manuale, care este consumator de timp și predispus la erori. Prin centralizarea informațiilor și automatizarea fluxurilor de lucru, se realizează o eficientizare semnificativă a interacțiunilor dintre toți actorii implicați în procesul de practică: studenți, tutori de practică desemnați de companiile partenere, cadre didactice universitare care asigură supervizarea academică, personalul administrativ responsabil cu evidența documentelor și, nu în ultimul rând, personalul de conducere al facultății și universității implicat în etapele de aprobare finală.

Platforma digitală oferă un mediu transparent, accesibil și ușor de utilizat, având ca scop reducerea birocrației, minimizarea erorilor umane și asigurarea unui flux informațional rapid, coerent și securizat între toate părțile implicate.

**3.2. Actorii Sistemului (Roluri de Utilizatori)**

Pentru a modela corect interacțiunile și permisiunile în cadrul sistemului, au fost identificate și definite următoarele roluri distincte de utilizatori. Fiecare rol dispune de un set specific de responsabilități, drepturi de acces la funcționalități și date, precum și interfețe personalizate, adaptate nevoilor sale operaționale.

**Student (ROLE\_STUDENT):** Beneficiarul direct al procesului de practică.

**Responsabilități:** Inițiază procesul de creare a unei noi convenții de practică, completând cu atenție toate câmpurile necesare referitoare la perioada stagiului, compania parteneră selectată, tutorele desemnat de companie și cadrul didactic supervizor din partea facultății. Are responsabilitatea de a încărca documentele justificative solicitate (dacă este cazul) și imaginea propriei semnături olografe. Monitorizează constant statusul convenției sale în fluxul de aprobare și poate edita convenția în anumite etape (de exemplu, înainte de prima aprobare sau dacă aceasta a fost returnată pentru corecturi).

**Tutore de Practică (ROLE\_TUTORE):** Reprezentantul calificat al companiei unde studentul își va desfășura stagiul, având rolul de îndrumător direct (trainer/mentor) al studentului pe parcursul activităților practice.

**Responsabilități:** Primește spre analiză convențiile studenților pe care compania i-a acceptat pentru supervizare din punct de vedere al conținutului practic al stagiului. Are datoria de a revizui detaliile convenției, de a verifica corectitudinea datelor referitoare la sarcinile și obiectivele practice ale studentului. Poate aproba sau respinge convenția din această perspectivă, oferind un motiv clar în cazul unei respingeri. Contribuie la definirea planului de practică al studentului și monitorizează activitatea acestuia. Poate încărca propria semnătură dacă este desemnat ca semnatar pe document din partea operațională a companiei, distinct de reprezentantul legal.

**Partener de Practică (ROLE\_PARTENER):** Reprezentantul legal sau persoana desemnată cu autoritate de semnătură din partea entității externe (companie, instituție) unde se desfășoară stagiul. Acest rol este responsabil pentru angajamentul formal al companiei în convenție.

**Responsabilități:** Este implicat în procesul de revizuire finală și semnare a convențiilor din perspectiva legală și contractuală a companiei. Încarcă imaginea semnăturii reprezentantului legal al companiei, confirmând acordul oficial al entității partenere. Poate gestiona aspecte administrative și contractuale generale legate de parteneriatele de practică ale companiei cu universitatea.

**Cadru Didactic Supervizor (ROLE\_CADRU\_DIDACTIC):** Membru al personalului didactic al facultății, responsabil pentru monitorizarea academică a stagiului de practică.

**Responsabilități:** Revizuiește conținutul convenției din perspectiva academică, asigurându-se că obiectivele stagiului sunt aliniate cu programul de studii și cerințele educaționale. Aprobă sau respinge convenția din punct de vedere academic și încarcă imaginea semnăturii sale, confirmând supervizarea.

**Prodecan (ROLE\_PRODECAN):** Membru al conducerii executive a facultății, cu responsabilități în validarea finală a convențiilor la nivel de facultate.

**Responsabilități:** Intervine în fluxul de aprobare după ce convenția a primit avizele prealabile (tutore, cadru didactic, partener de practică). Verifică conformitatea cu regulamentele facultății și universității și ia decizia finală de aprobare sau respingere la nivelul său de competență. Deține și funcționalități administrative extinse, putând gestiona conturile de utilizatori (studenți, cadre didactice, tutori, parteneri de practică), profilurile companiilor și poate accesa rapoarte și statistici.

**Prorector (ROLE\_PRORECTOR):** Membru al conducerii executive a universității, implicat în aprobarea convențiilor care necesită un nivel superior de validare.

**Responsabilități:** Similar Prodecanului, dar la un nivel ierarhic superior, intervine în fluxul de aprobare pentru anumite categorii de convenții sau conform procedurilor specifice universității, asigurând alinierea cu politicile instituționale generale.

**3.3. Capabilitățile Sistemului**

Sistemul informatic implementat pentru gestionarea convențiilor de practică oferă o serie de funcționalități și caracteristici care răspund nevoilor identificate și obiectivelor proiectului. Aceste capabilități sunt descrise în continuare:

**Managementul Utilizatorilor și al Rolurilor:**

* + Sistemul oferă un mecanism securizat și eficient pentru înregistrarea noilor utilizatori, diferențiat pe roluri (student, cadru didactic, partener de practică etc.). La crearea contului, se generează automat o parolă inițială robustă, comunicată utilizatorului în mod securizat, cu recomandarea de a o schimba la prima autentificare.
  + Autentificarea se realizează printr-un formular de login dedicat (email și parolă), cu măsuri de protecție împotriva tentativelor de acces neautorizat.
  + Accesul la funcționalități și date este strict controlat printr-un sistem granular bazat pe roluri, fiecare rol având permisiuni specifice.
  + Utilizatorii administrativi (Prodecan) dispun de interfețe pentru gestionarea conturilor: creare, vizualizare, actualizare și activare/dezactivare.
  + Fiecare utilizator își poate vizualiza și actualiza propriile informații de profil și își poate schimba parola.
  + Există o funcționalitate de resetare securizată a parolei prin email.

**Gestiunea Convențiilor de Practică:**

* + Studenții inițiază crearea convențiilor printr-un formular web structurat, completând informații despre stagiu, companie, tutore și cadru didactic.
  + Studenții pot modifica convențiile proprii în etape specifice ale fluxului (înainte de prima aprobare sau dacă sunt returnate pentru corecturi).
  + Utilizatorii vizualizează o listă a convențiilor lor cu statusul curent detaliat.
  + Studenții pot anula convențiile inițiate, în stadii incipiente ale fluxului.
  + Fiecare parte implicată (student, partener de practică, tutore , cadru didactic, prodecan, prorector) poate încărca o imagine a semnăturii sale olografe, stocată securizat și asociată convenției.

**Fluxul de Aprobare a Convențiilor:**

* + Platforma facilitează întregul flux de documente, de la inițierea convenției de către student până la aprobarea finală de către autoritățile academice competente.
  + Sistemul automatizează fluxul de lucru pentru aprobarea convențiilor.
  + Partenerii externi( partener de practica si tutori) și personalul academic (cadre didactice, prodecani, prorectori) accesează liste de convenții care necesită revizuirea și aprobarea/respingerea lor, cu posibilitatea de a adăuga motive în caz de respingere.
  + Fluxul de aprobare este flexibil, permițând multiple niveluri ierarhice.
  + Statusul convenției este actualizat în timp real și este vizibil actorilor implicați, menținându-se un istoric al modificărilor.

**Administrarea Companiilor Partenere:**

* + Utilizatorii administrativi (Prodecan) gestionează companiile partenere (adăugare, editare, vizualizare).
  + Administratorii pot gestiona conturile de utilizatori pentru partenerii de practică, asociindu-i companiilor corespunzătoare.
  + Studenții selectează compania dintr-o listă actualizată la crearea convenției.

**Generarea Documentelor și Notificări:**

* + Sistemul generează automat documentul oficial al convenției de practică în formate multiple precum PDF și Word (DOCX). Documentele sunt completate dinamic cu datele din sistem și includ imaginile semnăturilor.
  + Utilizatorii autorizați pot descărca aceste documente.
  + Un sistem de notificări informează utilizatorii despre actualizările de status și acțiunile necesare.

**Interfață și Experiență Utilizator:**

* + Fiecare tip de utilizator dispune de un tablou de bord (dashboard) personalizat care prezintă informații relevante și acces rapid la acțiuni frecvente.
  + Interfața este implementată cu tehnologii frontend moderne (HTML5, CSS3, Bootstrap, JavaScript, jQuery, DataTables) pentru a asigura interactivitate, accesibilitate și adaptabilitate.

**4.Proiectarea**

În acest capitol sunt prezentate aspectele cheie ale proiectării Sistemului de Gestiune a Convențiilor de Practică. Proiectarea a avut ca scop crearea unei soluții software robuste, modulare și ușor de extins, care să satisfacă cerințele funcționale și non-funcționale identificate în capitolul anterior. Se vor detalia arhitectura generală a aplicației, structura bazei de date și principalele diagrame UML care modelează sistemul.

**4.1. Proiectarea Bazei de Date**

Baza de date relațională, implementată în MySQL, este proiectată pentru a stoca în mod eficient și consistent toate informațiile necesare funcționării sistemului. Schema bazei de date reflectă entitățile identificate în sistem și relațiile dintre acestea.

Principalele tabele din baza de date (derivate din entitățile JPA și fișierul schema.sql) includ:

* **users:** Stochează informații despre toți utilizatorii sistemului, indiferent de rol (email, nume, prenume, parolă criptată, rol, status cont, etc.). Include câmpuri specifice pentru diferite roluri, cum ar fi departament, facultate, specializare, titlu\_academic.
* **student:** Stochează date specifice studenților (CNP, data nașterii, an de studiu etc.).
* **companie:** Informații despre companiile partenere (nume, CUI, reprezentant legal, adresă etc.).
* **cadru\_didactic:** Detalii despre cadrele didactice supervizoare (nume, prenume, funcție, specializare).
* **tutori:** Informații despre tutorii de practică din cadrul companiilor (nume, prenume, funcție, email, companie asociată).
* **partners:** Stochează date despre partenerii de practică (reprezentanții legali ai companiilor).
* **prodecan, prorector, rector:** Tabele pentru stocarea datelor specifice acestor roluri administrative, inclusiv semnăturile.
* **conventie (sau conventions):** Tabela centrală care stochează toate detaliile unei convenții de practică, incluzând legături către student, companie, cadru didactic, tutore, perioada, statusul etc.

Relațiile dintre tabele sunt implementate prin chei străine, asigurând integritatea referențială a datelor. De exemplu, tabela conventie conține chei străine către student\_id, companie\_id, cadru\_didactic\_id și tutore\_id.

**5. Implementarea sistemului**

Acest capitol descrie procesul de transpunere a designului sistemului, prezentat în capitolul anterior, într-o aplicație funcțională. Sunt detaliate structura proiectului software, implementarea principalelor modele de date (entități JPA), a repository-urilor pentru accesul la date, a serviciilor care încapsulează logica de business, precum și a controller-elor care gestionează interacțiunea cu utilizatorul. Se vor prezenta fragmente de cod relevante pentru a ilustra modul în care au fost implementate diversele componente ale Sistemului de Gestiune a Convențiilor de Practică.

**5.1. Structura proiectului**

Proiectul este structurat conform convențiilor standard ale Spring Boot, utilizând Maven ca instrument de build și management al dependențelor. Această organizare modulară, cu o separare clară a responsabilităților pe pachete distincte (de exemplu, controller, service, repository, entity, config, security), facilitează navigarea în cod, înțelegerea arhitecturii, mentenanța și extinderea ulterioară a aplicației.

A screenshot of a computer

AI-generated content may be incorrect.

Figura 2 - Structura generală a directoarelor și fișierelor principale

Pachetul src/main/java conține codul sursă Java al aplicației, organizat în sub-pachete corespunzătoare funcționalităților (ex: **ro.upt.ac.conventii.student**, **ro.upt.ac.conventii.conventie**, **ro.upt.ac.conventii.security**). Pachetul **src/main/resources** conține fișierele de configurare (ex**: application.properties**, **schema.sql**) și template-urile HTML Thymeleaf (în directorul **templates**).

**5.2. Implementarea Modelelor (Entități JPA)**

Entitățile JPA (Java Persistence API) reprezintă obiectele de business principale ale aplicației și sunt mapate direct la tabelele din baza de date relațională MySQL. Acestea sunt definite ca clase Java simple (POJO) adnotate cu adnotări JPA specifice (ex: @Entity, @Table, @Id, @Column, @ManyToOne, @ManyToMany) care specifică modul în care atributele clasei corespund coloanelor din tabel și cum sunt gestionate relațiile dintre entități.

**5.2.1. Entitatea User**

**Rol în Aplicație:** Entitatea User este piatra de temelie a sistemului de securitate. Spre deosebire de entitățile Student, Tutore, etc., care stochează date specifice rolului, User stochează informațiile *comune* de autentificare și autorizare. Un Student va avea un cont User asociat, un Tutore va avea alt cont User, și așa mai departe. Acest design decuplează logica de securitate (cine are voie să intre în sistem și ce rol are) de logica de business (ce date specifice are un student).

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Atribute și Rolul lor:**
  + email și password: Stochează credențialele de login. Parola este stocată în baza de date sub formă de hash (criptată), nu în text clar.
  + role: Un String care definește nivelul de acces (ex: "ROLE\_STUDENT"). Spring Security folosește acest câmp pentru a autoriza accesul la diferite URL-uri.
  + enabled: Un flag boolean care permite unui administrator să activeze sau să dezactiveze un cont fără a-l șterge.
  + firstLogin: Un flag de securitate important. Când un cont este creat de un administrator, acesta primește o parolă temporară. Setarea firstLogin la true forțează utilizatorul să își schimbe parola la prima autentificare, crescând securitatea contului.
  + resetPasswordToken: Stochează un token unic și temporar generat atunci când un utilizator solicită resetarea parolei, pentru a valida cererea.
* **Metode Suprascrise:**
  + toString(): Suprascrierea acestei metode este o bună practică pentru debugging. Permite afișarea unei reprezentări textuale clare a unui obiect User în log-uri (de ex: System.out.println(userObject)), fără a expune informații sensibile precum parola.

**5.2.2. Entitatea Tutore**

**Rol în Aplicație:** Entitatea Tutore modelează persoana de contact și îndrumătorul tehnic al studentului din cadrul companiei. Fiecare tutore este asociat unei singure companii, iar o companie poate avea mai mulți tutori. Această legătură este esențială pentru a permite studenților să selecteze tutorele corect după ce au ales compania.

A computer screen shot of a program

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Relația ManyToOne cu Companie**:
  + @ManyToOne: Indică faptul că mai mulți (Many) tutori pot aparține unei singure (One) companii.
  + @JoinColumn(name = "companie\_id"): Aceasta este instrucțiunea concretă pentru JPA. Ea creează în tabelul tutori o coloană numită companie\_id. Această coloană va stoca id-ul companiei corespunzătoare și va funcționa ca o cheie străină (foreign key) către tabelul companie, menținând integritatea referențială la nivelul bazei de date.

**5.2.3. Entitatea Partner**

**Rol în Aplicație:** Entitatea Partner este similară cu Tutore, dar reprezintă o persoană diferită din cadrul companiei: reprezentantul legal sau persoana cu drept de semnătură. Separarea Partner de Tutore permite un flux de aprobare mai granular, unde o persoană (tutorele) avizează conținutul tehnic al stagiului, iar alta (partenerul) oferă aprobarea formală, legală, din partea companiei.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Structură similară cu Tutore**: Se poate observa că structura este aproape identică cu cea a entității Tutore, incluzând o relație @ManyToOne cu Companie. Această similaritate este intenționată, deoarece ambele sunt roluri asociate unei companii. Diferențierea se face la nivel logic, în straturile Service și Controller, unde PartnerController și TutoreController vor gestiona fluxuri de aprobare distincte, chiar dacă datele de bază sunt structurate la fel.

**5.2.4. Entitatea Student**

**Rol în Aplicație:** Entitatea Student este centrală, reprezentând practicantul, beneficiarul final al stagiului de practică. Această clasă stochează toate datele de identificare personală și academică ale studentului, informații necesare pentru generarea automată a documentelor oficiale ale convenției. De asemenea, conține câmpul pentru semnătura digitală, un element esențial în procesul de aprobare electronică.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Adnotări de Clasă:**
  + @Entity: Anunță providerul JPA (Hibernate) că această clasă este o entitate și trebuie să fie mapată la un tabel în baza de date.
  + @Table(name = "student"): Specifică numele exact al tabelului corespunzător din schema MySQL. Fără această adnotare, JPA ar folosi, prin convenție, numele clasei (în acest caz, tot "student").
* **Atribute și Adnotări de Câmp:**
  + @Id și @GeneratedValue(strategy = GenerationType.IDENTITY): Aceste două adnotări lucrează împreună pentru a defini câmpul id ca fiind **cheia primară** a tabelului. IDENTITY este o strategie specifică care deleagă generarea valorii (auto-increment) direct către baza de date MySQL, asigurând unicitatea fiecărui student.
  + @Column(unique = true): Aplicată pe câmpul email, această adnotare impune o **constrângere de unicitate** la nivelul bazei de date. Acest lucru previne crearea a doi studenți cu aceeași adresă de email, menținând integritatea datelor.
  + @Lob și @Column(columnDefinition="LONGBLOB"): Această combinație este folosită pentru atributul semnatura, care stochează imaginea semnăturii ca un șir de octeți (byte[]).
    - @Lob (Large Object) indică faptul că acest câmp poate stoca date de mari dimensiuni.
    - columnDefinition="LONGBLOB" este o specificare directă pentru MySQL, instruindu-l să folosească tipul de date LONGBLOB, care poate stoca până la 4GB de date binare, fiind mai mult decât suficient pentru o imagine.
* **Metode Utilitare:**
  + getNumeComplet(): Este o metodă helper, care nu este mapată la baza de date, dar este utilă în straturile superioare (Service, Controller) sau în template-urile Thymeleaf pentru a afișa numele complet al studentului fără a concatena manual nume și prenume de fiecare dată.

**5.2.5. Entitatea Companie**

**Rol în Aplicație:** Entitatea Companie modelează partenerul de practică, adică firma sau instituția unde studentul își va desfășura stagiul. Aceasta stochează datele de identificare fiscală și juridică ale companiei, precum și datele reprezentantului legal. Aceste informații sunt esențiale pentru validitatea legală a convenției de practică generate de sistem.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Adnotări de Clasă:** Similar cu Student, @Entity și @Table(name = "companie") mapează clasa la tabelul companie din MySQL.
* **Atribute și Adnotări de Câmp:**
  + @Id și @GeneratedValue: Definesc cheia primară id cu auto-increment.
  + @Column(unique = true): Se aplică pe câmpurile cui (Cod Unic de Înregistrare), email și nrRegCom (Număr Registrul Comerțului). Aceasta este o măsură importantă de validare a datelor, prevenind înregistrarea de duplicate pentru aceeași entitate juridică, asigurând astfel că fiecare companie din sistem este unică.
  + @Column(name = "nr\_reg\_com", ...): Aici, atributul Java nrRegCom (scris în camelCase, conform convențiilor Java) este mapat la o coloană nr\_reg\_com (scrisă în snake\_case, o convenție des întâlnită în bazele de date). Această adnotare permite decuplarea convențiilor de denumire între codul Java și schema bazei de date.
* **Metode Utilitare:**
  + getReprezentant() și getCalitate(): Asemenea metodei getNumeComplet din Student, acestea sunt metode helper. getReprezentant() concatenează numele și prenumele reprezentantului legal, iar getCalitate() returnează funcția acestuia. Acestea simplifică afișarea informațiilor în template-urile de generare a documentelor (PDF, Word), unde textul "reprezentată de [Nume Complet] în calitate de [Funcție]" este frecvent utilizat.

**5.2.6. Entitatea CadruDidactic**

**Rol în Aplicație:** Această entitate modelează cadrul didactic din cadrul universității care are rolul de supervizor academic pentru student pe parcursul stagiului de practică. Informațiile stocate (nume, funcție, specializare) sunt folosite pentru a identifica corect supervizorul pe documentele oficiale, iar câmpul semnatura este utilizat în fluxul de avizare electronică a convenției.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Adnotări de Clasă:**
  + @Entity și @Table(name = "cadru\_didactic"): Mapează clasa CadruDidactic la tabelul cadru\_didactic din baza de date.
* **Atribute și Adnotări de Câmp:**
  + @Id și @GeneratedValue(strategy = GenerationType.IDENTITY): Definesc câmpul id ca fiind cheia primară a tabelului, cu valori generate automat de MySQL.
  + semnatura: byte[]: Atributul care stochează imaginea semnăturii cadrului didactic.
    - @Lob: Indică faptul că acest câmp este un "Large Object".
    - @Column(columnDefinition="LONGBLOB"): Forțează crearea coloanei în MySQL cu tipul de date LONGBLOB, capabil să stocheze obiecte binare de mari dimensiuni, asigurând că imaginile semnăturilor, indiferent de mărime, pot fi salvate fără probleme.
* **Metode Utilitare:**
  + getNumeComplet(): La fel ca la entitatea Student, această metodă simplifică logica de afișare în interfața web și în documentele generate, returnând numele și prenumele într-un singur șir de caractere.

**5.2.7. Entitățile Administrative (Prodecan, Prorector, Rector) (todo)**

**Rol în Aplicație:** Aceste trei entități (Prodecan, Prorector, Rector) modelează actorii cu roluri de decizie și aprobare finală în fluxul ierarhic al universității. Deși sunt entități separate, ele împărtășesc o structură aproape identică, stocând date de identificare și, cel mai important, semnătura digitală necesară pentru etapele finale de validare a convențiilor de practică. Separarea lor în entități distincte permite o gestiune granulară a rolurilor și o potențială extindere a funcționalităților specifice fiecărui rol în viitor.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Structura Comună:** Toate cele trei entități administrative conțin câmpuri similare pentru datele de identificare (nume, prenume, email, titluAcademic) și pentru semnătură. Câmpul facultate sau departament ajută la identificarea ariei de responsabilitate a fiecărui oficial.
* **Asocierea cu User:** Este important de menționat că, deși nu există o relație @OneToOne explicită în cod, legătura logică dintre un Prodecan și contul său de User se face prin câmpul email, care este unic în ambele tabele. Serviciile aplicației folosesc acest email pentru a regăsi profilul administrativ corespunzător utilizatorului autentificat.
* **Importanța Câmpului semnatura:** La fel ca la celelalte entități, câmpul semnatura de tip byte[] cu adnotarea @Lob este critic. În ProdecanController, ProrectorController și RectorController, la momentul aprobării unei convenții, conținutul fișierului imagine încărcat de utilizator este extras ca byte[] și salvat în acest câmp al entității corespunzătoare, validând astfel electronic documentul.

**5.3. Implementarea Repository-urilor**

Stratul de Repository este responsabil pentru toată comunicarea cu baza de date. În acest proiect, s-a utilizat **Spring Data JPA**, o componentă a ecosistemului Spring care simplifică masiv implementarea acestui strat. În loc să scriem manual interogări SQL și cod de mapare a rezultatelor la obiecte Java, definim interfețe care extind interfața JpaRepository<T, ID>.

JpaRepository oferă, implicit, un set complet de metode pentru operații CRUD (Create, Read, Update, Delete), cum ar fi save(), findById(), findAll(), și deleteById(). Pe lângă acestea, Spring Data JPA are un mecanism puternic de a genera automat interogări pe baza numelui metodelor definite în interfața noastră.

**5.3.1. ConventionRepository**

**Rol în Aplicație:** Gestionează persistența pentru entitatea centrală, Conventie. Definește numeroase metode personalizate pentru a regăsi convenții pe baza unei varietăți de criterii, esențiale pentru funcționalitatea dashboard-urilor și a listelor specifice fiecărui rol.

A computer code on a blue background

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **extends JpaRepository<Conventie, Long>**: Prin această declarație, ConventionRepository moștenește automat toate metodele CRUD standard pentru entitatea Conventie, a cărei cheie primară este de tip Long.
* **findByStudentEmailOrderByDataIntocmiriiDesc(...)**: Un exemplu excelent al puterii Spring Data. Numele este parsat astfel:
  + findBy: Prefixul pentru o interogare SELECT.
  + StudentEmail: Criteriul de filtrare. Spring Data navighează relația (conventie -> student -> email).
  + OrderByDataIntocmiriiDesc: Adaugă clauza ORDER BY pe câmpul dataIntocmirii în ordine descrescătoare.
* **findTop5By...**: Clauza Top5 limitează numărul de rezultate returnate la 5, util pentru afișarea pe dashboard-uri.

**5.3.2. UserRepository**

**Rol în Aplicație:** Repository esențial pentru securitate, folosit de UserService pentru a găsi un utilizator în timpul autentificării sau resetării parolei.

A computer screen shot of a computer program

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **Optional<User> findByEmail(String email)**: Metoda cheie pentru login. Utilizarea Optional este o bună practică care previne NullPointerException, forțând codul care o apelează să trateze explicit cazul în care utilizatorul nu este găsit.

**5.3.3. StudentRepository**

**Rol în Aplicație:** Gestionează operațiile de persistență pentru entitatea Student. Metoda sa personalizată este crucială pentru a regăsi profilul complet al unui student pe baza adresei de email, informație obținută de la utilizatorul autentificat.

A blue screen with white text

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **extends JpaRepository<Student, Long>**: Oferă automat metode CRUD (Create, Read, Update, Delete) pentru entitatea Student.
* **Optional<Student> findByEmail(String email)**: Implementează o căutare specifică după adresa de email. Returnarea unui Optional<Student> este o practică modernă și sigură, care forțează tratarea cazului în care nu se găsește niciun student cu email-ul respectiv, evitând astfel erori de tip NullPointerException.

**5.3.4. CompanieRepository**

**Rol în Aplicație:** Asigură persistența datelor pentru entitățile Companie. Este utilizat în special în modulele administrative ale prodecanului pentru a gestiona lista de companii partenere și în formularele de creare a convențiilor pentru a popula lista de selecție.

A blue screen with white text

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* Această interfață este un exemplu de bază. Faptul că este goală nu înseamnă că este inutilă. Prin simpla extindere a JpaRepository<Companie, Long>, ea moștenește un set complet de operațiuni CRUD (save, findById, findAll, delete, etc.) pentru entitatea Companie, care sunt suficiente pentru nevoile curente ale aplicației.

**5.3.5. TutoreRepository**

**Rol în Aplicație:** Gestionează entitățile Tutore. Metodele sale personalizate sunt esențiale: una pentru a găsi un tutore pe baza email-ului (pentru a-i asocia convențiile corecte la login) și alta pentru a popula dinamic lista de tutori dintr-o companie, după ce studentul a selectat compania în formular.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* **List<Tutore> findByCompanieId(int companieId)**: O metodă cheie pentru interfața utilizator. Când un student alege o companie dintr-un dropdown, o cerere AJAX este trimisă către un ApiController care folosește această metodă pentru a returna toți tutorii asociați cu id-ul companiei respective, populând dinamic al doilea dropdown, cel de tutori.

**5.3.6. Repository-uri Prodecan, Prorector, Partner**

Următoarele repository-uri urmează un model similar, fiecare având rolul de a gestiona persistența pentru un anumit tip de actor din sistem. Metoda principală definită în fiecare este findByEmail, care permite legătura între contul de User autentificat și profilul specific rolului său.

* **PartnerRepository**:

A computer screen shot of a computer program

AI-generated content may be incorrect.

* **ProdecanRepository**:

A computer screen shot of a computer screen

AI-generated content may be incorrect.

* **ProrectorRepository**:

A blue screen with white text

AI-generated content may be incorrect.

**5.4. Implementarea Serviciilor**

Stratul de Servicii constituie nucleul logic al aplicației. Aceste clase, adnotate în Spring cu @Service, sunt responsabile pentru implementarea regulilor de business, coordonarea operațiunilor care implică multiple surse de date și gestionarea tranzacțiilor pentru a asigura consistența datelor. Ele acționează ca un intermediar esențial între stratul web (Controllere) și stratul de persistență (Repository-uri).

**5.4.1. UserService**

**Rol în Aplicație:** UserService este o componentă critică pentru integrarea cu framework-ul Spring Security. Rolul său principal este de a acționa ca o punte de legătură între mecanismul de autentificare al Spring și datele utilizatorilor stocate în baza de date.

**A computer screen shot of a blue screen

AI-generated content may be incorrect.**

* **implements UserDetailsService**: Prin implementarea acestei interfețe din Spring Security, clasa UserService se angajează să ofere o implementare pentru metoda loadUserByUsername. Acesta este mecanismul prin care Spring Security învață cum să încarce datele unui utilizator din sursa noastră specifică (baza de date MySQL).
* **@Service**: Această adnotare marchează clasa ca fiind o componentă Spring de tip Service, făcând-o eligibilă pentru a fi detectată de mecanismul de scanare a componentelor și pentru a i se putea injecta alte dependențe.
* **@Autowired private UserRepository userRepository;**: Aici se realizează **injectarea dependențelor**. Containerul Spring va crea o instanță a UserRepository și o va asigna automat acestui câmp. UserService nu trebuie să știe cum să creeze un UserRepository, ci doar cum să îl folosească.
* **loadUserByUsername(String email)**: Această metodă este apelată automat de Spring Security în timpul procesului de login.
  1. Primește ca parametru email-ul introdus în formularul de login.
  2. Folosește userRepository.findByEmail(email) pentru a căuta utilizatorul în baza de date.
  3. Dacă utilizatorul nu este găsit, aruncă o excepție UsernameNotFoundException, pe care Spring Security o va interpreta ca un eșec de autentificare.
  4. Dacă este găsit, ar trebui să returneze o instanță de UserDetails (o interfață Spring Security) care conține informațiile necesare pentru autentificare (nume de utilizator, parolă criptată, roluri/autorități).

**5.4.2. PartnerService**

**Rol în Aplicație:** Centralizează logica de business pentru gestionarea partenerilor de practică (reprezentanții legali ai companiilor). Un exemplu elocvent este crearea unui nou partener, operațiune care implică crearea coordonată a două entități în baza de date: un Partner (cu date specifice) și un User (pentru acces la sistem).

A computer screen shot of a blue screen

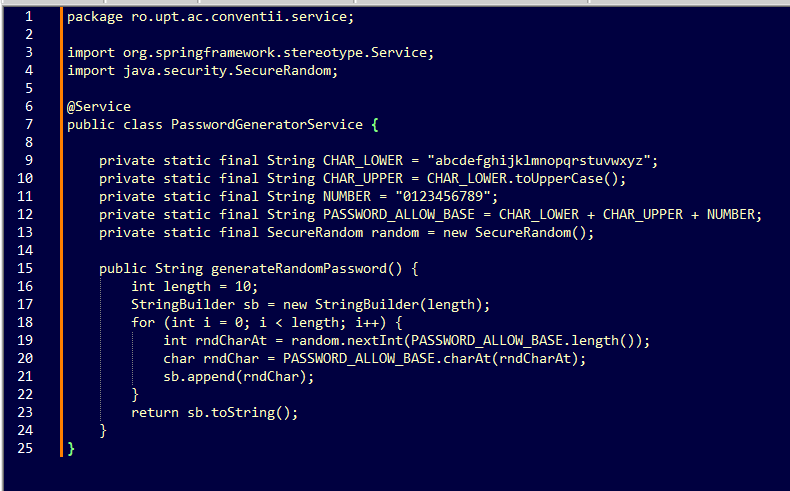
AI-generated content may be incorrect.

* **@Transactional**: Această adnotare este fundamentală. Ea îi spune lui Spring să execute întreaga metodă createPartnerWithUser într-o singură **tranzacție** a bazei de date. Asta înseamnă că fie ambele operațiuni partnerRepository.save() și userRepository.save() se execută cu succes, fie, dacă oricare dintre ele eșuează (de exemplu, din cauza unei constrângeri de unicitate), ambele sunt anulate automat (se face "rollback"). Acest mecanism garantează **consistența datelor**: nu vom avea niciodată un Partner în sistem fără un cont User asociat.
* **passwordEncoder.encode(temporaryPassword)**: Securitatea este prioritară. Înainte de a salva parola generată în baza de date, aceasta este trecută prin BCryptPasswordEncoder. Acesta aplică un algoritm de hashing puternic (BCrypt) pentru a transforma parola într-un șir de caractere nereversibil. Când utilizatorul se va autentifica, Spring va cripta din nou parola introdusă și va compara hash-urile, nu parolele în text clar.
* setFirstLogin(true): La fel ca în explicația entității User, acest flag este setat pe true pentru a obliga partenerul nou creat să-și schimbe parola temporară la prima autentificare, un pas esențial de securitate.
* **PartnerCreationResult**: Este o clasă internă de tip DTO (Data Transfer Object) folosită pentru a returna un rezultat multiplu de la metodă: atât entitatea Partner proaspăt creată, cât și parola temporară generată, pentru ca aceasta să poată fi afișată în interfața prodecanului.

1. **5.4.3. PasswordGeneratorService**

**Rol în Aplicație:** O clasă utilitară simplă, cu o singură responsabilitate: generarea unei parole aleatoare. Decuplarea acestei logici într-un serviciu separat respectă Principiul Responsabilității Unice (Single Responsibility Principle) și permite o testare mai ușoară.

**Explicația Detaliată a Codului:**



* SecureRandom: Se utilizează clasa SecureRandom în loc de Math.random() pentru a genera caracterele aleatorii. SecureRandom este un generator de numere pseudo-aleatorii sigur din punct de vedere criptografic, fiind recomandat pentru aplicații sensibile la securitate, cum ar fi generarea de parole sau token-uri.

**5.5. Implementarea Controller-elor**

Stratul de Controller reprezintă punctul de intrare al aplicației pentru toate interacțiunile venite de la utilizator prin intermediul browser-ului. Clasele din acest strat, adnotate în Spring cu @Controller, sunt responsabile pentru interceptarea cererilor HTTP, extragerea datelor din acestea, invocarea metodelor corespunzătoare din stratul de Servicii pentru a executa logica de business, și, în final, returnarea unui răspuns către utilizator. Acest răspuns este, de regulă, un view (o pagină HTML) redat de Thymeleaf, populat cu datele necesare (modelul).

**5.5.1. Controllere de Securitate și Navigație**

Acest grup de controllere gestionează funcționalitățile de bază, non-specifice unui rol anume, cum ar fi autentificarea, înregistrarea, resetarea parolei și redirecționarea inițială.

**HomeController**

**Rol în Aplicație:** Acționează ca un "dispecer" după autentificare. Preia utilizatorul logat și, în funcție de rolul acestuia, îl redirecționează către panoul de control (dashboard) corespunzător.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* @GetMapping("/"): Această metodă este mapată la calea rădăcină a aplicației (http://localhost:8080/).
* HttpServletRequest request: Primește ca parametru obiectul cererii HTTP, care conține informații despre utilizatorul autentificat.
* request.isUserInRole("ROLE\_..."): Metoda verifică, pe rând, ce rol are utilizatorul curent.
* return "redirect:/.../dashboard": În funcție de rolul găsit, metoda nu returnează un view, ci un string special de redirect. Spring va interpreta acest răspuns și va trimite browser-ului o instrucțiune de a naviga la URL-ul specificat (de ex. /student/dashboard), care va fi preluat de controller-ul corespunzător.

**UserController**

**Rol în Aplicație:** Gestionează paginile publice de înregistrare și login.

A computer screen shot of a blue screen

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* @GetMapping("/register"): Când utilizatorul accesează pagina de înregistrare, această metodă adaugă în model un obiect User gol și returnează view-ul register.html. Thymeleaf va folosi acest obiect gol pentru a lega câmpurile formularului.
* @PostMapping("/register/process"): Această metodă interceptează trimiterea formularului de înregistrare.
  + User user: Spring populează automat acest obiect cu datele din formular.
  + BindingResult result: Conține rezultatele validării și eventualele erori.
  + result.rejectValue(...): Se adaugă o eroare de validare dacă email-ul există deja.
  + passwordEncoder.encode(...): Parola introdusă de utilizator este criptată înainte de a fi salvată în baza de date.

**5.5.2. StudentController**

**Rol în Aplicație:** Este controller-ul principal pentru utilizatorii cu rolul ROLE\_STUDENT. Centralizează toate acțiunile pe care un student le poate efectua, de la vizualizarea panoului de control până la crearea, editarea, trimiterea și exportul convențiilor de practică.

**A screenshot of a computer screen

AI-generated content may be incorrect.**

A computer screen shot of text

AI-generated content may be incorrect.

* @RequestMapping("/student"): Ar putea fi folosit la nivel de clasă pentru a prefixa toate căile metodelor cu /student.
* **Principal principal**: Un parametru injectat automat de Spring Security, care reprezintă utilizatorul autentificat. principal.getName() returnează, în configurația noastră, adresa de email a utilizatorului, fiind esențial pentru a regăsi datele specifice acestuia.
* **Model model**: Un obiect în care controller-ul adaugă date (atribute) pe care dorește să le facă disponibile în view-ul Thymeleaf. De exemplu, model.addAttribute("conventii", listaDeConventii) va permite accesarea variabilei conventii în fișierul HTML.
* @ModelAttribute("conventie") Conventie conventie: La primirea unei cereri POST de la un formular, această adnotare creează un obiect Conventie și populează automat atributele sale cu datele din cerere care au nume corespunzătoare.
* **RedirectAttributes redirectAttributes**: Este folosit pentru a trimite mesaje (de succes sau de eroare) de la o cerere la alta, în special în cazul unui redirect. Atributele adăugate cu addFlashAttribute sunt stocate temporar în sesiune și sunt disponibile doar pentru prima cerere după redirect, fiind apoi șterse. Acest mecanism este ideal pentru afișarea de notificări precum "Convenția a fost salvată cu succes!".
* @PathVariable Long id: Extrage valoarea {id} din URL și o pasează ca parametru metodei, permițând efectuarea de operațiuni pe o entitate specifică.

**5.5.3. ProdecanController**

**Rol în Aplicație:** Acesta este cel mai complex controller din sistem, acționând ca un centru de comandă pentru rolul ROLE\_PRODECAN. Gestionează nu doar fluxul de aprobare a convențiilor, ci și modulele administrative pentru managementul tuturor celorlalte tipuri de utilizatori și entități (studenți, companii, tutori, parteneri, cadre didactice).

A screenshot of a computer program

AI-generated content may be incorrect.

A blue screen with white text

AI-generated content may be incorrect.

**Explicația Detaliată a Codului:**

* @RequestMapping("/prodecan"): Această adnotare la nivel de clasă stabilește un prefix pentru toate căile definite în interior. Astfel, metoda dashboard va răspunde la /prodecan/dashboard, listStudents la /prodecan/studenti, și așa mai departe. Acest lucru ajută la organizarea URL-urilor și la securizarea întregii secțiuni doar pentru rolul ROLE\_PRODECAN.
* **Injectarea Multiplă:** Acest controller demonstrează cum un rol administrativ necesită acces la o gamă largă de date, injectând multiple repository-uri și servicii pentru a putea efectua operațiuni asupra diferitelor entități.
* @RequestParam("semnatura") MultipartFile semnaturaFile: La aprobarea unei convenții, controllerul primește fișierul încărcat de prodecan. @RequestParam leagă fișierul din cererea HTTP la parametrul semnaturaFile de tip MultipartFile, un obiect Spring care facilitează accesul la conținutul, numele și tipul fișierului.

**5.5.4. Controllere pentru Rolurile din Fluxul de Aprobare (TutoreController, PartnerController, ProrectorController, RectorController)**

Acest grup de controllere urmează un model de implementare foarte similar, fiecare fiind dedicat unui rol specific din fluxul de avizare.

**Rol în Aplicație:** Fiecare controller oferă funcționalitățile necesare rolului său:

1. Un **dashboard** cu informații relevante (ex: convenții în așteptare).
2. O **listă** a tuturor convențiilor la care are acces.
3. O pagină de **vizualizare** a detaliilor unei convenții.
4. Acțiuni de **aprobare** și **respingere**.
5. Funcționalitate de **încărcare a semnăturii**.
6. Opțiuni de **export** a documentului (HTML, PDF, Word).

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Explicația Detaliată a Codului:**

* **Authentication authentication**: Spre deosebire de Principal, Authentication este un obiect mai bogat care, pe lângă numele utilizatorului, poate conține și lista de autorități (roluri) și alte detalii. În acest cod, este folosit similar cu Principal pentru a obține utilizatorul curent.
* **Logica de Export (exportConventiePdf)**: Aceste metode sunt mai complexe. Ele nu returnează un nume de view, ci un ResponseEntity<byte[]>. Logica internă construiește un document (PDF, Word) în memorie, îl scrie într-un ByteArrayOutputStream, și apoi îl returnează ca un șir de octeți împreună cu headere HTTP corespunzătoare (Content-Type, Content-Disposition) pentru a instrui browser-ul să descarce fișierul.

# BIBLIOGRAFIE

[1] Spring Framework Documentation, <https://docs.spring.io/spring-framework/reference/>

[2] Thymeleaf Documentation, <https://www.thymeleaf.org/documentation.html>

[3] Java Documentation, <https://docs.oracle.com/en/java/>

[4] Bootstrap Documentation, <https://getbootstrap.com/docs/>

[5] MySQL Documentation, <https://dev.mysql.com/doc/>

**Lista Figurilor**

Figura 1 - Arhitectura Spring Framework Runtime