

UNIVERSITATEA POLITEHNICA TIMIȘOARA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
SPECIALIZAREA INGINERIA SISTEMELOR

LUCRARE DE LICENȚĂ

APLICAȚIE WEB PENTRU GESTIONAREA
PLANURILOR DE ÎNVĂȚĂMÂNT

Candidat: Daniel-Ionuț MÎRZA

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

Sesiunea: Februarie 2025

CUPRINS

1	INTRODUCERE	3
2	TEHNOLOGII WEB	5
2.1	LIMBAJUL DE PROGRAMARE JAVA	5
2.2	BIBLIOTECA SPRING BOOT	9
2.3	BIBLIOTECA APACHE POI	11
2.4	SISTEM DE GESTIONARE DE DATE MYSQL	13
2.5	BIBLIOTECA THYMELEAF	15
2.6	BOOTSTRAP 5	17
3	CERINȚA ȘI SPECIFICAȚIILE PROIECTULUI	18
3.1	CERINȚE FUNCȚIONALE	18
3.2	CERINȚE NEFUNCȚIONALE	18
3.3	CERINȚE TEHNICE	19
3.4	FUNCȚIONALITATEA PROIECTULUI	19
3.5	ANALIZA DE RISC	20
4	PROIECTARE	21
4.1	SCHEMA BLOC	21
4.2	DIAGRAMA DE CLASE, SECVENTA SI STARE	22
4.3	SCHEMA BAZEI DE DATE	23
5	IMPLEMENTAREA	25
5.1	DESCRIERE CLASE	25
5.1.1	CLASA PlanInvatamant	25
5.1.2	CLASA PlanInvatamantController	26
5.1.3	CLASA Disciplina	29
5.1.4	CLASA DisciplinaZi si DisciplinaId	30

1 INTRODUCERE

Într-un mediu academic în continuă schimbare, gestionarea eficientă a informațiilor referitoare la planurile de învățământ reprezintă o provocare semnificativă. Lucrarea de licență intitulată *"Aplicație web pentru gestionarea planurilor de învățământ"* își propune să răspundă acestei provocări prin dezvoltarea unei soluții software care să automatizeze și să simplifice procesul de administrare a datelor academice.

Aplicația este construită folosind tehnologii moderne, cum ar fi Java, Apache POI, MySQL, Spring Boot și Thymeleaf, oferind o platformă scalabilă și intuitivă pentru utilizatori. Dezvoltarea aplicației se bazează pe procesarea fișierelor Excel și Word existente, care conțin planurile de învățământ și fișele de disciplină, asigurând astfel o integrare eficientă a datelor.

Proiectul include trei module principale:

1. **Modul administrator** – permite autentificarea, importul planurilor de învățământ, gestionarea fișelor de disciplină și alocarea cadrelor didactice la discipline.
2. **Modul cadru didactic** – oferă posibilitatea autentificării, vizualizării și încărcării de fișe de disciplină.
3. **Modul student** – facilitează accesul la fișele de disciplină filtrate după generație, ciclu de studii și an de studiu.

Lucrarea evidențiază procesul de dezvoltare al aplicației, pornind de la modelarea bazei de date relaționale și integrarea fișierelor de tip Excel și Word, până la implementarea funcționalităților de administrare și utilizare practică.

Prin această aplicație, se dorește crearea unei platforme care să îmbunătățească experiența utilizatorilor și să reducă eforturile administrative, contribuind astfel la o gestionare mai eficientă a resurselor academice.

În capitolul 2 prezentăm tehnologiile web utilizate în cadrul proiectului, oferind o descriere detaliată a principalelor instrumente și framework-uri utilizate. Acestea includ limbajul de programare **Java**, framework-ul **Spring Boot** pentru dezvoltarea aplicațiilor web, **Thymeleaf** pentru partea de interfață, precum și biblioteca **Apache POI** pentru procesarea fișierelor Excel și Word. De asemenea, sistemul de gestionare a bazelor de date **MySQL** este folosit pentru stocarea și administrarea datelor într-un model relațional. Capitolul explorează funcționalitățile oferite de fiecare tehnologie și justifică alegerea acestora în contextul cerințelor aplicației.

În capitolul 3 descriem specificațiile proiectului, detaliind cerințele funcționale și nefuncționale, structura arhitecturală a aplicației, precum și principalele module dezvoltate. Acest capitol include o analiză a fluxurilor de lucru, relațiile dintre componente și mecanismele utilizate pentru a asigura funcționalitatea dorită, oferind o imagine clară asupra designului și implementării proiectului.

În capitolul 4 prezentăm proiectarea proiectului, incluzând arhitectura sistemului, modelarea bazei de date și designul principalelor componente software. Sunt detaliate diagrama de clase, diagrama de relații ale bazei de date, precum și structura interfeței utilizator, evidențiind modul în care elementele aplicației colaborează pentru a satisface cerințele specificate.

În capitolul 5

În capitolul 6

În capitolul 7

2 TEHNOLOGII WEB

2.1 LIMBAJUL DE PROGRAMARE JAVA

Java este un limbaj de programare orientată pe obiecte, bazată pe clase care este proiectată să aibă cât mai puține *dependențe* posibile. Este renumită pentru portabilitatea sa, oferind programatorilor de a scrie cod care poate rula pe orice platformă ce suportă Java fără a necesita recompilare, conform sloganului „Write Once, Run Anywhere” (WORA).

Introducere

Limbajul de programare Java a fost creat de Mike Sheridan, Patrik Naughton și James Gosling, cel din urmă cunoscut drept "părintele Java". Acesta a dezvoltat limbajul în timp ce lucra pentru compania Sun Microsystems, începând din 1991. Proiectul a fost inițial cunoscut sub numele de cod "Oak", iar scopul său era crearea unui limbaj portabil pentru dispozitive electronice. În cele din urmă, limbajul a fost redenumit "Java" și a fost lansat public în 1995.

Scurta istorie

James Gosling, un informatician de origine canadiană, este apreciat pentru contribuția sa semnificativă la tehnologia modernă a calculatoarelor și a fost influențat de alte limbaje precum C și C++, dar a dorit să creeze un limbaj mai simplu, mai sigur și orientat pe obiecte. Sub conducerea sa, Java a devenit unul dintre cele mai populare limbaje de programare din lume, folosit pe scară largă în dezvoltarea de aplicații pentru servere, desktop, dispozitive mobile și web.

După plecarea sa de la Sun Microsystems (care a fost achiziționată de Oracle în 2010), Gosling a continuat să lucreze în industrie și să contribuie la diverse proiecte tehnologice. Astăzi, el este considerat o figură emblematică în domeniul programării.

Parintele Java

Java este recunoscută pentru securitatea sa robustă, care este asigurată printr-o combinație de caracteristici integrate și mecanisme avansate. Printre cele mai importante măsuri se numără sandboxing-ul, care izolează codul și previne accesul neautorizat la resurse, precum și verificarea bytecode-ului de către JVM, care asigură integritatea și conformitatea codului. Un alt element central este Security Manager-ul, care controlează accesul aplicațiilor la fișiere, rețele și alte resurse critice.

Securitate

Java este un limbaj de programare cu mai multe caracteristici remarcabile care îl fac ideal pentru dezvoltarea de aplicații variate. În primul rând, Java este orientat pe obiecte, ceea ce înseamnă că aproape totul este definit ca un obiect, cu excepția tipurilor de date primitive. Această abordare face ca programele să fie mai ușor de întreținut și de extins, datorită modularității și reutilizării codului.

Intro OOP

Programarea orientată pe obiecte (OOP) este un model de dezvoltare software care organizează codul în jurul conceptelor de **clase** și **obiecte**. O **clasă** reprezintă o schiță sau un model abstract care definește atributele și comportamentele specifice unui tip de entitate. Pe baza acestei clase, se pot crea obiecte, care sunt instanțe concrete ce conțin valori specifice și pot efectua acțiuni definite de comportamentele clasei.

Clase si Obiecte

Un alt concept fundamental este **moștenirea**, care permite unei clase să preia atributele și metodele unei alte clase, sprijinind astfel reutilizarea codului și crearea unor relații ierarhice între clase. Acest mecanism facilitează extinderea funcționalităților fără a modifica clasele existente, permițând dezvoltarea software-ului într-un mod modular și flexibil.

Mostenire

Pentru a organiza mai bine codul, Java utilizează **pachetele**, care grupează clasele și interfețele în module logice. Acestea ajută la structurarea aplicațiilor mari și la evitarea conflictelor între numele claselor din diferite părți ale unui proiect sau din biblioteci externe.

Pachete

Un alt principiu esențial al OOP este **încapsularea**, care protejează datele interne ale unei clase prin ascunderea acestora și furnizarea unui acces controlat prin metode specifice. Acest lucru contribuie la menținerea securității și integrității datelor și permite modificarea logicii interne fără a afecta alte părți ale aplicației.

Incapsulare

Prin combinarea acestor concepte, programarea orientată pe obiecte permite crearea unor aplicații bine structurate, ușor de întreținut și scalabile. Fiecare concept contribuie la organizarea și gestionarea complexității, asigurând un proces de dezvoltare mai eficient și mai robust.

Concluzie OOP

O altă caracteristică esențială este independența platformei. Programele Java sunt compilate într-un format intermediar numit bytecode, care poate fi executat pe orice platformă ce suportă Java Virtual Machine (JVM). Această abilitate este cunoscută prin sloganul „Write Once, Run Anywhere”.

Independenta platformei

Limbajul suportă multitasking prin funcționalități de multi-threading, permițând rularea simultană a mai multor fire de execuție, ceea ce duce la performanțe crescute în aplicații complexe.

Multi-threading

Biblioteca standard Java este extrem de bogată, oferind suport pentru o gamă largă de funcționalități, de la manipularea colecțiilor de date până la lucrul cu rețele. Toate aceste caracteristici fac din Java un limbaj versatil, utilizat pe scară largă în dezvoltarea de aplicații desktop, mobile, web și enterprise.

Biblioteca bogata

Java este împărțit în mai multe ediții, fiecare concepută pentru un anumit tip de aplicații și utilizatori. Cele trei ediții principale sunt Java Standard Edition (SE), Java Enterprise Edition (EE), și Java Micro Edition (ME). Fiecare dintre acestea are un set specific de funcționalități și biblioteci, adaptate cerințelor mediilor în care sunt utilizate.

Versiuni Java

Java Standard Edition (SE) este ediția de bază a platformei Java și oferă toate funcționalitățile necesare pentru dezvoltarea aplicațiilor independente sau desktop. Este cea mai utilizată ediție pentru învățarea limbajului Java și acoperă toate caracteristicile fundamentale ale acestuia.

Java SE

Java Enterprise Edition (EE), cunoscut acum ca Jakarta EE, este construit pe baza Java SE și extinde funcționalitățile acestuia pentru a permite dezvoltarea de aplicații complexe, distribuite și scalabile, utilizate în mediul enterprise.

Java EE

Java Micro Edition (ME) este o ediție ușoară și specializată a platformei Java, destinată dispozitivelor cu resurse limitate, cum ar fi telefoanele mobile, sistemele integrate (embedded systems), și dispozitivele IoT (Internet of Things).

Java ME

Java Persistence API (JPA) este un standard Java pentru gestionarea persistenței datelor, adică pentru stocarea și accesarea obiectelor Java într-o bază de date relațională. Este parte a platformei Java EE (Enterprise Edition), dar poate fi utilizată și în aplicații Java SE (Standard Edition). JPA oferă o modalitate de a interacționa cu bazele de date folosind concepte orientate pe obiecte, eliminând necesitatea de a scrie SQL manual.

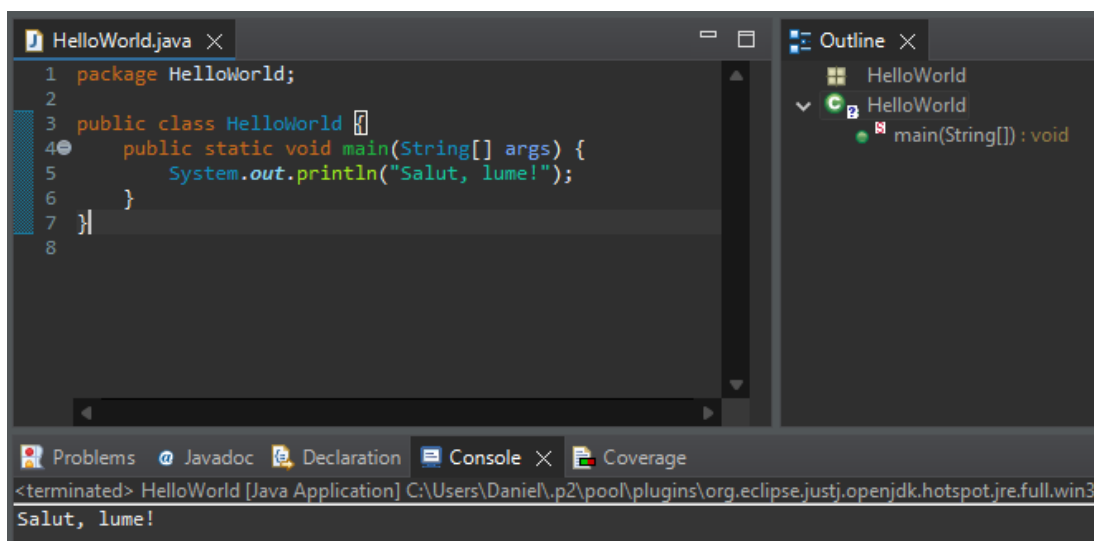
JPA

Unul dintre principalele beneficii este utilizarea mapării obiect-relaționale (ORM), care asociază clasele Java cu tabelele din baza de date și atributele acestora cu coloanele corespunzătoare. Aceasta se realizează cu ajutorul adnotărilor precum `@Entity` pentru declararea entităților persistente, `@Table` pentru specificarea tabelului asociat sau `@Id` pentru identificatori unici.

Un alt aspect important al JPA este suportul său pentru un limbaj de interogare asemănător SQL, denumit JPQL (Java Persistence Query Language), care permite efectuarea interogărilor pe obiecte, păstrând însă un nivel ridicat de abstractizare. De asemenea, JPA gestionează automat ciclul de viață al entităților, de la crearea acestora și atașarea la contextul persistent, până la ștergere.

Caracteristici JPA

Un program simplu Java constă în general dintr-o clasă care conține o metodă principală (main), unde începe execuția programului.



```
1 package HelloWorld;
2
3 public class HelloWorld {
4     public static void main(String[] args) {
5         System.out.println("Salut, lume!");
6     }
7 }
8
```

Outline:

- HelloWorld
- main(String[]) : void

Console:

<terminated> HelloWorld [Java Application] C:\Users\Daniel\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32
Salut, lume!

```
public class HelloWorld {
```

Acesta definește o clasă numită HelloWorld. În Java, toate programele sunt organizate în clase, care reprezintă unități fundamentale ale codului. Cuvântul cheie `public` indică faptul că această clasă poate fi accesată de oriunde în program.

```
public static void main(String[] args) {
```

Aceasta este metoda principală care reprezintă punctul de intrare al programului. Fiecare aplicație Java pornește execuția din metoda `main`.

- `public` înseamnă că metoda poate fi apelată de JVM din afara clasei.
- `static` indică faptul că metoda aparține clasei și poate fi rulată fără a crea o instanță a clasei.
- `void` semnalează că metoda nu returnează nicio valoare.
- `String[] args` este un parametru care permite transmiterea de argumente către program din linia de comandă.

```
System.out.println("Salut, lume!");
```

Această linie utilizează metoda `println` din clasa `System.out` pentru a afișa textul „Salut, lume!” urmat de un rând nou în consolă.

- `System.out` este un obiect standard pentru ieșire, folosit pentru a trimite mesaje către consolă.
- `println` este metoda utilizată pentru afișarea mesajului și adăugarea unui caracter de rând nou la final.

Când acest cod este executat, rezultatul va fi:

```
Salut, lume!
```

Program simplu

Java este un limbaj de programare versatil, sigur și scalabil, care s-a impus ca un standard în dezvoltarea software. De-a lungul anilor, a evoluat pentru a satisface cerințele unor domenii diverse, de la aplicații enterprise și dezvoltare web, până la soluții pentru dispozitive mobile și IoT. Caracteristicile sale precum portabilitatea, gestionarea automată a memoriei și orientarea pe obiecte îl fac ușor de utilizat și de înțeles, fiind ideal atât pentru începători, cât și pentru profesioniști.

Scurta concluzie

2.2 BIBLIOTECA SPRING BOOT

Spring Boot este un framework Java care facilitează crearea și execuția aplicațiilor Java. Simplifică procesul de configurare și setare, permițându-le dezvoltărilor să se concentreze mai mult pe scrierea de cod a aplicațiilor.

Intro

Spring Boot este dezvoltată de către Pivotal Team și este o combinație dintre Spring Framework și Embedded Servers. În octombrie 2012 un client pe nume Mike Youngstrom a făcut o cerere Jira cerând „bootstrapping the spring framework” pentru a putea porni mai repede. Astfel, la începutul lui 2013 Spring Boot a fost creat.

Scurta istorie

Printre avantajele principale ale utilizării acestui framework se numără configurarea automată, care detectează librăriile incluse în proiect și configurează automat componentele necesare. Acest lucru elimină necesitatea unei configurări manuale complexe. De asemenea, Spring Boot permite crearea de aplicații autonome, care pot fi rulate direct ca fișiere JAR, fără a necesita un server extern, deoarece acesta este inclus în aplicație. O altă caracteristică importantă este utilizarea Starter POMs, care ajută la gestionarea simplificată a dependențelor prin includerea unui singur modul, evitând configurările complicate. În plus, Spring Boot Actuator oferă funcționalități avansate pentru monitorizarea și diagnosticarea aplicațiilor.

Avantaje

Din punct de vedere al componentelor de bază, Spring Boot include Starter Templates, care sunt seturi predefinite de dependențe pentru diferite tipuri de aplicații. Configurarea aplicației este simplificată prin utilizarea fișierelor `application.properties` sau `application.yml`, eliminând nevoia de a lucra cu fișiere XML lungi și complicate. O altă componentă utilă este Spring Boot Initializr, o interfață web ce permite generarea rapidă a proiectelor preconfigurate, oferind utilizatorilor opțiunea de a selecta dependențele, versiunea Java și alte detalii. Aceste caracteristici fac din Spring Boot un instrument ideal pentru dezvoltarea rapidă și eficientă a aplicațiilor Java.

Caracteristici

Crearea unei aplicații Spring Boot începe prin configurarea proiectului pe site-ul **Spring Initializr**, unde dezvoltatorul alege setările necesare, precum sistemul de build (Maven), limbajul (Java) și dependențele esențiale (Spring Web, Spring Data JPA, H2 Database). Proiectul descărcat este apoi importat într-un IDE, iar Maven se ocupă de gestionarea automată a dependențelor. Structura proiectului este deja configurată, incluzând foldere pentru cod sursă, fișiere de configurare și specificațiile Maven.

Pentru a adăuga funcționalitate, dezvoltatorul creează un controller simplu, o clasă Java adnotată cu `@RestController` și `@GetMapping`, care gestionează cererile HTTP și răspunde cu mesaje predefinite. Configurarea suplimentară a aplicației se face în fișierul `application.properties`, unde se pot personaliza detalii precum portul serverului și numele aplicației.

```
1 package HelloWorld;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 public class SalutController {
8
9     @GetMapping("/salut")
10    public String spuneSalut() {
11        return "Salut, lume!";
12    }
13 }
14
```

Outline: HelloWorld > SalutController
spuneSalut() : String

Console: <terminated> HelloWorld [Java Application] C:\Users\Daniel\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_2020.10\jre\bin\java.exe
Salut, lume!

După ce aplicația este pornită prin rularea clasei principale, aceasta poate fi testată prin accesarea unei rute definite. În timp, funcționalitățile pot fi extinse prin adăugarea de persistare a datelor, servicii pentru logica aplicației și teste pentru validarea codului, transformând aplicația inițială într-un proiect scalabil și bine structurat.

Program simplu Spring Boot

Spring și Spring Boot sunt două instrumente din ecosistemul Spring, dar au scopuri diferite. Spring este un framework general pentru dezvoltarea aplicațiilor Java, oferind multă flexibilitate, dar necesită configurări manuale extinse. Pe de altă parte, Spring Boot simplifică procesul de dezvoltare prin configurări implicite și este ideal pentru aplicații standalone, gata de producție.

Când vine vorba de lansare și rulare, aplicațiile dezvoltate cu Spring au nevoie de un server extern, cum ar fi Tomcat, pentru a putea rula. În schimb, Spring Boot permite rularea aplicațiilor ca standalone, având servere încorporate, astfel încât aplicațiile pot fi lansate direct cu o simplă comandă `java -jar`.

În final, Spring necesită mai mult timp pentru configurare, dar oferă flexibilitate maximă, în timp ce Spring Boot este ideal pentru lansarea rapidă a aplicațiilor. Alegerea între cele două depinde de specificul proiectului: Spring este potrivit pentru aplicații complexe, în timp ce Spring Boot este soluția preferată pentru proiecte moderne și rapide.

Spring vs Spring Boot

Spring Boot este un framework modern și eficient, creat pentru a simplifica dezvoltarea aplicațiilor Java. Cu un sistem de configurare automat, suport pentru microservicii, integrare perfectă cu ecosistemul Spring și un set vast de funcționalități, Spring Boot accelerează procesul de dezvoltare și reduce complexitatea. Este o soluție ideală atât pentru aplicații mici, cât și pentru sisteme complexe și scalabile, fiind susținut de o comunitate activă și documentație extinsă. Alegerea Spring Boot pentru un proiect asigură flexibilitate, performanță și ușurință în utilizare, adaptându-se perfect cerințelor moderne de dezvoltare software. **Concluzie**

2.3 BIBLIOTECA APACHE POI

Apache POI este o bibliotecă Java utilizată pentru a citi și scrie fișiere Microsoft Office, inclusiv formate precum Excel (XLS, XLSX), Word (DOC, DOCX) și PowerPoint (PPT, PPTX). Este foarte utilă în aplicații care necesită manipularea documentelor Office în mod programatic, fie că este vorba de generarea rapoartelor în format Excel, modificarea documentelor Word sau crearea de prezentări PowerPoint.

Intro

Apache POI a fost dezvoltat ca parte a **Apache Software Foundation** pentru a oferi o soluție open-source destinată manipulării fișierelor Microsoft Office. Proiectul a început în anii 2000, cu obiectivul inițial de a crea un API pentru citirea și scrierea fișierelor Excel în formatul vechi (.xls). Aceasta a fost realizată prin API-ul **HSSF** (Horrible Spreadsheet Format), numele său având o notă ironică, caracteristică proiectelor din acea perioadă.

Pe măsură ce Microsoft a introdus formatele bazate pe XML odată cu lansarea Office 2007, Apache POI și-a extins funcționalitățile pentru a include suport pentru noile formate (.xlsx, .docx, .pptx). Aceasta a condus la dezvoltarea API-urilor **XSSF** pentru Excel, **XWPF** pentru Word și **XSLF** pentru PowerPoint.

Istorie

Apache POI oferă funcționalități extinse pentru manipularea fișierelor Microsoft Office. În ceea ce privește fișierele Excel, suportă atât formatele mai vechi (.xls) prin API-ul HSSF (Horrible Spreadsheet Format), cât și cele moderne (.xlsx) prin API-ul XSSF (XML Spreadsheet Format). Pentru documentele Word, bibliotecile HWPf și XWPF permit lucrul cu fișierele DOC și DOCX, respectiv. Similar, fișierele PowerPoint sunt gestionate prin HSLF pentru PPT și XSLF pentru PPTX. În plus, POI facilitează manipularea obiectelor precum foi de calcul, celule, paragrafe, tabele, imagini și stiluri, fiind astfel potrivit pentru o gamă variată de aplicații.

Caracteristici

Apache POI este un proiect open-source, ceea ce îl face gratuit și accesibil oricui. Este compatibil cu multiple formate Microsoft Office, oferind flexibilitate în manipularea fișierelor din diferite versiuni de Office. În plus, API-ul său este bine documentat și susținut de o comunitate activă, ceea ce facilitează învățarea și integrarea sa în proiecte software.

Avantaje

Exemplul următor demonstrează modul de utilizare a bibliotecii Apache POI pentru a citi și scrie fișiere Excel în format .xlsx. Procesul începe prin importul și deschiderea fișierului folosind un flux de intrare (*FileInputStream*). Fișierul este apoi încărcat într-un obiect *XSSFWorkbook*, care reprezintă întreaga structură a documentului Excel. Accesul la o anumită foaie se face fie prin index (de exemplu, *getSheetAt(0)* pentru prima foaie), fie prin nume.

Pentru citirea unei valori, este accesat rândul dorit folosind metoda *getRow()*, iar celula corespunzătoare este obținută cu *getCell()*. Valoarea este extrasă sub forma unui șir de caractere prin metoda *getStringCellValue()*. Dacă celula conține date de alt tip, cum ar fi numerice, se folosește metoda *getNumericCellValue()*.

```
ExcelExample.java X
1 package HelloWorld;
2 import org.apache.poi.xssf.usermodel.XSSFSheet;
3 import org.apache.poi.xssf.usermodel.XSSFWorkbook;
4
5 import java.io.FileInputStream;
6 import java.io.FileOutputStream;
7 import java.io.IOException;
8
9 public class ExcelExample {
10     public static void main(String[] args) throws IOException
11     {
12         // Creeare workbook și sheet
13         FileInputStream file = new FileInputStream("exemplu.xlsx");
14         XSSFWorkbook workbook = new XSSFWorkbook(file);
15         XSSFSheet sheet = workbook.getSheetAt(0);
16
17         // Citirea unei valori dintr-o celulă
18         String valoare = sheet.getRow(0).getCell(0).getStringCellValue();
19         System.out.println("Valoarea citită: " + valoare);
20
21         // Scrierea unei valori într-o celulă
22         sheet.getRow(0).createCell(1).setCellValue("Nouă valoare");
23
24         // Salvarea fișierului modificat
25         FileOutputStream outputStream = new FileOutputStream("exemplu_modificat.xlsx");
26         workbook.write(outputStream);
27
28         // Închidere workbook și file stream
29         workbook.close();
30         file.close();
31         outputStream.close();
32     }
33 }
34
```

Scrierea unei valori într-o celulă implică fie crearea unei celule noi cu *createCell()*, fie suprascrierea unei celule existente. Valoarea este apoi setată prin metoda *setCellValue()*.

Modificările aduse fișierului sunt salvate prin deschiderea unui flux de ieșire (*FileOutputStream*) și apelarea metodei *write()* a obiectului *XSSFWorkbook*. După ce modificările sunt scrise, fluxurile de intrare și ieșire sunt închise, împreună cu workbook-ul, pentru a elibera resursele utilizate.

Pentru utilizarea bibliotecii Apache POI într-un proiect, trebuie să incluzi dependența corespunzătoare în fișierul de configurare al proiectului, cum ar fi *pom.xml* pentru Maven. Gestionarea excepțiilor este esențială, deoarece operațiile de citire și scriere implică riscul de erori legate de I/O.

Exemplu de utilizare POI

Apache POI reprezintă o soluție robustă și flexibilă pentru manipularea fișierelor Microsoft Office în cadrul aplicațiilor Java. Prin funcționalitățile sale, permite citirea, scrierea și gestionarea completă a datelor din formatele Excel, Word și PowerPoint, atât în versiunile clasice, cât și în cele moderne bazate pe XML.

Concluzie

2.4 SISTEM DE GESTIONARE DE DATE MYSQL

MySQL este un sistem de gestionare a bazelor de date relaționale (RDBMS) foarte popular și puternic, utilizat pentru a stoca și gestiona date. Este open-source și este folosit în mod frecvent în aplicații web, fiind compatibil cu diferite sisteme de operare, precum Linux, Windows și macOS.

Intro

O bază de date este o colecție organizată de informații sau date care sunt stocate și gestionate electronic, de obicei într-un sistem de gestionare a bazelor de date (DBMS – Database Management System). Baza de date permite stocarea, manipularea, accesarea și actualizarea eficientă a datelor.

Despre baza de date

MySQL a fost lansat oficial în 1995 de către Michael Widenius, David Axmark și Allan Larsson, fondatorii companiei MySQL AB din Suedia, fiind dezvoltat inițial ca un sistem de baze de date rapid și ușor de utilizat pentru aplicații web. În anul 2000, MySQL a devenit open-source sub licența GNU GPL, ceea ce a contribuit semnificativ la creșterea popularității sale în rândul dezvoltatorilor.

În 2008, compania MySQL AB a fost achiziționată de Sun Microsystems pentru suma de 1 miliard USD, marcând o etapă importantă în evoluția produsului prin adăugarea de resurse și investiții. Doi ani mai târziu, în 2010, Sun Microsystems a fost cumpărată de Oracle Corporation, care a preluat și dezvoltarea MySQL. Deși această schimbare a provocat controverse în comunitatea open-source, Oracle a continuat să îmbunătățească MySQL.

O etapă semnificativă în istoria MySQL a avut loc în 2013, odată cu lansarea versiunii 5.6, care a introdus îmbunătățiri majore în performanță, replicare și optimizarea interogărilor. În 2021, versiunea MySQL 8.0 a devenit standardul pentru cele mai noi funcționalități, incluzând suport pentru JSON, optimizări în performanță și administrare avansată.

Istoria MySQL

Numele MySQL provine din combinația cuvântului „SQL” (Structured Query Language), limbajul standard utilizat pentru gestionarea bazelor de date, și „My”, care face referire la fiica lui Michael Widenius (unul dintre fondatorii MySQL), al cărei prenume este „My”. Această combinație reflectă atât o notă personală, cât și un accent tehnic, ceea ce a contribuit la unicitatea brandului.

Denumire MySQL

Logo-ul MySQL este reprezentat de o delfin albastru, simbolizând agilitatea, viteza și ușurința de utilizare – trăsături care caracterizează acest sistem de gestionare a bazelor de date. Delfinul din logo este numit „Sakila”, nume ales de comunitatea MySQL printr-un concurs organizat de dezvoltatori.

Logo-ul MySQL

MySQL este un sistem de gestionare a bazelor de date relaționale (RDBMS), care organizează datele în tabele ce conțin rânduri și coloane. Utilizează limbajul SQL (Structured

Query Language) pentru interogare și gestionare. Este recunoscut pentru performanța ridicată, oferind viteze mari de procesare a datelor și gestionând volume mari de informații. De asemenea, MySQL include controale de acces avansate și criptare pentru securitate, fiind capabil să gestioneze baze de date de dimensiuni mari. Este compatibil cu diverse limbaje de programare, cum ar fi PHP, Java și Python, ceea ce îl face ușor de integrat în diferite tipuri de aplicații.

Caracteristici

MySQL este un instrument open-source, ceea ce înseamnă că este gratuit și beneficiază de sprijinul unei comunități mari care contribuie la îmbunătățirea sa constantă. Este compatibil cu mai multe platforme, inclusiv Windows, Linux și macOS, ceea ce îl face extrem de versatil. Interfața prietenoasă și instrumentele ușor de utilizat îl fac accesibil pentru dezvoltatorii aflați la început de drum. Totodată, MySQL este recunoscut pentru simplitatea configurării și implementării, oferind performanță ridicată la un cost redus.

Avantaje

SQL și MySQL reprezintă concepte diferite, deși sunt strâns legate. SQL este un limbaj standardizat utilizat pentru gestionarea și interogarea bazelor de date relaționale. Este un instrument teoretic și practic care permite dezvoltatorilor să creeze, să actualizeze, să șteargă și să interogheze datele din tabelele bazelor de date. Printre comenzile SQL comune se numără CREATE TABLE pentru crearea tabelelor, SELECT pentru extragerea datelor, INSERT pentru adăugarea de date noi, UPDATE pentru actualizarea datelor existente și DELETE pentru ștergerea acestora. SQL este un limbaj universal, independent de platformă, utilizat de toate sistemele de gestionare a bazelor de date relaționale (RDBMS), inclusiv MySQL, Oracle, PostgreSQL și Microsoft SQL Server.

Pe de altă parte, MySQL este un sistem de gestionare a bazelor de date relaționale (RDBMS) care implementează limbajul SQL. MySQL este un software concret utilizat pentru stocarea, gestionarea și accesarea datelor. Spre deosebire de SQL, care este doar un standard teoretic, MySQL oferă o platformă practică și completă pentru administrarea bazelor de date, incluzând funcții avansate precum securitatea, replicarea și optimizarea performanței. MySQL este specific, fiind dezvoltat ca un software concret care extinde standardul SQL pentru a include caracteristici practice necesare gestionării bazelor de date.

MySQL vs SQL

MySQL este un sistem de gestionare a bazelor de date relaționale (RDBMS) care utilizează limbajul standard SQL pentru a stoca, administra și accesa datele. Dezvoltat inițial ca un instrument rapid, simplu și open-source, MySQL a devenit un standard de facto în industria IT datorită versatilității, performanței ridicate și compatibilității cu diverse platforme și limbaje de programare. Cu o istorie solidă și sprijin din partea comunității open-source, dar și al companiei Oracle, MySQL este folosit pe scară largă în aplicații web, sisteme de management al conținutului și platforme critice, fiind apreciat pentru combinația sa de accesibilitate și funcționalitate avansată.

Concluzie

2.5 BIBLIOTECA THYMELEAF

Thymeleaf este un motor de template Java folosit pentru a genera cod HTML, XML, JavaScript, CSS și text. Este des utilizat în aplicațiile de tip Spring Boot pentru a crea interfețe de utilizator dinamice și intuitive. Thymeleaf permite generarea de pagini web pe partea de server și poate fi integrat cu ușurință în arhitectura MVC (Model-View-Controller), fiind destinat în special pentru proiectele de tip server-side rendering.

Intro

Thymeleaf are scopul principal de a oferi un mod elegant și foarte ușor de întreținut de a crea șabloane. Pentru a realiza acest lucru se bazează pe conceptul de Șabloane Naturale pentru a-și insera logica în fișierele șablon într-un mod în care să nu afecteze șablonul de la a fi folosit ca un prototip de design. Acest lucru îmbunătățește comunicarea designului și reduce decalajul dintre echipele de proiectare și dezvoltare.

Scop si concept

Thymeleaf a fost lansat pentru prima dată în 2011 de ****Daniel Fernández****, având ca scop principal să ofere un motor de template Java care să fie ușor de utilizat și să producă șabloane HTML care să poată fi vizualizate direct în browsere, fără a necesita preprocesare. Inițial, a fost conceput pentru a înlocui motoarele de template existente, cum ar fi JSP (JavaServer Pages), oferind o abordare mai modernă și mai apropiată de standardele HTML.

Popularitatea Thymeleaf a crescut odată cu adoptarea pe scară largă a framework-ului Spring Boot, datorită integrării sale native și a ușurinței cu care dezvoltatorii puteau construi interfețe de utilizator dinamice. De-a lungul anilor, Thymeleaf a evoluat, adăugând suport pentru noi funcționalități, precum manipularea XML, reutilizarea fragmentelor și suportul pentru diverse expresii logice și condiționale.

Istorie Thymeleaf

Thymeleaf se remarcă printr-o sintaxă care se aseamănă mult cu HTML-ul, ceea ce îl face ușor de citit și întreținut. Se integrează fără probleme cu Spring Framework, oferind suport nativ pentru Spring MVC, iar această compatibilitate îl face o alegere populară în proiectele bazate pe Java. De asemenea, poate fi utilizat atât în mod standalone, cât și în aplicații web care rulează pe partea de server. O altă caracteristică importantă este capacitatea de a reutiliza fragmente de template, ceea ce simplifică organizarea și modularitatea codului. În plus, Thymeleaf permite utilizarea expresiilor logice, cum ar fi condiții și bucle, direct în șabloanele HTML.

Caracteristici

Avantajele principale ale Thymeleaf includ simplitatea și ușurința de utilizare, mai ales pentru cei care sunt deja familiarizați cu HTML. Datorită integrării native cu Spring Boot, dezvoltarea aplicațiilor web devine mai rapidă și mai eficientă. Thymeleaf facilitează generarea dinamică a paginilor pe partea de server și oferă posibilitatea de a reutiliza codul prin fragmentarea șabloanelor, ceea ce contribuie la o mai bună organizare a proiectului. În același timp, codul produs este ușor de întreținut și scalabil, fiind potrivit și pentru aplicații complexe.

Avantaje

Thymeleaf este compatibil cu diverse tipuri de conținut, inclusiv HTML5, XML și JavaScript, ceea ce îl face potrivit pentru o gamă largă de proiecte. De asemenea, este extensibil, permițând adăugarea de funcționalități personalizate prin plugin-uri și dialecte proprii, ceea ce îl face extrem de flexibil și adaptabil la cerințele complexe ale proiectelor mari.

Compatibilitate si extensibilitate

Thymeleaf este un motor de template server-side, potrivit pentru aplicații web tradiționale bazate pe Java și Spring Boot, unde logica de prezentare este gestionată pe server. Este simplu de utilizat, ideal pentru proiecte mici și medii, dar mai puțin eficient pentru aplicații complexe sau interactive.

În contrast, Angular este un framework client-side creat pentru aplicații single-page (SPA), care gestionează logica și datele pe client, oferind performanță ridicată și interactivitate. Este potrivit pentru aplicații mari, scalabile și moderne, dar necesită o curbă de învățare mai abruptă și utilizarea API-urilor pentru comunicarea cu backend-ul.

Alegerea între cele două depinde de cerințele proiectului: Thymeleaf pentru simplitate și integrare cu Java, Angular pentru aplicații dinamice și scalabilitate pe client.

Thymeleaf vs Angular

Thymeleaf este o alegere excelentă pentru aplicațiile web care rulează pe partea de server și care sunt dezvoltate în Java, în special cele care folosesc framework-ul Spring Boot. Este ideal pentru proiectele în care colaborarea între dezvoltatori și designeri este esențială, deoarece permite crearea de șabloane care pot fi vizualizate direct în browser fără preprocesare. De asemenea, este potrivit pentru aplicațiile care necesită generarea dinamică de conținut și manipularea avansată a formularelor.

Cand sa folosesti Thymeleaf

Cu toate acestea, Thymeleaf are și câteva limitări. Nu este la fel de performant pentru aplicațiile single-page (SPA), unde framework-uri precum React sau Angular sunt mai potrivite. De asemenea, nu este recomandat pentru aplicațiile web care se bazează exclusiv pe logica de prezentare pe partea de client. În proiectele extrem de mari, unde există o complexitate ridicată în gestionarea logicii de prezentare, utilizarea sa poate deveni mai puțin eficientă.

Limitari

Thymeleaf este un motor de template puternic și flexibil, ideal pentru aplicațiile Java server-side, în special cele construite cu Spring Boot. Datorită sintaxei sale naturale și integrării perfecte cu Spring, este o alegere excelentă pentru crearea de interfețe web dinamice și gestionarea formularelor. Deși nu este optim pentru aplicațiile single-page (SPA) și poate deveni mai puțin eficient în proiectele mari cu logică complexă de prezentare, Thymeleaf rămâne o soluție robustă pentru majoritatea aplicațiilor tradiționale Java. Sursele oficiale și tutorialele online oferă suport suplimentar pentru învățarea și utilizarea eficientă a acestui motor de template.

Concluzie

2.6 **BOOTSTRAP 5**

Bootstrap 5 este cea mai recentă versiune a popularului framework front-end Bootstrap. Acesta este utilizat pentru a dezvolta interfețe web responsive și moderne cu mai puțin efort și cod personalizat. Versiunea 5 aduce îmbunătățiri semnificative, inclusiv renunțarea la dependența de jQuery, un design mai performant și o flexibilitate crescută.

Intro

Bootstrap 5 este cea mai recentă versiune a framework-ului Bootstrap, lansată oficial în mai 2021, continuând tradiția acestuia de a simplifica dezvoltarea web. Bootstrap a fost creat inițial de dezvoltatorii de la Twitter în 2011, ca un framework intern pentru a uniformiza interfețele. Bootstrap 1.0 a introdus sistemul de grilă și un set de stiluri predefinite pentru crearea de interfețe consistente. În 2012, Bootstrap 2 a adus design-ul responsive, făcând paginile web să se adapteze automat la diferite dimensiuni de ecran.

În 2013, Bootstrap 3 s-a concentrat pe abordarea mobile-first, oferind un sistem de grilă mai flexibil, ideal pentru dispozitivele mobile. Lansarea din 2018, Bootstrap 4, a făcut tranziția la Sass, un preprocesor CSS avansat, și a introdus variabile CSS și Flexbox pentru un control mai mare asupra layout-urilor. În cele din urmă, Bootstrap 5, lansat în 2021, a eliminat dependența de jQuery, a adăugat componente moderne precum offcanvas și a optimizat performanța utilizând variabile CSS pentru o personalizare mai simplă.

Istorie Bootstrap

Bootstrap 5 aduce o serie de îmbunătățiri notabile care îl fac mai eficient și mai flexibil pentru dezvoltarea interfețelor web. În primul rând, această versiune renunță la dependența de jQuery, ceea ce reduce dimensiunea proiectelor și crește performanța aplicațiilor. Sistemul de grilă a fost actualizat pentru a include cinci puncte de întrerupere, oferind o mai mare flexibilitate în crearea designurilor responsive. De asemenea, sunt introduse noi componente, precum offcanvas, iar componentele existente au fost simplificate și optimizate.

Un alt aspect important este utilizarea variabilelor CSS, care permite personalizarea temelor și a culorilor într-un mod mai simplu și mai eficient. Performanța generală a fost îmbunătățită, cu un consum mai redus de resurse ale browserului. În plus, documentația a fost extinsă și îmbunătățită, fiind mai ușor de utilizat pentru dezvoltatori, indiferent de nivelul lor de experiență.

Avantaje

Bootstrap 5 reprezintă un pas important în evoluția dezvoltării web, oferind un framework modern, performant și ușor de personalizat. Prin renunțarea la jQuery, introducerea variabilelor CSS și a unor componente noi precum offcanvas, această versiune răspunde cerințelor actuale ale dezvoltatorilor. Sistemul de grilă flexibil, tematicile personalizabile și suportul îmbunătățit pentru tehnologii moderne îl fac o alegere excelentă pentru proiecte responsive și elegante. Cu o documentație bine organizată și resurse disponibile, Bootstrap 5 este un instrument esențial pentru crearea de interfețe web rapide și atractive.

Concluzie

3 CERINȚA ȘI SPECIFICAȚIILE PROIECTULUI

Proiectarea aplicației web pentru gestionarea planurilor de învățământ reflectă un set bine definit de cerințe funcționale, nefuncționale și tehnice, fiecare având un rol esențial în realizarea unei soluții robuste, eficiente și scalabile.

Introducere

3.1 CERINȚE FUNCȚIONALE

Aplicația trebuie să permită utilizatorilor autentificarea și autorizarea, oferind acces diferențiat în funcție de rolurile definite: administrator, cadru didactic și student. Administratorii vor avea posibilitatea de a importa planuri de învățământ din fișiere Excel, aceste planuri fiind asociate generațiilor și anilor academici corespunzători. De asemenea, ei vor putea vizualiza, edita sau șterge aceste planuri.

Pentru gestionarea fișelor de disciplină, administratorii trebuie să poată genera automat fișe pe baza șabloanelor Word existente, să asocieze fișele cu discipline și cadre didactice, precum și să copieze fișele disciplinelor în generații noi la începutul fiecărui an universitar. Tot administratorii vor avea funcționalitatea de a alocă cadre didactice la discipline specifice.

Cadrele didactice vor putea să se autentifice în aplicație, să vizualizeze fișele de disciplină asociate lor și să încarce fișe noi. În același timp, aplicația va permite studenților să acceseze fișele de disciplină corespunzătoare generației, ciclului de studii (licență sau master) și anului lor academic, utilizând filtre care facilitează accesul rapid la informațiile relevante.

Cerințele funcționale se concentrează pe oferirea unui set clar de funcționalități destinate diferitelor categorii de utilizatori (administratori, cadre didactice, studenți), având ca scop principal automatizarea proceselor administrative. Acestea includ importul fișierelor Excel, gestionarea fișelor de disciplină și asigurarea unei interfețe intuitive pentru utilizatori.

Concluzie c. funcționale

3.2 CERINȚE NEFUNCȚIONALE

Aplicația trebuie să ofere performanță ridicată, cu timpi de răspuns de sub două secunde pentru cele mai frecvente operațiuni, inclusiv procesarea fișierelor Excel și Word de dimensiuni mari. Securitatea este o cerință esențială, fiind necesară protecția datelor utilizatorilor prin criptarea parolilor și prevenirea atacurilor de tip SQL Injection și Cross-Site Scripting. De asemenea, aplicația trebuie să fie scalabilă, astfel încât să poată gestiona volume mai mari de date și să răspundă cerințelor viitoare.

Compatibilitatea aplicației cu toate browserele moderne (Chrome, Firefox, Edge) și dispozitivele (desktop, tabletă, mobil) este esențială. Interfața trebuie să fie intuitivă, astfel încât utilizatorii să înțeleagă rapid funcționalitățile oferite. Gestionarea erorilor trebuie realizată eficient, prin mesaje clare pentru utilizatori și loguri detaliate pentru administratori, pentru a facilita identificarea problemelor.

Cerințele nefuncționale asigură calitatea și fiabilitatea aplicației prin standarde de performanță, securitate, scalabilitate și compatibilitate. Astfel, aplicația trebuie să funcționeze eficient chiar și în scenarii complexe, să protejeze datele utilizatorilor și să fie accesibilă pe multiple platforme și browsere moderne.

Concluzie c. nefuncționale

3.3 CERINȚE TEHNICE

Implementarea aplicației trebuie să fie realizată folosind Java pentru backend, cu ajutorul framework-ului Spring Boot, care asigură gestionarea serviciilor, autentificării și API-urilor REST. Manipularea fișierelor Excel și Word va fi realizată prin biblioteca Apache POI, iar baza de date utilizată va fi MySQL, configurată pentru performanță și integritate.

Frontend-ul poate fi dezvoltat fie cu Thymeleaf, pentru o integrare directă cu backend-ul. Aplicația va respecta arhitectura Model-View-Controller (MVC) și va integra API-uri REST pentru a permite comunicarea între componentele frontend și backend. De asemenea, testarea va fi realizată printr-o combinație de teste unitare, de integrare și de performanță, pentru a garanta funcționalitatea, fiabilitatea și calitatea produsului final.

Cerințele tehnice oferă cadrul tehnologic și arhitectural pentru implementare. Alegerea unui stack tehnologic solid (Java, Spring Boot, MySQL, Apache POI, Thymeleaf) garantează o infrastructură flexibilă și ușor de întreținut, în timp ce arhitectura MVC și utilizarea standardelor moderne (REST API, Spring Security) asigură o dezvoltare modulară și bine structurată.

Concluzie c. tehnice

Prin îmbinarea acestor cerințe, aplicația urmărește nu doar satisfacerea nevoilor curente ale utilizatorilor, ci și crearea unei soluții sustenabile și extensibile, capabile să se adapteze viitoarelor cerințe și schimbări din mediul academic. Acest echilibru între funcționalitate, performanță și tehnologie reprezintă fundamentul pentru succesul proiectului.

Concluzie toate cerințele

3.4 FUNCȚIONALITATEA PROIECTULUI

Proiectul propus reprezintă o aplicație web complexă, structurată pe module care răspund nevoilor diferitelor categorii de utilizatori.

Administratorul are acces la o serie de funcționalități care îi permit gestionarea datelor academice. Acesta se poate autentifica pentru a accesa aplicația. Poate importa planuri de învățământ din fișiere Excel direct în baza de date și poate inițializa fișe de disciplină utilizând fișiere Word existente. Totodată, are posibilitatea de a aloca cadre didactice pentru disciplinele din planurile de învățământ și de a copia fișele de disciplină recente într-o generație nouă atunci când anul universitar se schimbă.

Profesorii pot utiliza aplicația pentru a se autentifica și accesa fișele de disciplină aferente cursurilor pe care le predau. De asemenea, aceștia pot încărca fișe de disciplină noi pentru actualizarea sau completarea datelor aferente materiilor lor.

Studentii pot vizualiza fișele de disciplină disponibile, acestea fiind filtrate în funcție de generație (de exemplu, 2020–2024 sau 2021–2025), ciclul de studii (Licență sau Master) și anul de studiu (Anul 1, Anul 2 etc.).

Proiectul include un model de date relațional SQL care reflectă structura și relațiile planurilor de învățământ preluate din fișiere Excel. În același timp, este implementat un sistem de stocare a fișierelor Word asociate fișelor de disciplină, permițând încărcarea, salvarea și accesarea acestora. Generațiile de planuri de învățământ pot fi gestionate cu ușurință, iar aplicația dispune de o interfață de utilizator intuitivă, dezvoltată folosind tehnologiile Thymeleaf, pentru o utilizare eficientă și ușoară.

Funcționalități tehnice generale

Aceste funcționalități sunt integrate pentru a răspunde nevoilor administratorilor, profesorilor și studenților, oferind o soluție completă pentru gestionarea planurilor de învățământ și a fișelor de disciplină.

Concluzie funcționalitate

3.5 ANALIZA DE RISC

Analiza de risc a proiectului implică identificarea și gestionarea factorilor care pot afecta dezvoltarea aplicației. Printre riscurile principale se numără cele tehnologice, cum ar fi problemele de integrare între componente, performanța redusă a aplicației sau defecțiunile hardware. De asemenea, există riscuri legate de cerințe, cum ar fi ambiguitatea specificațiilor sau modificările neprevăzute în funcționalitățile solicitate. Alte riscuri pot apărea în etapa de dezvoltare, incluzând întârzieri în implementare și lipsa resurselor tehnice sau umane necesare. În plus, securitatea poate fi amenințată de breșe sau pierderi de date, iar operațional, aplicația poate întâmpina dificultăți din cauza fișierelor importate incorect sau a dificultăților de utilizare întâmpinate de utilizatori.

Pentru a gestiona aceste riscuri, ele sunt evaluate în funcție de probabilitatea de apariție și de impactul asupra proiectului, stabilindu-se astfel priorități de intervenție. Strategiile de gestionare a riscurilor includ prevenirea prin planificare timpurie și validarea automată a datelor, minimizarea impactului prin testare frecventă și audituri periodice și acceptarea riscurilor inevitabile, pregătind soluții pentru gestionarea lor.

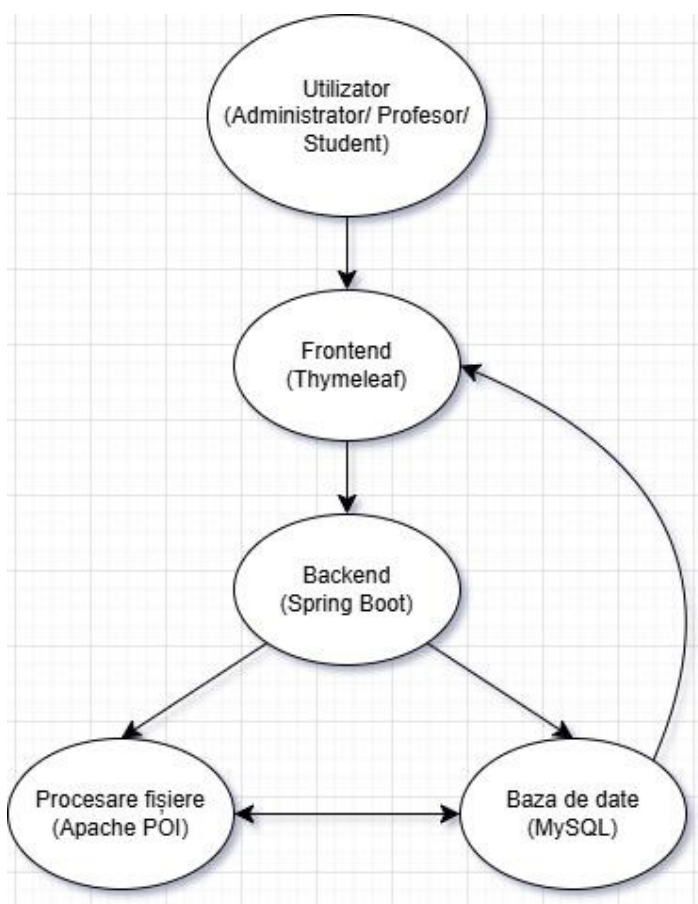
Monitorizarea riscurilor este un proces continuu, care presupune revizuirea regulată a listei de riscuri și adaptarea planurilor de răspuns în funcție de noile provocări.

Prin identificarea timpurie a riscurilor și implementarea unor măsuri adecvate de gestionare, analiza de risc contribuie la dezvoltarea unei aplicații robuste, securizate și eficiente, care să respecte cerințele și să fie livrată în termenii stabiliți. **Concluzie**

4 PROIECTARE

4.1 SCHEMA BLOC

Schema arhitecturală a aplicației web ilustrează structura sa modulară și fluxurile de date între componentele principale. La nivel de utilizatori, aplicația se adresează a trei categorii distincte: administrator, cadru didactic și student. Aceștia interacționează cu aplicația printr-o interfață web, accesibilă și intuitivă.



Frontend-ul aplicației, realizat cu ajutorul Thymeleaf sau Angular, reprezintă nivelul de prezentare. Acesta afișează informațiile relevante pentru utilizatori și colectează datele introduse de aceștia. Toate cererile utilizatorilor sunt procesate de backend-ul aplicației, dezvoltat în Spring Boot. Backend-ul implementează logica aplicației și acționează ca intermediar între frontend și componentele backend-ului, cum ar fi baza de date și modulul de procesare a fișierelor.

Pentru stocarea informațiilor, aplicația utilizează o bază de date relațională MySQL. Aceasta găzduiește date despre planurile de învățământ, fișele de disciplină, utilizatori și alte informații administrative. Operarea bazei de date este asigurată de backend, care efectuează interogări și actualizări în funcție de cerințele utilizatorilor.

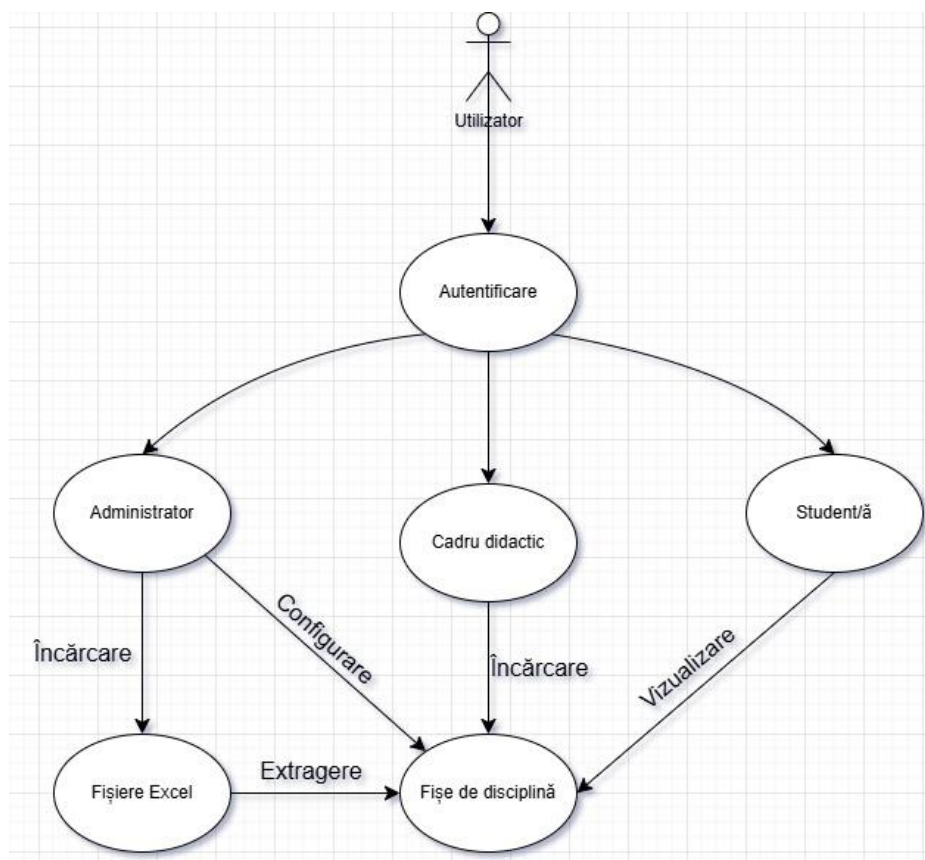
Un modul special, bazat pe Apache POI, gestionează importul și exportul fișierelor Excel și Word. Acesta permite integrarea datelor din fișierele educaționale existente în aplicație și exportul lor în format compatibil.

Fluxul de date în aplicație urmează un traseu clar. Utilizatorii trimit cereri către frontend, care le direcționează către backend pentru procesare. Backend-ul, în funcție de natura cererii, poate accesa baza de date pentru stocare sau retragerea informațiilor ori modulul Apache POI pentru manipularea fișierelor. Rezultatul procesării este transmis înapoi utilizatorilor prin intermediul frontend-ului.

Această arhitectură, bazată pe separarea responsabilităților între componente, asigură scalabilitatea și întreținerea facilă a aplicației, oferind în același timp o experiență optimă utilizatorilor finali.

4.2 DIAGRAMA DE CLASE, SECVENTA SI STARE

Schema UML de clasă pentru aplicația web descrie structura statică a sistemului, evidențiind principalele clase și relațiile dintre ele.



Clasa **Administrator** are ca atribute un identificator unic (`id`), numele administratorului (`nume`) și adresa de e-mail (`email`). Aceasta include metode precum

``importPlanuri()``, care permite importul planurilor de învățământ din fișiere Excel, ``initializeazaFiseDisciplina()``, pentru inițializarea fișelor de disciplină din fișiere Word, și ``alocaCadreDidactice()``, care atribuie cadre didactice disciplinelor.

Clasa ***PlanInvatamant*** reprezintă planurile de învățământ și conține atribute precum un identificator unic (``id``), generația asociată planului (de exemplu, 2020–2024) și o listă de ani academici (``aniStudiu``, cum ar fi anul 1: 2020–2021). Această clasă dispune de metoda ``copierePlanNou()``, care permite crearea unei copii a planului pentru o generație nouă.

Clasa ***FisaDisciplina*** descrie detaliile unei discipline și include atributele ``id`` (identificator unic), ``denumire`` (denumirea disciplinei) și ``profesor`` (legătura către cadrul didactic responsabil). Metoda principală a clasei, ``incarcaFisier()``, permite încărcarea fișierelor asociate fișelor de disciplină.

Clasa ***CadruDidactic*** reprezintă profesorii și are atribute precum ``id`` (identificator unic) și ``nume`` (numele profesorului). Metoda sa principală, ``vizualizeazaFiseDisciplina()``, permite afișarea fișelor de disciplină asociate profesorului.

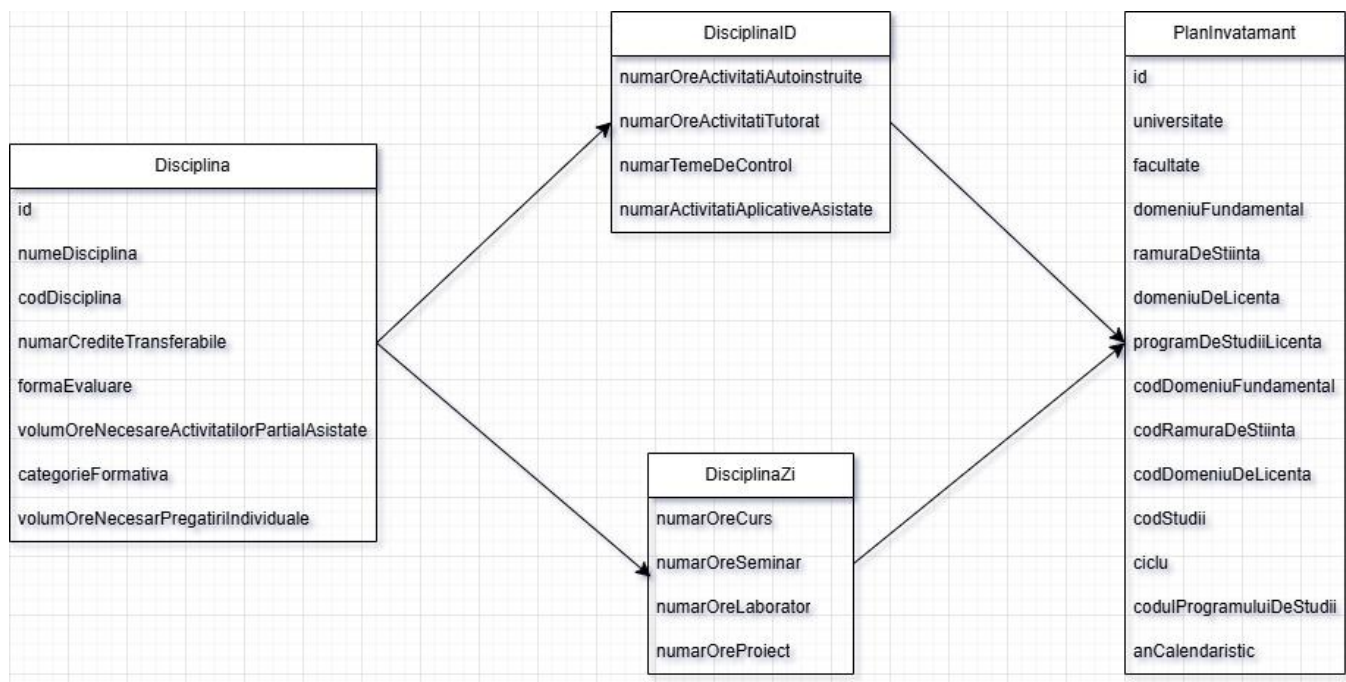
Clasa ***Student*** descrie studenții și conține atributele ``id`` (identificator unic) și ``nume`` (numele studentului). Metoda ``vizualizeazaPlanuri()`` permite studenților să vizualizeze planurile de învățământ, filtrate după generație, ciclul de studii (licență sau master) și an de studiu.

Relațiile dintre aceste clase sunt bine definite. Clasa *Administrator* este asociată cu *PlanInvatamant*, facilitând gestionarea și importul planurilor de învățământ. Clasa *PlanInvatamant* este într-o relație de compoziție cu *FisaDisciplina*, ceea ce reflectă faptul că un plan de învățământ conține mai multe fișe de disciplină. De asemenea, *FisaDisciplina* este asociată cu *CadruDidactic*, pentru a indica alocarea unui profesor unei discipline. În plus, clasa *Student* este asociată cu *PlanInvatamant*, permițând accesul studenților la planurile de învățământ.

Această structură logică evidențiază principalele componente ale aplicației și modul în care acestea interacționează pentru a satisface cerințele proiectului.

4.3 SCHEMA BAZEI DE DATE

Schema bazei de date este proiectată pentru a susține gestionarea planurilor de învățământ într-o aplicație web, organizând datele academice într-un mod structurat și eficient. Aceasta include mai multe tabele interconectate, fiecare având un rol bine definit în cadrul sistemului.



În centrul schemei se află tabelul **Generatie**, care stochează informații despre generațiile academice, inclusiv identificatorul unic, denumirea și intervalul de ani, precum 2020–2024 sau 2021–2025. Generațiile sunt baza pentru organizarea planurilor de învățământ.

Tabelul **PlanInvatamant** conține detalii despre planurile de învățământ, precum denumirea planului și anul de studiu asociat. Fiecare plan este legat de o anumită generație, oferind o perspectivă temporală clară asupra structurii curriculare.

Disciplina este reprezentată în schema bazei de date prin tabelul **Disciplina**, care include informații esențiale precum denumirea disciplinei, numărul de credite alocate și tipul acesteia (obligatorie sau opțională). Fiecare disciplină este asociată unui plan specific, reflectând detaliile curriculare.

Fișele de disciplină sunt gestionate prin tabelul **FisaDisciplina**, care stochează locația fișierelor Word asociate și data ultimei actualizări. Această structură permite accesarea și actualizarea facilă a documentelor relevante pentru fiecare disciplină.

Tabelul **CadruDidactic** conține informații despre cadrele didactice, incluzând numele, prenumele, emailul și parola acestora. Acest tabel este utilizat atât pentru autentificarea cadrelor didactice, cât și pentru gestionarea relației lor cu disciplinele predate.

În final, tabelul **Student** stochează detalii despre studenți, precum numele, prenumele și generația din care fac parte. Acesta sprijină funcționalitățile care permit filtrarea fișelor de disciplină în funcție de generație, ciclu de studii și an de studiu.

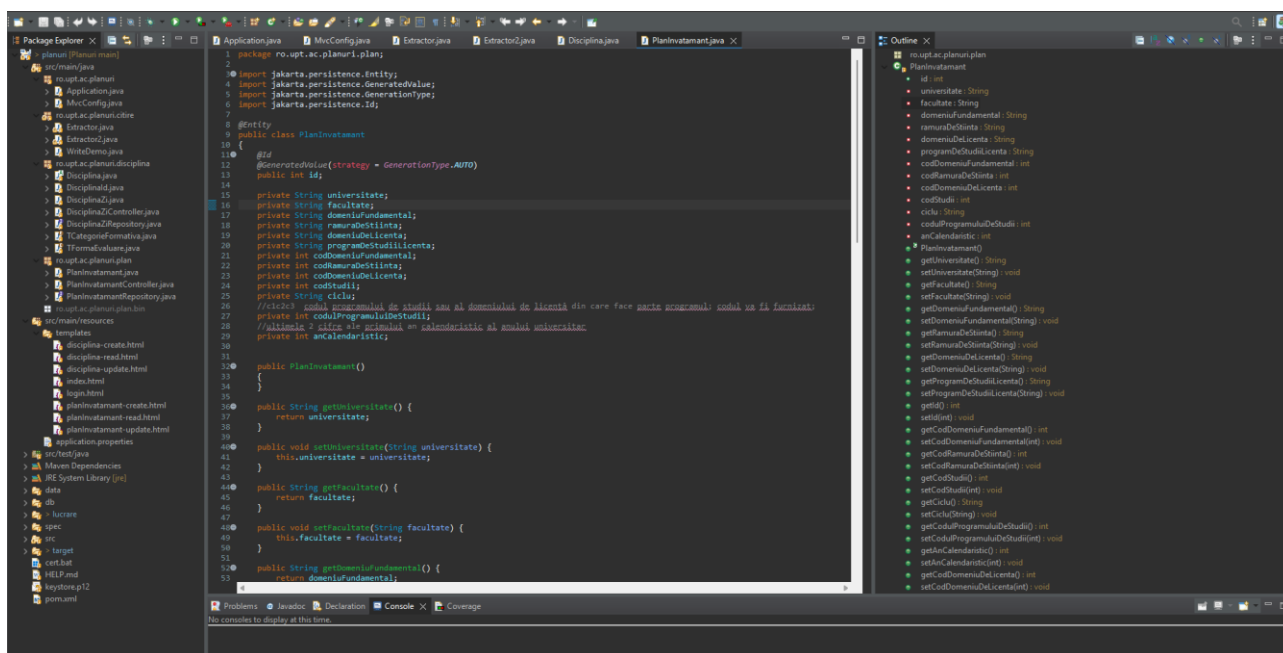
Această schema susține funcționalități precum importul planurilor de învățământ din fișiere Excel, stocarea și gestionarea fișelor de disciplină, autentificarea utilizatorilor și atribuirea cadrelor didactice la discipline. În plus, schema permite vizualizarea fișelor de disciplină de către cadrele didactice și studenți, utilizând filtre dinamice bazate pe generație și an de studiu. Datorită structurii flexibile și scalabile, schema poate fi extinsă pentru a integra noi funcționalități în viitor.

5 IMPLEMENTAREA

5.1 DESCRIERE CLASE

5.1.1 CLASA PlanInvatamant

Implementarea claselor într-o aplicație Java implică definirea atributelor, constructorilor și metodelor care modelează entitățile dintr-un sistem. În cazul clasei **PlanInvatamant**, aceasta este o entitate JPA, utilizată pentru maparea datelor între obiectele Java și o tabelă din baza de date.



Codul reprezintă o clasă Java numită 'PlanInvatamant', care definește o entitate JPA utilizată pentru maparea unui obiect Java la o tabelă din baza de date. Prin intermediul anotării '@Entity', clasa este declarată ca fiind o entitate JPA, iar implicit, numele tabelii asociate va fi același cu numele clasei. Dacă este necesar, numele poate fi modificat folosind '@Table'.

Atributul ``id`` este definit ca fiind cheia primară a entității, utilizând anotarea ``@Id``. Generarea automată a valorii acestuia este specificată prin ``@GeneratedValue(strategy = GenerationType.AUTO)``, ceea ce înseamnă că baza de date va gestiona valorile unice ale acestui câmp.

Clasa conține mai multe atribute, cum ar fi ``universitate``, ``facultate``, ``domeniuFundamental``, ``ramuraDeStiinta`` și altele, care descriu caracteristicile unui plan de învățământ. Acestea vor fi mapate la coloanele tabelului corespunzătoare din baza de date. Tipurile de date utilizate includ ``int`` pentru valorile numerice și ``String`` pentru text.

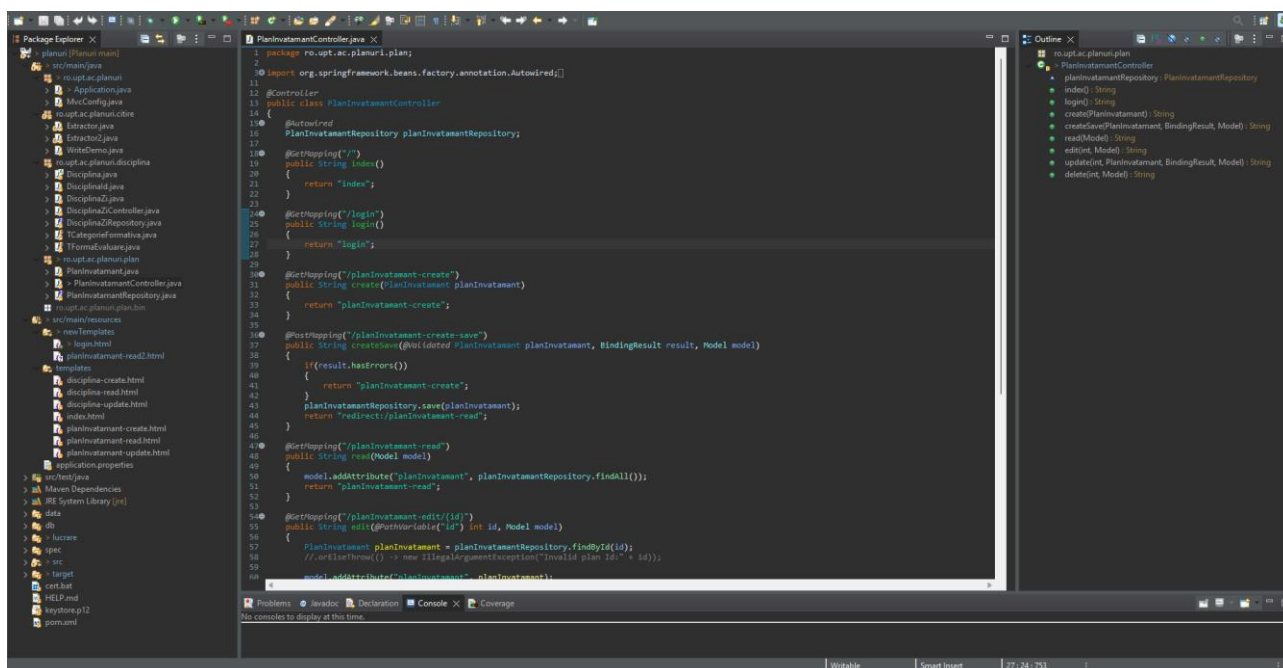
Constructorul implicit, definit prin ``public PlanInvatamant()``, este necesar pentru ca JPA să poată crea instanțe ale clasei. Metodele getter și setter sunt incluse pentru a permite accesul și modificarea valorilor atributelor. De exemplu, metoda ``getUniversitate()`` returnează valoarea atributului ``universitate``, iar metoda ``setUniversitate(String universitate)`` permite atribuirea unei noi valori acestui atribut.

Fiecare atribut are o semnificație specifică: ``universitate`` și ``facultate`` reprezintă numele instituției și al facultății, în timp ce ``domeniuFundamental``, ``ramuraDeStiinta`` și ``domeniuDeLicenta`` oferă informații academice detaliate. Atributul ``programDeStudiiLicenta`` indică numele programului de studii, iar diversele coduri numerice (``codDomeniuFundamental``, ``codRamuraDeStiinta``, etc.) sunt identificatori pentru categorii specifice de informații. De asemenea, atributul ``ciclu`` indică nivelul de studiu (de exemplu, Licență sau Master), iar ``anCalendaristic`` reține ultimele două cifre ale primului an universitar.

Într-o aplicație Spring Boot, această clasă poate fi utilizată pentru a salva, actualiza, citi sau șterge informații din baza de date prin intermediul unui repository. De exemplu, un obiect ``PlanInvatamant`` poate fi populat cu date, iar apoi salvat în baza de date folosind metoda ``save()`` a unui repository.

5.1.2 CLASA PlanInvatamantController

Acest cod definește un controller Spring Boot, care gestionează operațiile CRUD pentru un "Plan de Învățământ". Fiind o clasă anotată cu `@Controller`, ea este responsabilă pentru procesarea cererilor HTTP și afișarea paginilor HTML corespunzătoare. Controller-ul utilizează injecția de dependențe prin `@Autowired` pentru a accesa repository-ul `PlanInvatamantRepository`, care interacționează direct cu baza de date.



Controller-ul conține mai multe metode, fiecare având un rol specific. Prima parte include metodele index() și login(), care afișează paginile principale ale aplicației: o pagină de start (index.html) și o pagină pentru autentificare (login.html).

Operațiile legate de crearea unui plan de învățământ sunt implementate în două metode. Metoda create() afișează un formular gol utilizând pagina HTML planInvatamant-create.html. După ce utilizatorul completează formularul și îl trimite, metoda createSave() validează datele primite. Dacă există erori de validare, utilizatorul este redirecționat înapoi la formular. În caz contrar, datele sunt salvate în baza de date utilizând metoda save() din repository, iar utilizatorul este redirecționat la lista planurilor existente.

Pentru a afișa toate planurile de învățământ, metoda read() accesează repository-ul și obține toate intrările din baza de date. Acestea sunt transmise paginii HTML planInvatamant-read.html prin intermediul unui obiect Model, care facilitează afișarea lor într-un format adecvat.

Editarea unui plan existent este gestionată de metodele edit() și update(). Metoda edit() primește un parametru id din URL, găsește planul corespunzător în baza de date și transmite datele către formularul de editare afișat în planInvatamant-update.html. După ce utilizatorul finalizează editările, metoda update() validează și salvează modificările. Dacă există erori, formularul este reafiat pentru corectarea acestora; altfel, datele sunt actualizate în baza de date.

În cazul ștergerii unui plan, metoda delete() primește un id, caută intrarea corespunzătoare în baza de date și o elimină utilizând metoda delete() din repository. După ștergere, utilizatorul este redirecționat la lista actualizată de planuri.

Codul presupune existența unei entități **PlanInvatamant**, a unui repository **PlanInvatamantRepository** și a mai multor pagini HTML pentru fiecare operație. Gestionarea excepțiilor pentru situații precum căutarea unui plan inexistent nu este complet implementată, lucru indicat de secțiunile comentate din metodele `edit()` și `delete()`.

5.1.3 CLASA **PlanInvatamantRepository**

Codul definește o interfață pentru gestionarea entității **PlanInvatamant** utilizând Spring Data JPA. Acesta face parte din stratul de acces la date și facilitează interacțiunea cu baza de date pentru operațiuni standard (CRUD).

Interfața este declarată în pachetul **ro.upt.ac.planuri.plan** și este marcată cu anotarea **@Repository**, ceea ce permite Spring-ului să o detecteze automat și să genereze o implementare la rulare.

```
PlanInvatamantRepository.java X
1 package ro.upt.ac.planuri.plan;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6 @Repository
7 public interface PlanInvatamantRepository extends JpaRepository<PlanInvatamant,Integer>
8 {
9     PlanInvatamant findById(int id);
10 }
11
12
```

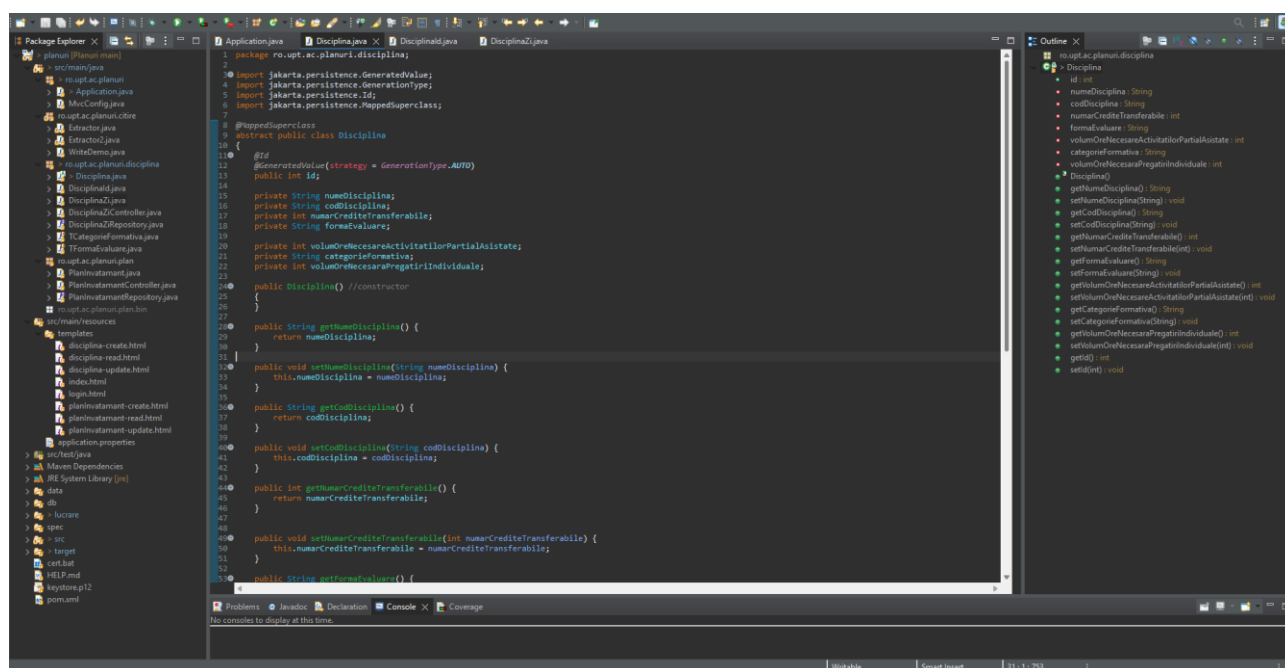
Interfața extinde **JpaRepository<PlanInvatamant, Integer>**, ceea ce oferă acces la o gamă largă de metode predefinite, cum ar fi `findAll()` pentru obținerea tuturor entităților, `findById(Integer id)` pentru căutarea unei entități după ID, `save()` pentru salvarea sau actualizarea unei entități și `deleteById()` pentru ștergerea unei entități după ID.

Pe lângă aceste metode predefinite, interfața definește o metodă personalizată, **`findById(int id)`**, care permite găsirea unei entități **PlanInvatamant** pe baza unui identificator numeric (`id`) și returnează direct entitatea în locul unui `Optional<PlanInvatamant>`, simplificând utilizarea.

Acest repository asigură un mod convenabil și eficient de a interacționa cu baza de date fără a scrie manual cod SQL.

5.1.4 CLASA Disciplina

Clasa **Disciplina** reprezintă un model abstract utilizat pentru a defini structura de bază a entităților ce descriu disciplinele academice într-o aplicație Java care utilizează Jakarta Persistence API (JPA). Aceasta include atribute esențiale, precum numele, codul, numărul de credite și detalii despre volum de ore, fiind concepută pentru a fi extinsă de alte clase concrete. Prin utilizarea adnotării `@MappedSuperclass`, clasa oferă o bază comună pentru maparea și gestionarea entităților în baza de date, promovând reutilizarea și organizarea eficientă a codului.



Primul atribut, `id`, este marcat cu `@Id` pentru a fi utilizat ca identificator unic al entității. Adnotarea `@GeneratedValue(strategy = GenerationType.AUTO)` specifică faptul că valoarea acestuia este generată automat de baza de date. Atributul are tipul `int`.

Clasa conține mai multe atribute care descriu proprietățile unei discipline, cum ar fi numele disciplinei (`numeDisciplina`), codul unic de identificare (`codDisciplina`), numărul de credite ECTS asociate (`numarCrediteTransferabile`), și forma de evaluare (`formaEvaluare`, care poate fi examen, colocviu etc.). De asemenea, sunt incluse detalii privind volumul de ore necesare pentru activitățile parțial asistate (`volumOreNecesareActivitatilorPartialAsistate`) și pentru pregătirea individuală (`volumOreNecesarePregatiriiIndividuale`), precum și categoria formativă a disciplinei (`categoriaFormativa`, de exemplu: fundamentală, de specialitate).

Constructorul implicit este gol, ceea ce permite JPA să creeze instanțe ale clasei fără a specifica inițial valori pentru atribute. În plus, clasa include metode getter și setter

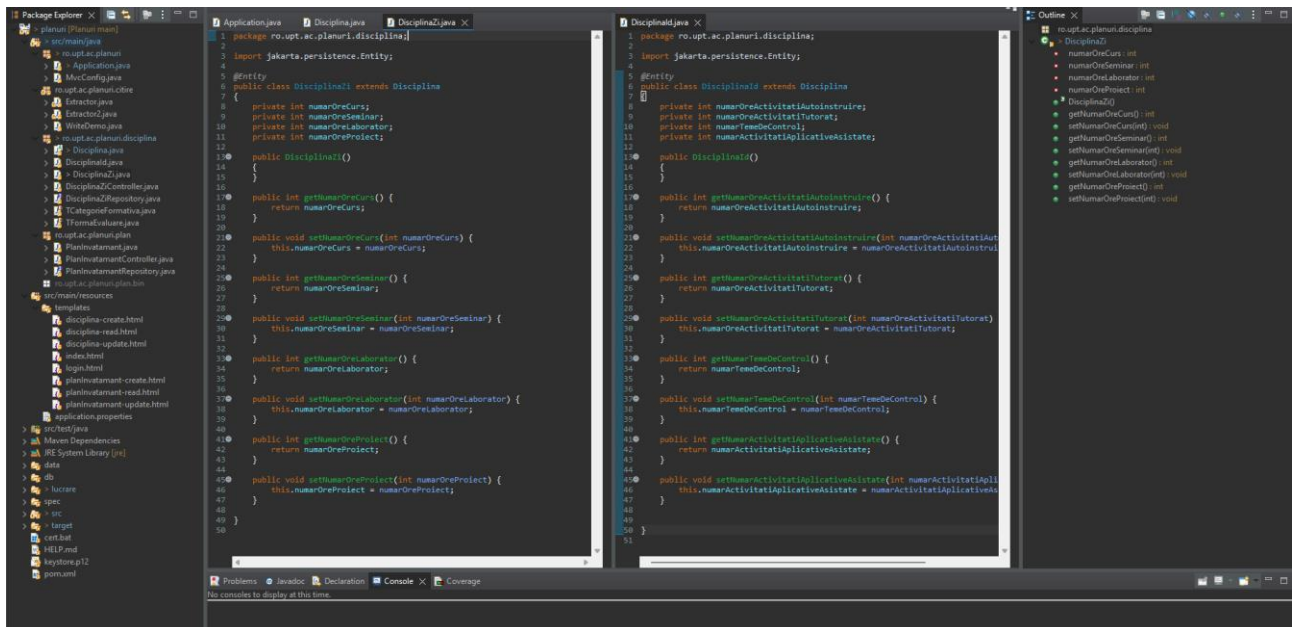
pentru fiecare atribut, asigurând accesul și modificarea acestora în conformitate cu principiile de incapsulare. Metodele getter returnează valorile atributelor, iar metodele setter permit atribuirea unor valori noi.

Fiind o clasă abstractă, *Disciplina* nu poate fi instanțiată direct. Ea este destinată să fie extinsă de alte clase, cum ar fi *DisciplinaTehnica*, care poate adăuga atribute sau metode specifice pentru un anumit tip de disciplină.

Scopul principal al clasei este de a furniza o structură comună și reutilizabilă pentru entitățile care reprezintă disciplinele academice, facilitând maparea acestora în baza de date și promovând extensibilitatea și reutilizarea codului.

5.1.5 CLASA *DisciplinaZi* și *DisciplinaId*

Ambele clase, ***DisciplinaZi*** și ***DisciplinaId***, extind clasa de bază ***Disciplina***, moștenind comportamentele acesteia, dar adăugând propriile caracteristici pentru a reflecta diferite tipuri de discipline. Ele sunt marcate cu anotarea `@Entity`, ceea ce le face entități JPA și implică faptul că vor fi mapate la tabele distincte în baza de date. Ambele clase includ un constructor implicit (fără parametri) și metode `get` și `set` pentru fiecare atribut, ceea ce permite manipularea facilă a datelor.



DisciplinaZi este concepută pentru a descrie activități academice caracteristice sistemului de învățământ de zi. Aceasta include numărul de ore pentru cursuri, seminare, laboratoare și proiecte, reflectând accentul pus pe interacțiunea față în față și activitățile sincronizate.

Pe de altă parte, **Disciplinald** este adaptată pentru programele de învățământ la distanță, unde metodele de predare și învățare sunt mai independente. Aceasta include numărul de ore dedicate activităților de autoinstruire și tutorat, numărul de teme de control și numărul de activități aplicative asistate, reflectând nevoia de a sprijini învățarea individuală cu resurse și asistență adecvate.

Diferențele dintre ele constau în atributele lor specifice: **DisciplinaZi** pune accent pe activități tradiționale, în timp ce **Disciplinald** se concentrează pe activități care sprijină învățarea la distanță. Deși structura și funcționalitatea lor sunt similare, scopul fiecărei clase este să modeleze tipuri diferite de discipline, în funcție de contextul educațional.