

Mirza Daniel-Ionut

de ..

Data depunerii: 31-ian.-2025 04:14PM (UTC+0200)

ID-ul depunerii: 2576092300

Numele fișierului: Mirza_Daniel_Ionut_Documentatie_AIA_Licenta.pdf (2.35M)

Numărul cuvintelor: 17413

Numărul caracterelor: 113836

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	4
2 TEHNOLOGII WEB	6
2.1 LIMBAJUL DE PROGRAMARE JAVA	6
2.1.1 Istoria Java	6
2.1.2 Caracteristici esențiale	7
2.1.3 Programarea orientată pe obiecte	7
2.1.4 Biblioteci și ediții	8
2.1.5 Java Persistence API	9
2.1.6 Java Database Connectivity	10
2.1.7 Securitate	11
2.1.8 Exemplu de program	12
2.1.9 Concluzie Java	13
2.2 BIBLIOTECA SPRING BOOT	13
2.2.1 Istorie Spring Boot	14
2.2.2 Caracterisitici	14
2.2.3 Spring vs Spring Boot	15
2.2.4 Avantajele Spring Boot	15
2.2.5 Program Spring Boot	16
2.2.6 Concluzii Spring Boot	17
2.3 BIBLIOTECA APACHE POI	17
2.3.1 Istoria POI	17
2.3.2 Caracteristicile Apache POI	18
2.3.3 Avantaje Apache POI	19
2.3.4 Exemplu de program	19
2.3.5 Concluzie Apache POI	21
2.4 SISTEM DE GESTIONARE DE DATE MYSQL	21
2.4.1 Istoria MySQL	21
2.4.2 Caracteristici MySQL	22
2.4.3 MySQL vs SQL	23
2.4.4 Avantajele MySQL	24
2.4.5 Concluzie	25
2.5 BIBLIOTECA THYMELEAF	25
2.5.1 Istorie Thymeleaf	25
2.5.2 Caracteristici Thymeleaf	26
2.5.3 Thymeleaf vs Angular	26
2.5.4 Avantaje Thymeleaf	27
2.5.5 Concluzie	28
2.6 BOOTSTRAP 5	28
2.6.1 Istoria Bootstrap 5	28
2.6.2 Caracteristici Bootstrap 5	29
2.6.3 Avantajele Bootstrap 5	29
2.6.4 Concluzie	30

3 CERINȚA ȘI SPECIFICAȚIILE PROIECTULUI	31
3.1 CERINȚE FUNCȚIONALE	31
3.2 CERINȚE NEFUNCȚIONALE	31
3.3 CERINȚE TEHNICE	31
3.4 FUNCȚIONALITATEA PROIECTULUI	32
3.5 ANALIZA DE RISC	33
4 PROIECTARE	34
4.1 SCHEMA BLOC	34
4.2 DIAGRAMA DE CLASE, SECVENTA SI STARE	35
4.3 SCHEMA BAZEI DE DATE	36
5 IMPLEMENTAREA	39
5.1 Planuri	39
5.1.1 Clasa PlanInvatamant	39
5.1.2 Repository și Controllere	40
5.1.3 Sumar	42
5.2 Discipline	42
5.2.1 Clasa Discipline și Enum-uri	42
5.2.2 Repository și Controllere	44
5.2.3 Sumar	46
5.3 Extractoare	47
5.4 User	48
5.5 Templates	48
5.5.1 Template Index / Login / Sign-up	49
5.5.2 Template Planuri	49
5.5.3 Template Discipline	51
5.5.4 Sumar	53
6 TESTARE ȘI UTILIZARE	54
7 CONCLUZII	55
8 BIBLIOGRAFIE	56

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, inclusiv 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 unul dintre cele mai populare și versatile limbaje de programare din lume, cunoscut pentru principiul său „Write Once, Run Anywhere” (WORA), care asigură portabilitatea aplicațiilor pe diverse platforme. Dezvoltat de Sun Microsystems și lansat oficial în 1995, Java s-a impus rapid datorită robustei, securității și flexibilității sale [1].

Fiind un limbaj orientat pe obiecte, Java oferă un mediu ideal pentru dezvoltarea aplicațiilor enterprise, software-ului desktop, aplicațiilor mobile și sistemelor integrate. Platforma sa include diverse ediții precum **Java SE**, pentru aplicații generale, **Java EE**, pentru sisteme enterprise complexe, și **Java ME**, pentru dispozitive cu resurse limitate, ceea ce îl face potrivit pentru o gamă largă de scenarii. Pe lângă limbaj, ecosistemul Java este întărit de instrumente precum **JDBC**, pentru conectivitatea cu baze de date, și specificații precum **JPA**, care simplifică gestionarea persistentei datelor [2].

2.1.1 Istoria Java

Java este un limbaj de programare și o platformă dezvoltată de **Sun Microsystems**, lansată oficial în **1995**. Limbajul a fost creat de o echipă condusă de **James Gosling**, cunoscut și ca „părintele Java”, în cadrul unui proiect denumit inițial „Green Project” [3].

În anii 1990, pe măsură ce tehnologia digitală avansa rapid, exista o nevoie tot mai mare de un limbaj de programare care să fie portabil, robust și capabil să ruleze pe diferite tipuri de dispozitive. Proiectul „Green” s-a axat inițial pe dezvoltarea unui limbaj pentru dispozitive electronice de consum (precum televizoarele inteligente sau telecomenzile). Limbajul rezultat s-a numit inițial **Oak**, după un stejar aflat lângă biroul lui Gosling [1].

Numele „Oak” a trebuit schimbat, deoarece era deja folosit de altă companie. După mai multe propuneri, inclusiv **Silk** și **DNA**, s-a ales „Java”, inspirat de iubirea echipei pentru cafeaua Java, o cafea deosebit de populară [4].

Java a fost lansat oficial în 1995, odată cu versiunea Java 1.0, și a fost promovat sub sloganul „Write Once, Run Anywhere” (WORA). Acest concept sublinia avantajul principal al limbajului, și anume portabilitatea: codul scris în Java putea fi executat pe orice dispozitiv care avea o Mașină Virtuală Java (JVM). La început, Java a fost utilizat intens pentru dezvoltarea aplicațiilor web, fiind apreciat pentru posibilitatea de a crea **Applet-uri Java** care rulau direct în browsere. Aceasta a reprezentat un pas important către dezvoltarea aplicațiilor independente de platformă [4].

De-a lungul anilor, Java a trecut prin mai multe etape de evoluție care i-au consolidat poziția în lumea programării. În 1998, odată cu lansarea Java 2 (J2SE 1.2), au fost introduse platformele J2EE (Enterprise Edition), destinate aplicațiilor complexe pentru companii, și J2ME (Micro Edition), adaptată pentru dispozitive mobile și integrate. În 2004, Java 5 a adus

schimbări semnificative, cum ar fi introducerea Generics, Annotations și Autoboxing, care au facilitat programarea mai clară și mai eficientă [4].

Java 8, lansat în 2014, a fost o versiune revoluționară, integrând Streams API și Lambda Expressions, oferind astfel suport extins pentru programarea funcțională. În 2018, Java 11 a devenit prima versiune LTS (Long-Term Support) după Java 8, venind cu îmbunătățiri de performanță și eliminarea API-urilor învechite. Ulterior, Java 17, lansat în 2021, a adus noi funcționalități, precum pattern matching, și a continuat să perfecționeze funcțiile JVM, fiind o altă versiune LTS importantă pentru comunitatea dezvoltatorilor [4].

2.1.2 Caracteristici esențiale

Java se remarcă prin mai multe caracteristici care l-au făcut extrem de popular în rândul programatorilor și companiilor. Printre acestea, portabilitatea este una dintre cele mai importante trăsături, deoarece Mașina Virtuală Java (JVM) permite rularea codului Java pe orice platformă, indiferent de arhitectura hardware sau de sistemul de operare. De asemenea, Java este un limbaj complet orientat pe obiecte, ceea ce facilitează dezvoltarea modulară, reutilizarea codului și menținerea pe termen lung [5].

Un alt aspect important al Java este securitatea. Limbajul include mecanisme avansate pentru protecția datelor și prevenirea accesului neautorizat, făcându-l ideal pentru aplicații critice. În ceea ce privește performanța, deși Java este un limbaj interpretat (prin JVM), utilizează tehnici avansate precum compilarea JIT (Just-In-Time), care îmbunătățesc semnificativ viteza de execuție a aplicațiilor.

Java oferă și suport nativ pentru multithreading, ceea ce permite rularea simultană a mai multor procese, fiind astfel potrivit pentru aplicațiile complexe și de mare scală. Aceste caracteristici combinate au contribuit la succesul și adoptia pe scară largă a limbajului de-a lungul timpului.

2.1.3 Programarea orientată pe obiecte

Programarea orientată pe obiecte (OOP) este un paradigmă de programare care organizează codul în jurul conceptelor de **obiecte** și **clase**. Este utilizată pentru a dezvolta software modular, scalabil și reutilizabil. În OOP, un program este format din **obiecte** care interacționează între ele pentru a îndeplini sarcini.

Unul dintre conceptele esențiale ale OOP este **obiectul**, care reprezintă o entitate ce combină date și comportamente. Aceasta este o instanță a unei **clase**, ce poate fi considerată un şablon pentru crearea obiectelor. Clasele definesc atributele (datele) și metodele (funcționalitățile) pe care obiectele le vor avea [6].

Encapsularea este un alt principiu important al OOP și constă în protejarea datelor dintr-o clasă, permitând accesul la acestea doar prin metode specifice. Acest lucru ajută la ascunderea detaliilor de implementare și la oferirea unei interfețe clare pentru utilizatori [7].

Moștenirea permite unei clase să preia proprietăți și metode de la o altă clasă, ceea ce facilitează reutilizarea codului și extinderea funcționalităților existente. De exemplu, o clasă părinte „Animal” poate defini caracteristici generale, cum ar fi metoda „respira()”, iar clasele derivate „Pisică” și „Câine” pot adăuga propriile particularități [7].

Polimorfismul este capacitatea unui obiect de a se comporta diferit în funcție de context. Această caracteristică poate fi observată atât în timpul compilării (prin metode cu același nume, dar cu parametri diferiți), cât și în timpul execuției (prin metode suprascrisse în clasele derivate).

Abstracția este procesul prin care sunt ascunse detaliile de implementare, expunându-se doar funcționalitățile relevante. Acest lucru este realizat prin utilizarea claselor abstrakte și a interfețelor, care definesc contracte pe care clasele ce le implementează trebuie să le respecte.

Programarea orientată pe obiecte oferă un avantaj major prin organizarea modulară a codului, ceea ce îl face mai ușor de înțeles și gestionat. Clasele și obiectele permit împărțirea aplicației în părți mici și independente, fiecare cu responsabilități bine definite. Acest lucru nu doar că îmbunătățește claritatea codului, dar și facilitează colaborarea în echipe mari de dezvoltare.

Un alt beneficiu semnificativ este reutilizarea codului, posibilă datorită moștenirii. Clasele existente pot fi extinse pentru a adăuga noi funcționalități, fără a rescrie codul deja scris. Aceasta duce la economisirea timpului și resurselor, în special în proiectele de mari dimensiuni. În plus, polimorfismul și abstractizarea oferă o flexibilitate crescută, permitând adaptarea codului la cerințe noi și simplificând gestionarea comportamentelor complexe.

OOP contribuie la întreținerea mai usoară a aplicațiilor. Prin encapsulare, datele sunt protejate, iar riscul de a introduce erori prin modificări accidentale este redus. Totodată, structura modulară permite adăugarea sau actualizarea funcționalităților fără a afecta restul sistemului. Aceste avantaje fac ca programarea orientată pe obiecte să fie una dintre cele mai populare paradigme, ideală pentru dezvoltarea de software scalabil și robust.

Programarea orientată pe obiecte reprezintă o paradigmă puternică și flexibilă, care facilitează dezvoltarea de software modular, reutilizabil și scalabil. Prin concepte precum clasele, obiectele, moștenirea, polimorfismul, abstracția și encapsularea, OOP asigură o structură clară și bine organizată a codului. Aceste caracteristici o fac ideală pentru proiecte complexe, în care întreținerea, extinderea și colaborarea sunt esențiale pentru succesul pe termen lung.

2.1.4 Biblioteci și ediții

Bibliotecile Java (cunoscute și ca **framework-uri** sau **API-uri**) sunt colecții de clase predefinite care facilitează dezvoltarea de aplicații prin oferirea unor funcționalități reutilizabile. Ele sunt esențiale pentru eficiența dezvoltării software.

Java este disponibil în mai multe **ediții principale** (platforme), fiecare adresând nevoile specifice ale unui anumit tip de aplicație. Cele mai importante ediții sunt **Java SE (Standard Edition)**, **Java EE (Enterprise Edition)** (actualmente **Jakarta EE**) și **Java ME (Micro Edition)** [8].

Java SE (Standard Edition) reprezintă ediția de bază a platformei Java, utilizată pentru dezvoltarea aplicațiilor generale care pot rula pe diverse dispozitive, de la desktopuri la servere. Aceasta include Mașina Virtuală Java (JVM), care asigură portabilitatea aplicațiilor și un set extins de API-uri fundamentale. Printre acestea se numără java.lang, pentru operațiuni de bază precum manipularea stringurilor și a numerelor, java.util, pentru colecții și manipularea timpului, și java.io împreună cu java.nio, pentru gestionarea fișierelor și fluxurilor de date. De asemenea, Java SE oferă suport pentru dezvoltarea interfețelor grafice prin AWT, Swing și JavaFX. Este ideal pentru aplicații standalone și proiecte care nu necesită funcționalități avansate [9].

Java EE (Enterprise Edition), redenumită recent Jakarta EE, extinde funcționalitățile Java SE cu API-uri și framework-uri special concepute pentru aplicațiile enterprise. Aceasta oferă instrumente precum Servlets și JSP pentru crearea aplicațiilor web dinamice, JPA pentru gestionarea persistenței datelor și JMS pentru procesarea mesajelor între aplicații. De asemenea, include Enterprise Java Beans (EJB) pentru logica de afaceri și CDI pentru gestionarea ciclurilor de viață ale obiectelor. Java EE este optimizată pentru serverele de aplicații, precum WildFly, GlassFish sau TomEE, și este utilizată în principal pentru dezvoltarea de sisteme mari, scalabile și distribuite, cum ar fi portaluri web și aplicații bancare [10].

Java ME (Micro Edition) este o versiune adaptată a Java, destinată dispozitivelor cu resurse limitate, cum ar fi telefoane mobile vechi, dispozitive IoT sau alte echipamente integrate. Aceasta include un subset simplificat al API-urilor din Java SE, fiind optimizată pentru constrângerile hardware ale acestor dispozitive. Java ME utilizează specificații precum CLDC (Connected Limited Device Configuration) și MIDP (Mobile Information Device Profile) pentru a permite dezvoltarea aplicațiilor mobile și pentru IoT. Este folosită în principal pentru aplicațiile care rulează pe dispozitive cu spațiu redus de memorie și putere de procesare limitată, cum ar fi senzori, termostate sau dispozitive medicale [11].

Cele trei ediții principale ale Java sunt proiectate pentru a răspunde unor nevoi diverse: **Java SE** pentru aplicații de bază, **Java EE** pentru aplicații complexe și distribuite, și **Java ME** pentru dispozitive cu resurse limitate. Fiecare ediție contribuie la versatilitatea platformei Java, făcând-o potrivită pentru o gamă largă de scenarii și tehnologii.

2.1.5 Java Persistence API

Java Persistence API (JPA) este o specificație Java care facilitează gestionarea și interacțiunea cu baze de date relaționale într-un mod orientat pe obiecte. JPA definește

un set standard de reguli și adnotări care permit dezvoltatorilor să mapze clasele Java pe tabelele dintr-o bază de date.

JPA nu este o implementare propriu-zisă, ci o interfață. Pentru utilizarea sa efectivă, este necesar un framework sau o bibliotecă care implementează specificația JPA, cum ar fi **Hibernate**, **EclipseLink** sau **OpenJPA**.

JPA facilitează maparea obiect-relațională, permitând dezvoltatorilor să lucreze cu obiecte Java care reprezintă datele stocate în tabelele unei baze de date. În loc să se bazeze pe ² fișiere XML complexe, JPA utilizează adnotări pentru a specifica configurarea claselor și a relațiilor dintre ele. De exemplu, `@Entity` marchează o clasă ca entitate persistentă, în timp ce `@Table` definește tabelul asociat. Alte adnotări, precum `@Id`, indică cheia primară, iar `@OneToMany` sau `@ManyToOne` sunt folosite pentru a gestiona relațiile dintre entități [12].

JPA gestionează ciclul de viață al entităților, automatizând operațiunile de inserare, actualizare și ștergere în baza de date. În plus, oferă un limbaj propriu pentru interogări, JPQL (Java Persistence Query Language), care permite scrierea interogărilor orientate pe obiecte, fără a depinde de SQL-ul specific unui furnizor de bază de date. O altă caracteristică importantă este caching-ul, care îmbunătățește performanța prin reducerea interacțiunilor directe cu baza de date.

Un avantaj major al JPA este că simplifică interacțiunea cu bazele de date, eliminând necesitatea scrierii manuale a SQL-ului pentru operațiunile de bază. În plus, JPA este portabil, ceea ce înseamnă că aplicațiile scrise cu JPA pot funcționa cu diverse baze de date, atâtă timp cât se utilizează o implementare compatibilă, cum ar fi Hibernate sau EclipseLink [12].

Productivitatea dezvoltatorilor este crescută datorită utilizării adnotărilor, care reduc complexitatea configurării relațiilor dintre entități. De asemenea, JPA permite gestionarea relațiilor complexe între date, cum ar fi relațiile „unu-la-mulți” sau „mulți-la-mulți”, prin adnotări simple. În fine, utilizarea caching-ului și a mecanismelor de optimizare oferite de implementările JPA poate îmbunătăți performanța generală a aplicațiilor, reducând numărul de interogări către baza de date.

JPA este un instrument esențial pentru dezvoltatorii Java, oferind un mod standardizat și eficient de a interacționa cu bazele de date relaționale. Datorită flexibilității și puterii sale, este folosit în majoritatea aplicațiilor enterprise moderne. Implementările populare, cum ar fi Hibernate, extind și îmbunătățesc specificația, făcând JPA o alegere preferată pentru gestionarea persistentei datelor.

2.1.6 Java Database Connectivity

JDBC (Java Database Connectivity) este o tehnologie Java care permite conectarea aplicațiilor la baze de date relaționale și executarea interogărilor SQL. Este o

interfață standardizată în cadrul platformei Java, oferind dezvoltatorilor un mod uniform de a lucra cu baze de date, indiferent de furnizorul bazei de date (Oracle, MySQL, PostgreSQL, etc.). JDBC face parte din **Java SE** și oferă un set de clase și interfețe pentru conectarea la baze de date, trimiterea interogărilor SQL și manipularea rezultatelor.

JDBC oferă o interfață standard pentru conectarea aplicațiilor Java la baze de date relaționale. Prin JDBC, dezvoltatorii pot stabili o conexiune cu baza de date utilizând DriverManager și un URL specific bazei de date. După conectare, comenzi SQL pot fi executate fie printr-un obiect Statement pentru interogări simple, fie prin PreparedStatement pentru interogări parametrizate, care oferă o securitate mai bună împotriva atacurilor SQL Injection [13].

Rezultatele interogărilor sunt gestionate prin obiectul ResultSet, care permite navigarea printre rândurile și coloanele returnate de baza de date. JDBC suportă, de asemenea, drivere multiple care asigură compatibilitatea cu diverse tipuri de baze de date, precum MySQL, Oracle sau PostgreSQL. Astfel, codul scris folosind JDBC este portabil și poate funcționa cu orice bază de date, atât timp cât există un driver corespunzător [13].

Unul dintre principalele avantaje ale JDBC este flexibilitatea sa. Dezvoltatorii pot utiliza interogări SQL native, având un control complet asupra modului în care datele sunt accesate și manipulate. JDBC este, de asemenea, portabil, ceea ce înseamnă că aplicațiile scrise cu acesta pot fi folosite cu diverse baze de date, fără a necesita modificări majore, datorită standardizării interfeței [14].

De asemenea, API-ul JDBC este relativ simplu de utilizat, ceea ce îl face ideal pentru aplicații mici sau pentru dezvoltatorii care au nevoie de acces direct la baza de date. Totuși, pentru aplicații mai complexe, poate deveni repetitiv și consumator de timp, motiv pentru care framework-uri precum Hibernate sunt preferate. Chiar și așa, JDBC rămâne o bază solidă pentru interacțiunea cu baze de date relaționale și este esențial pentru înțelegerea conceptelor fundamentale ale programării cu baze de date [14].

JDBC este o tehnologie de bază pentru conectarea aplicațiilor Java la baze de date relaționale. Deși este simplu de utilizat pentru aplicații mici și interogări SQL directe, dezvoltatorii preferă adesea să folosească framework-uri mai avansate, precum Hibernate sau JPA, care adaugă un strat de abstractie și simplifică gestionarea bazelor de date. Totuși, JDBC rămâne un instrument esențial, mai ales pentru înțelegerea fundamentelor interacțiunii dintre aplicațiile Java și bazele de date.

2.1.7 Securitate

Java este renumit pentru mecanismele sale robuste de securitate, fiind conceput încă de la început ca un limbaj care pune accent pe siguranța aplicațiilor și protecția împotriva amenințărilor cibernetice. Platforma Java oferă un set de caracteristici integrate și biblioteci dedicate care ajută dezvoltatorii să creeze aplicații securizate.

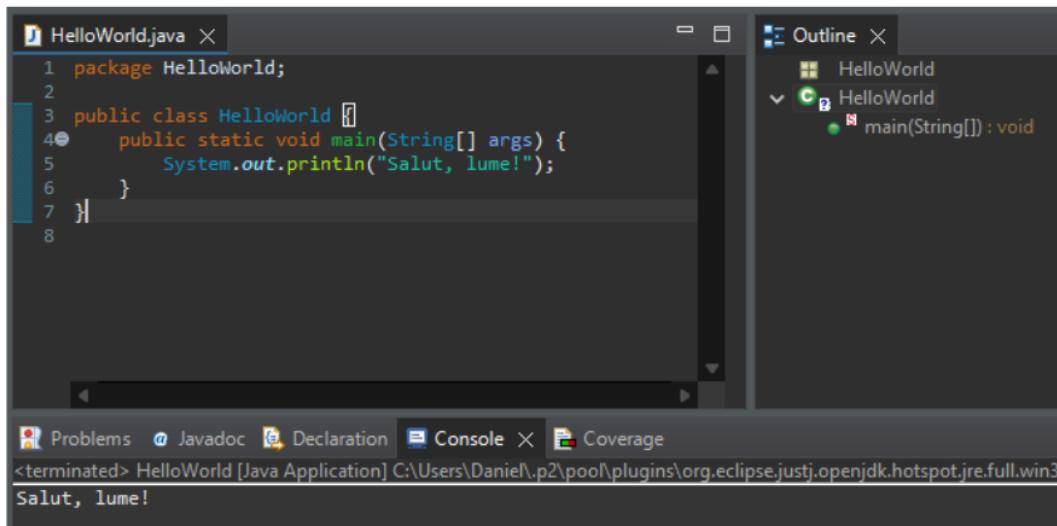
Java pune un accent deosebit pe securitate, oferind multiple mecanisme pentru protecția aplicațiilor și a datelor. În primul rând, Mașina Virtuală Java (JVM) creează un mediu controlat pentru rularea aplicațiilor, cunoscut sub numele de **sandbox**, care limitează acțiunile pe care codul neautorizat le poate executa. Această izolare împiedică accesul direct al aplicațiilor la resursele hardware sau sistemul de operare [15].

Un alt aspect important este verificarea bytecode-ului înainte de execuție. JVM analizează codul pentru a se asigura că respectă regulile platformei și nu conține instrucțiuni malicioase sau periculoase. Astfel, sunt prevenite erori precum coruperea memoriei sau rularea codului nesigur.

Securitatea este o componentă centrală a platformei Java, oferind mecanisme robuste și instrumente avansate pentru protecția datelor și aplicațiilor. Prin combinarea caracteristicilor integrate ale platformei cu bunele practici de dezvoltare, aplicațiile Java pot fi securizate împotriva majorității amenințărilor cibernetice moderne.

2.1.8 Exemplu de program

Un program simplu Java constă în general dintr-o clasă care conține o metodă principală (main), unde începe execuția programului, precum în figura 1.



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

Figura 1- Program Java

```
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!
```

2.1.9 Concluzie Java

Java rămâne un limbaj de programare esențial datorită combinației sale de portabilitate, flexibilitate și un ecosistem bine definit. Suportul pentru multiple ediții și API-uri puternice face din Java o alegere preferată pentru dezvoltarea de aplicații scalabile, robuste și sigure. Deși framework-urile moderne precum Spring sau Hibernate au extins funcționalitățile sale, concepțele fundamentale precum **JDBC** și **JPA** continuă să joace un rol important în interacțiunea cu bazele de date. În plus, evoluția constantă a limbajului, prin noile versiuni și îmbunătățiri aduse, asigură relevanța sa pe termen lung, menținând Java în centrul industriei de dezvoltare software. Indiferent de domeniul – enterprise, IoT sau aplicații web – Java rămâne o soluție de încredere pentru dezvoltatori.

2.2 BIBLIOTECA SPRING BOOT

Spring Boot este un framework modern și puternic, conceput pentru a simplifica procesul de dezvoltare a aplicațiilor Java. Bazat pe fundația solidă oferită de Spring

Framework, Spring Boot aduce configurare automată, instrumente intuitive și o integrare perfectă cu ecosistemul Spring. Este o alegere ideală pentru dezvoltatorii care doresc să creeze rapid aplicații web, API-uri REST sau microservicii scalabile, eliminând barierele tradiționale de configurare și reducând timpul necesar pentru inițializarea proiectelor. Cu servere web încorporate și suport nativ pentru arhitecturi cloud-native, Spring Boot facilitează construirea aplicațiilor moderne, eficiente și ușor de gestionat.

2.2.1 Istorie Spring Boot

Spring Boot este un framework open-source bazat pe Java, utilizat pentru dezvoltarea de aplicații web și microservicii. A fost creat de echipa Pivotal și se bazează pe **Spring Framework**.

Spring Boot a fost lansat pentru prima dată în anul 2014, cu scopul de a simplifica utilizarea Spring Framework și de a oferi o abordare modernă în dezvoltarea aplicațiilor Java. Prima versiune stabilă, Spring Boot 1.0.0, a fost bine primită datorită configurației automate și ușurinței de utilizare, eliminând multe dintre complexitățile tradiționale ale configurației aplicațiilor Spring [16].

Între 2015 și 2017, framework-ul a câștigat rapid popularitate în comunitatea de dezvoltare, devenind o soluție de bază pentru crearea de aplicații web și microservicii. Caracteristici precum starter POMs și serverele web încorporate au făcut din Spring Boot o alegere preferată pentru dezvoltatorii care doreau să lanseze rapid proiecte fără a se preocupa de configurații complicate.

În 2020, cu lansarea versiunii 2.x, Spring Boot a adus îmbunătățiri semnificative pentru susținerea microserviciilor și a integrării cu platformele moderne, cum ar fi Kubernetes. Acest lucru a permis adoptarea sa pe scară largă în mediile de producție, în special pentru arhitecturi cloud-native.

Odată cu lansarea versiunii 3.x, începând din 2023, Spring Boot a introdus suport complet pentru Java 17+ și a adus o integrare mai profundă cu GraalVM, permitând crearea de aplicații native extrem de eficiente. De asemenea, a continuat să evolueze, adaptându-se la noile cerințe tehnologice și consolidându-și poziția ca unul dintre cele mai populare framework-uri pentru dezvoltarea aplicațiilor Java moderne.

2.2.2 Caracteristici

3

Spring Boot se remarcă printr-o serie de caracteristici care îl fac ideal pentru dezvoltarea rapidă și eficientă a aplicațiilor Java. În primul rând, configurația automată simplifică procesul de inițializare a proiectelor, detectând automat și configurând bibliotecile necesare în funcție de dependențele utilizate.

De asemenea, framework-ul este optimizat pentru crearea de microservicii scalabile, facilitând astfel dezvoltarea arhitecturilor moderne distribuite. O altă caracteristică

importantă este includerea serverelor web încorporate, precum Tomcat, Jetty sau Undertow, ceea ce elimină necesitatea configurației unui server extern [17].

Spring Boot oferă, de asemenea, starter POMs, un set de dependențe predefinite care ajută la integrarea rapidă a funcționalităților comune, cum ar fi web, securitate sau baze de date. Instrumentul Spring Initializr contribuie la generarea rapidă de proiecte, economisind timp în faza de configurație inițială [17].

Pentru monitorizare și gestionare, Spring Boot include Spring Actuator, care oferă informații detaliate despre starea aplicației și ajută la depanare. Aceste caracteristici, combinate cu suportul pentru standardele moderne și integrările în ecosistemul cloud, fac din Spring Boot un framework extrem de versatil și ușor de utilizat.

2.2.3 Spring vs Spring Boot

Spring Framework este un framework cuprinzător pentru dezvoltarea aplicațiilor Java. Acesta oferă o gamă largă de instrumente și biblioteci pentru gestionarea aspectelor variante ale dezvoltării software, cum ar fi inversiunea controlului (IoC), programarea orientată pe aspecte (AOP), securitatea, accesul la baze de date și multe altele. Totuși, configurația în Spring Framework este mai complexă, necesitând adesea utilizarea XML sau a numeroase anotări, ceea ce crește timpul de inițializare și curba de învățare. Această flexibilitate ridicată face ca Spring să fie potrivit pentru aplicațiile complexe care necesită personalizare extinsă [18].

În contrast, Spring Boot este o extensie a Spring Framework creată pentru a simplifica dezvoltarea aplicațiilor. Oferă configurație automată, detectând și configuriând setările implicate pentru dependențele utilizate. De asemenea, vine cu servere web încorporate, precum Tomcat sau Jetty, eliminând necesitatea unui server extern. Starter POM-urile predefinite permit integrarea rapidă a funcționalităților comune, cum ar fi accesul la date, securitatea sau dezvoltarea web. Mai mult, instrumente precum Spring Actuator și Spring Initializr fac procesul de monitorizare, gestionare și inițializare a proiectelor mult mai simplu. Aceste caracteristici fac din Spring Boot o soluție excelentă pentru aplicații rapide, microservicii sau arhitecturi cloud-native [18].

Diferențele dintre cele două constau în nivelul de complexitate și în scopurile lor principale. Spring Framework necesită configurație manuală și este potrivit pentru aplicații mari și complexe, în timp ce Spring Boot oferă o experiență simplificată și rapidă, fiind ideal pentru dezvoltatorii care doresc să se concentreze pe funcționalitate, fără a pierde timp în configurație. Alegerea între cele două depinde de tipul și complexitatea proiectului.

2.2.4 Avantajele Spring Boot

Spring Boot se remarcă prin configurația automată și simplitatea procesului de dezvoltare, ceea ce îl face ideal pentru proiecte rapide și eficiente. Framework-ul elimină necesitatea configurațiilor complexe, detectând automat dependențele și aplicând setări

implicite optimizeaza. Generarea proiectelor este rapidă datorită instrumentului Spring Initializr, care creează un schelet de aplicație configurat pentru nevoile specifice ale dezvoltatorilor [18].

Un alt avantaj major este serverul web încorporat, care permite rularea imediată a aplicațiilor fără necesitatea unui server extern. În plus, integrarea completă cu ecosistemul Spring asigură acces facil la module precum Spring Data, Spring Security și Spring Cloud, extinzând funcționalitățile aplicației fără efort suplimentar. Suportul pentru microservicii și integrarea nativă cu platformele cloud-native îl fac o soluție excelentă pentru arhitecturile moderne distribuite.

Spring Boot oferă, de asemenea, instrumente avansate pentru monitorizare și gestionare prin Spring Actuator, ceea ce facilitează depanarea și optimizarea aplicațiilor. Combinând toate aceste caracteristici cu o comunitate vastă și documentație extinsă, Spring Boot asigură dezvoltarea rapidă, scalabilă și ușor de întreținut a aplicațiilor moderne.

2.2.5 Program Spring Boot

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.

The screenshot shows the Eclipse IDE interface. The left pane displays the code for `SalutController.java`, which contains a single endpoint method `spuneSalut()` that returns the string "Salut, lume!". The right pane shows the `Outline` view, which lists the package `HelloWorld` and the class `SalutController`, along with its method `spuneSalut()`. The bottom navigation bar includes tabs for Problems, Javadoc, Declaration, Console, and Coverage. The `Console` tab is active, showing the output "Salut, lume!".

```
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 }
```

Problems Javadoc Declaration Console X Coverage

<terminated> HelloWorld [Java Application] C:\Users\Daniel\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_2

Salut, lume!

Figura 2 - Program Spring Boot

În figura 2 arătăm cum pentru a adăuga funcționalitate, dezvoltatorul creează un controller simplu, o clasă Java anotată 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.

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.

2.2.6 Concluzii Spring Boot

Spring Boot se distinge prin capacitatea să de a accelera dezvoltarea, oferind performanță optimizată "out of the box" și suport avansat pentru monitorizare și gestionare. Prin integrarea strânsă cu modulele Spring și prin simplitatea implementării, Spring Boot devine un instrument esențial pentru dezvoltatorii care doresc să livreze aplicații robuste și scalabile într-un timp scurt. Cu o comunitate puternică și documentație detaliată, Spring Boot continuă să fie unul dintre cele mai utilizate framework-uri în ecosistemul Java, susținând cu succes cerințele aplicațiilor moderne și arhitecturilor distribuite.

2.3 BIBLIOTECA APACHE POI

Apache POI este o bibliotecă open-source dezvoltată pentru a permite manipularea fișierelor Microsoft Office în aplicații Java. Reprezintă o soluție extrem de versată și fiabilă, care oferă suport atât pentru formatele vechi, binare (.xls, .doc, .ppt), cât și pentru cele moderne, bazate pe XML (.xlsx, .docx, .pptx). Datorită modularității și funcționalităților sale avansate, Apache POI permite dezvoltatorilor să creeze, să modifice și să proceseze fișiere Office cu un nivel ridicat de control și eficiență. Folosită în diverse domenii, de la educație și finanțe până la aplicații enterprise, biblioteca este apreciată pentru flexibilitatea, compatibilitatea multiplatformă și capacitatea să de a gestiona mari volume de date.

2.3.1 Istoria POI

Istoria bibliotecii Apache POI începe la începutul anilor 2000, fiind creată ca parte a proiectului Apache Jakarta, dedicat dezvoltării de librării și instrumente pentru Java. Scopul principal al acestei biblioteci a fost să ofere dezvoltatorilor o soluție open-source pentru manipularea fișierelor Microsoft Office, care utilizau formate binare proprietare greu de interpretat. Prima versiune a bibliotecii a fost axată pe gestionarea fișierelor Excel (.xls) prin API-ul HSSF (Horrible Spreadsheet Format), care permitea lucrul cu formatul BIFF8 folosit în Microsoft Excel 97-2003 [19].

Pe măsură ce popularitatea suitei Microsoft Office a crescut, Apache POI a fost extinsă pentru a suporta și alte formate de fișiere. Astfel, au fost introduse module precum HWPF pentru fișiere Word (.doc), HSLF pentru fișiere PowerPoint (.ppt) și DDF pentru diagrame și structuri grafice Office.

După lansarea suitei Microsoft Office 2007, care a introdus formatele bazate pe XML, biblioteca Apache POI a evoluat pentru a include suport pentru noile tipuri de fișiere. Au fost adăugate API-uri precum XSSF (XML Spreadsheet Format) pentru fișiere Excel .xlsx, XWPF (XML Word Processing Format) pentru fișiere Word .docx și XSLF (XML SlideShow Format) pentru fișiere PowerPoint .pptx. Această extindere a permis compatibilitatea cu formatele moderne și a oferit dezvoltatorilor funcționalități avansate.

De-a lungul anilor, Apache POI a devenit din ce în ce mai populară, fiind utilizată pe scară largă pentru gestionarea datelor, generarea rapoartelor automate și integrarea fișierelor Office în soluții Java. Comunitatea open-source și fundația Apache au sprijinit constant dezvoltarea bibliotecii, contribuind la îmbunătățirea funcționalităților și performanței.

Astăzi, Apache POI este una dintre cele mai utilizate biblioteci Java pentru manipularea fișierelor Office, fiind apreciată pentru flexibilitatea și documentația sa extinsă. Biblioteca suportă atât formatele binare vechi, cât și cele moderne bazate pe XML, și rămâne un instrument indispensabil în diverse domenii, de la educație și finanțe până la dezvoltarea de aplicații complexe.

2.3.2 Caracteristicile Apache POI

Apache POI este o bibliotecă Java recunoscută pentru capacitatea sa de a manipula fișiere Microsoft Office. Aceasta suportă atât formatele vechi, binare, cât și pe cele moderne, bazate pe XML. De exemplu, pentru fișierele Excel, oferă HSSF (pentru .xls) și XSSF (pentru .xlsx), iar pentru fișierele Word, HWPF (pentru .doc) și XWPF (pentru .docx). În mod similar, pentru PowerPoint, suportă HSLF (pentru .ppt) și XSLF (pentru .pptx).

Biblioteca permite crearea, citirea și modificarea fișierelor Office, inclusiv extractia datelor din foi de calcul Excel, documente Word și prezentări PowerPoint. De asemenea, utilizatorii pot insera texte, imagini, diagrame sau alte elemente grafice în fișierele respective. Apache POI oferă funcționalități avansate, precum gestionarea stilurilor de celule, formule matematice și formatari condiționate în Excel. În Word, biblioteca permite lucrul cu tabele, paragrafe, stiluri de text și elemente precum antete și note de subsol. Pentru PowerPoint, oferă suport pentru editarea diapozitivelor, textului, animațiilor și obiectelor multimedia [20].

Proiectul este modular, fiecare componentă fiind dedicată unui anumit tip de fișier. Această abordare permite dezvoltatorilor să utilizeze doar modulele necesare în funcție de cerințe. De asemenea, fiind complet implementată în Java, Apache POI este compatibilă cu orice platformă care suportă JVM, ceea ce o face o soluție ideală pentru diverse aplicații enterprise, desktop sau web.

Un alt punct forte al bibliotecii este capacitatea sa de a gestionea mari volume de date. Prin API-ul său optimizat, SXSSF (Streaming Usermodel API), poate procesa eficient

foi de calcul mari, chiar și cele care conțin mii de linii și coloane. În plus, biblioteca permite extractia textului brut din fișiere Office, conversia între formatele vechi și cele moderne, precum și gestionarea hyperlink-urilor, comentariilor și altor metadate.

Apache POI este o bibliotecă open-source, disponibilă gratuit sub licența Apache License 2.0, fiind sprijinită de o comunitate activă care contribuie constant cu actualizări și suport. Documentația sa extinsă, care include ghiduri și exemple de cod, facilitează adoptarea și utilizarea eficientă a bibliotecii. Mai mult, Apache POI se integrează cu alte tehnologii Java, cum ar fi Spring sau Hibernate, ceea ce o face extrem de versatilă pentru crearea de soluții robuste și scalabile.

2.3.3 Avantaje Apache POI

Apache POI este o bibliotecă extrem de versatilă, care permite manipularea fișierelor Microsoft Office în aplicații Java. Unul dintre cele mai mari avantaje este suportul său extins pentru formatele Office, incluzând atât fișierele binare vechi (.xls, .doc, .ppt), cât și cele moderne bazate pe XML (.xlsx, .docx, .pptx). Această compatibilitate o face utilă în diverse scenarii, indiferent de versiunea suitei Office utilizate. În plus, Apache POI este gratuită și open-source, fiind disponibilă sub licența Apache License 2.0, ceea ce o face accesibilă atât pentru proiecte personale, cât și comerciale [21].

Un alt avantaj major este flexibilitatea oferită de structura modulară a bibliotecii. Fiecare tip de fișier este gestionat de un modul dedicat, ceea ce permite integrarea doar a componentelor necesare, optimizând resursele și performanța. De asemenea, biblioteca poate gestiona eficient fișiere mari, datorită API-ului optimizat SXSSF pentru Excel, ceea ce o face ideală pentru procese ce implică volume mari de date. Funcționalitățile sale avansate, precum gestionarea stilurilor, formatarea condiționată, tabelele complexe și animațiile, oferă dezvoltatorilor un control detaliat asupra fișierelor Office [21].

Susținută de o comunitate activă și o documentație extinsă, Apache POI este ușor de adoptat și utilizat. Integrarea facilă cu alte tehnologii Java, precum Spring sau Hibernate, o face un instrument de încredere pentru construirea aplicațiilor enterprise. Stabilitatea și fiabilitatea sa, dovedite în numeroase scenarii de utilizare, consolidează poziția Apache POI ca una dintre cele mai importante soluții pentru manipularea fișierelor Office în mediul Java.

2.3.4 Exemplu de program

Exemplul din figura 3 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
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             // Creează 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             // Salvearea fisierului modificat
25             FileOutputStream outputStream = new FileOutputStream("exemplu_modificat.xlsx");
26             workbook.write(outputStream);
27
28             // Închiderea workbook și file stream
29             workbook.close();
30             file.close();
31             outputStream.close();
32         }
33     }
34 }
```

Figura 3 - Program Apache POI

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 exceptiilor este esențială, deoarece operațiile de citire și scriere implică riscul de erori legate de I/O.

2.3.5 Concluzie Apache POI

Apache POI se remarcă prin combinația sa de accesibilitate, funcționalități avansate și stabilitate. Fiind gratuită, bine documentată și susținută de o comunitate activă, biblioteca oferă dezvoltatorilor un instrument puternic pentru manipularea fișierelor Microsoft Office în Java. Integrarea sa ușoară cu alte tehnologii și capacitatea de a gestiona eficient sarcini complexe o fac o alegere de încredere pentru proiectele care necesită procesarea fișierelor Office. Datorită acestor caracteristici, Apache POI continuă să fie o soluție indispensabilă în mediul de dezvoltare Java.

2.4 SISTEM DE GESTIONARE DE DATE MYSQL

MySQL este unul dintre cele mai populare sisteme de gestionare a bazelor de date relaționale din lume, recunoscut pentru performanță, fiabilitatea și ușurința în utilizare. Lansat ca software open-source, MySQL a devenit rapid o alegere preferată pentru aplicații web și proiecte software datorită accesibilității sale și a compatibilității extinse cu diverse platforme și limbi de programare. Utilizat pe scară largă de companii precum Google, Facebook și Twitter, MySQL este esențial pentru gestionarea eficientă a datelor în proiecte de toate dimensiunile, de la aplicații mici la platforme globale.

Baza de date reprezintă o colecție organizată de date, stocate și accesate electronic, care permite gestionarea eficientă a informațiilor. Într-o lume în care datele sunt esențiale pentru luarea deciziilor, bazele de date sunt fundamentalul majorității aplicațiilor software și sistemelor informatici. De la stocarea informațiilor despre utilizatori în aplicațiile web, până la gestionarea complexă a tranzacțiilor financiare, bazele de date sunt utilizate în diverse domenii, asigurând acces rapid și sigur la informații. Ele sunt gestionate prin sisteme specializate, numite sisteme de gestionare a bazelor de date (DBMS), care facilitează crearea, întreținerea și interogarea datelor. Printre cele mai utilizate sisteme de gestionare a bazelor de date se numără MySQL, o soluție recunoscută pentru performanță și versatilitatea sa.

2.4.1 Istoria MySQL

MySQL este unul dintre cele mai utilizate sisteme de gestionare a bazelor de date relaționale din lume, fiind apreciat pentru viteza, fiabilitatea și ușurința sa de utilizare. Creat în 1995 de Michael Widenius, David Axmark și Allan Larsson în cadrul companiei MySQL AB din Suedia, MySQL a fost conceput pentru a răspunde nevoilor emergente ale aplicațiilor web. Numele său provine din combinația dintre "My", numele fiicei lui Michael Widenius, și "SQL", limbajul de interogare utilizat în bazele de date.

Lansarea inițială în 1995 a adus un software open-source care a atras rapid interesul dezvoltatorilor datorită modelului său simplu și eficient. În anii 2000, MySQL a devenit extrem de popular, fiind integrat în platforma LAMP (Linux, Apache, MySQL, PHP/Perl/Python), o soluție standard pentru dezvoltarea web. Modelul de licențiere duală,

care includea o licență open-source (GNU GPL) și una comercială pentru aplicații proprietare, a contribuit la adoptarea sa pe scară largă [22].

4 Un moment definitiv în istoria MySQL a avut loc în 2008, când Sun Microsystems a achiziționat MySQL AB pentru aproximativ un miliard de dolari, consolidându-i poziția în ecosistemul open-source. În 2010, Oracle Corporation a preluat Sun Microsystems, devenind astfel noul proprietar al MySQL. Această tranzacție a generat controverse în comunitatea open-source, care se temea că Oracle ar putea restricționa dezvoltarea software-ului. Aceste temeri au dus la crearea unor alternative importante, cum ar fi MariaDB, dezvoltată de Michael Widenius, și Percona Server [22].

De-a lungul timpului, MySQL a evoluat tehnologic, introducând funcționalități avansate, precum replicarea, tranzacțiile ACID, și motoare de stocare multiple, cum ar fi InnoDB și MyISAM. Astăzi, MySQL este utilizat de companii și organizații importante, precum Google, Facebook, Twitter și YouTube, rămânând un pilon al dezvoltării aplicațiilor web și software.

În concluzie, MySQL a evoluat de la un proiect simplu, dar revoluționar, la o tehnologie fundamentală în gestionarea datelor. Deși viitorul său sub conducerea Oracle a ridicat unele întrebări, sprijinul comunității și al companiilor asigură relevanța sa continuă în peisajul tehnologic.

6 2.4.2 Caracteristici MySQL

MySQL este un sistem de gestionare a bazelor de date relaționale cunoscut pentru performanță, flexibilitatea și fiabilitatea sa, utilizat pe scară largă în aplicații web și alte medii software. Una dintre principalele sale calități este performanța ridicată, datorită optimizării resurselor și a suportului pentru motoare de stocare precum InnoDB, destinat tranzacțiilor, și MyISAM, optimizat pentru operații rapide de citire.

MySQL este compatibil cu mai multe sisteme de operare, inclusiv Windows, Linux, macOS și Unix, și suportă o varietate de limbaje de programare, precum Java, C++, Python, PHP, Ruby sau Go. Această portabilitate îl face o soluție versatilă pentru dezvoltatori și organizații. De asemenea, MySQL este scalabil și poate gestiona baze de date de dimensiuni variate, de la aplicații mici la proiecte mari și complexe. Funcționalitățile de replicare permit copierea automată a datelor între servere, oferind atât redundanță, cât și capacitatea de a gestiona volume mari de trafic [23].

2 Un alt aspect important al MySQL este modelul său de licențiere duală. Acesta este disponibil gratuit sub licență open-source GNU GPL, dar oferă și opțiuni comerciale pentru utilizare în aplicații proprietare. În plus, ușurința în utilizare este un punct forte al MySQL, datorită unei interfețe intuitive și a instrumentelor precum MySQL Workbench, care simplifică proiectarea, gestionarea și administrarea bazelor de date.

Sistemul asigură integritatea datelor prin suportul pentru tranzacții ACID (Atomicitate, Consistență, Izolare, Durabilitate), oferind funcționalități precum rollback, commit și puncte de salvare. De asemenea, securitatea avansată este garantată prin autentificare robustă, management al utilizatorilor bazat pe roluri și privilegii, criptare a datelor și transferuri securizate prin SSL/TLS [23].

MySQL include caracteristici avansate de replicare și clustering. Replicarea permite sincronizarea automată a datelor între servere, iar soluțiile de clustering, cum ar fi MySQL NDB Cluster, oferă posibilitatea gestionării bazelor de date distribuite. Acestea suportă și utilizarea mai multor motoare de stocare, ceea ce permite adaptarea la nevoi specifice: InnoDB pentru tranzacții complexe, MyISAM pentru acces rapid și Memory pentru operații rapide în memorie.

Standardizarea SQL este o altă caracteristică esențială a MySQL, care respectă standardele limbajului SQL și suportă comenzi complexe precum JOIN, GROUP BY și HAVING. Sistemul poate gestiona simultan mai multe baze de date pe același server, oferind suport pentru proiecte diverse și complexe.

MySQL beneficiază de o comunitate activă și globală, care contribuie la îmbunătățirea sa continuă și oferă suport tehnic. Oracle, compania care deține MySQL, asigură documentație oficială și opțiuni de suport comercial, ceea ce îl face o alegere fiabilă atât pentru dezvoltatori independenți, cât și pentru companii mari.

Aceste caracteristici fac din MySQL o soluție extrem de performantă, sigură și versatilă pentru gestionarea bazelor de date, adaptabilă atât nevoilor simple, cât și celor mai complexe proiecte.

2.4.3 MySQL vs SQL

Deși MySQL și SQL sunt adesea asociate, acestea reprezintă concepte diferite, fiecare având un scop distinct. SQL, sau Structured Query Language, este un limbaj standardizat utilizat pentru gestionarea și interogarea bazelor de date relaționale. Este un set de comenzi care permite utilizatorilor să creeze, să modifice, să șteargă și să interogheze datele dintr-o bază de date. SQL este recunoscut internațional ca un standard și este implementat de diferite sisteme de gestionare a bazelor de date relaționale (RDBMS), inclusiv MySQL [24].

3

În contrast, MySQL este un sistem concret de gestionare a bazelor de date relaționale (RDBMS) care folosește SQL pentru a lucra cu datele. MySQL oferă o platformă completă pentru crearea, stocarea și gestionarea bazelor de date. Este un software dezvoltat pentru a implementa funcționalitățile limbajului SQL, dar adaugă și caracteristici proprii, cum ar fi suportul pentru replicare, clustering și diverse motoare de stocare.

SQL este doar un limbaj, fără funcționalități de implementare sau stocare a datelor, fiind utilizat pentru a interacționa cu bazele de date. De exemplu, o interogare SQL standard,

precum SELECT * FROM tabel WHERE coloana = 'valoare';, poate fi rulată pe diferite sisteme RDBMS, cum ar fi MySQL, PostgreSQL, Oracle sau Microsoft SQL Server. MySQL, pe de altă parte, este software-ul concret în care această interogare este executată, oferind instrumente suplimentare pentru gestionarea și administrarea bazelor de date, cum ar fi interfața grafică MySQL Workbench [24].

Deși SQL respectă standardele internaționale și poate fi utilizat în orice RDBMS compatibil, MySQL introduce extensii proprii care adaugă funcționalități unice. De exemplu, MySQL permite utilizarea mai multor motoare de stocare, cum ar fi InnoDB pentru tranzacții sau MyISAM pentru acces rapid la date. În plus, MySQL include caracteristici pentru securitate, cum ar fi gestionarea utilizatorilor, criptarea datelor și suportul pentru conexiuni securizate prin SSL/TLS.

SQL definește doar sintaxa și structura pentru lucru cu bazele de date, dar nu oferă mecanisme pentru replicare, scalare sau stocare fizică a datelor. Aceste funcționalități sunt implementate de sisteme precum MySQL. De asemenea, SQL nu include instrumente grafice pentru gestionarea bazelor de date, în timp ce MySQL oferă astfel de soluții, simplificând munca administratorilor și dezvoltatorilor.

În concluzie, SQL este limbajul de bază utilizat pentru interacțiunea cu bazele de date relaționale, în timp ce MySQL este un sistem de gestionare a bazelor de date care implementează acest limbaj și oferă un set complet de funcționalități pentru administrarea și utilizarea bazelor de date. Dacă SQL este fundamentalul teoretic, MySQL reprezintă aplicația practică a acestuia.

2.4.4 Avantajele MySQL

MySQL se remarcă prin costurile reduse și accesibilitatea sa, fiind un software open-source disponibil gratuit sub licența GNU GPL. Acest lucru îl face ideal pentru startupuri, proiecte mici sau organizații care doresc să beneficieze de un sistem de gestionare a bazelor de date performant fără costuri inițiale. În plus, pentru proiecte comerciale sau complexe, Oracle oferă versiuni comerciale cu suport tehnic dedicat.

Un alt avantaj major al MySQL este performanța sa ridicată și scalabilitatea. Este optimizat pentru a gestiona volume mari de date și interogări complexe, fiind utilizat pe scară largă în aplicații web și platforme globale precum Google, Facebook și YouTube. Suportul pentru replicare și clustering asigură redundanță și disponibilitate ridicată a datelor, în timp ce compatibilitatea cu diverse sisteme de operare și limbaje de programare, precum PHP, Python și Java, îl face extrem de versatil pentru o gamă largă de aplicații.

Securitatea avansată reprezintă un alt punct forte al MySQL, oferind autentificare bazată pe roluri, criptare a datelor și conexiuni securizate prin SSL/TLS. În plus, suportul pentru tranzacții conforme cu standardul ACID, gestionat prin motorul de stocare InnoDB, garantează integritatea datelor în aplicații critice. Aceste caracteristici, combinate cu o

comunitate activă și suport extins, fac din MySQL o soluție fiabilă și de încredere pentru dezvoltatori și companii de toate dimensiunile.

2.4.5 Concluzie

MySQL se distinge prin flexibilitatea, scalabilitatea și securitatea sa, oferind funcționalități avansate precum suport pentru tranzacții ACID, replicare și clustering. Fiind atât o soluție accesibilă datorită naturii sale open-source, cât și una puternică prin funcționalitățile sale comerciale, MySQL răspunde unei game largi de nevoi ale dezvoltatorilor și organizațiilor. Cu o comunitate globală activă și suport tehnic oferit de Oracle, MySQL rămâne o alegere de încredere pentru gestionarea bazelor de date în era digitală.

2.5 BIBLIOTECA THYMELEAF

Thymeleaf este un motor de şablonane server-side conceput special pentru aplicațiile web Java, fiind bine integrat în ecosistemul Spring Framework. Popularitatea sa derivă din simplitatea utilizării, flexibilitatea și capacitatea de a genera şablonane HTML dinamice, dar valide, care pot fi vizualizate și editate direct în browsere. Cu un set bogat de funcționalități, inclusiv suport pentru internaționalizare, securitate încorporată și posibilitatea de a reutiliza fragmente de şablonane, Thymeleaf este o alegere excelentă pentru dezvoltarea aplicațiilor web tradiționale și multilingve.

2.5.1 Istorie Thymeleaf

Thymeleaf este o bibliotecă Java dedicată construirii de aplicații web, utilizată în special pentru generarea de şablonane de interfață într-un mod simplu și eficient. Dezvoltată în 2011 de către Daniel Fernández, această bibliotecă a fost creată pentru a oferi o alternativă modernă și prietenoasă la soluțiile existente, cum ar fi JSP (JavaServer Pages) sau FreeMarker. Thymeleaf a fost conceput pentru a fi intuitiv și accesibil pentru dezvoltatori, permitând utilizarea atât pe partea serverului, cât și pe partea clientului. De asemenea, unul dintre aspectele sale inovatoare a fost conceptul de „Natural Templates”, prin care şablonanele HTML sunt valide și editabile direct în browsere, ceea ce simplifică procesul de dezvoltare.

Prima versiune, lansată în 2011, s-a remarcat prin suportul nativ pentru HTML5, o caracteristică rară la acel moment. În 2012, Thymeleaf 2.0 a adus funcționalități noi, precum suportul pentru şablonane fragmentate și îmbunătățiri în procesarea datelor condiționale și iterative. Integrarea cu Spring Framework a devenit tot mai solidă, contribuind la popularitatea sa printre dezvoltatorii care construiau aplicații Spring MVC.

În 2017, lansarea Thymeleaf 3.0 a reprezentat un punct de cotitură. Această versiune a introdus un mecanism de parsare mai eficient și mai rapid, alături de concepte precum Spring Layout Dialect, pentru o organizare mai bună a şablonelor. Thymeleaf 3.0 a continuat să îmbunătățească suportul pentru HTML5, fiind ideal pentru aplicațiile moderne.

Datorită integrării excelente cu Spring Boot, Thymeleaf a devenit una dintre opțiunile implicite pentru generarea de şabioane în acest ecosistem.

Astăzi, Thymeleaf este folosit pe scară largă în dezvoltarea aplicațiilor Java web și este întreținut activ de comunitatea open-source. Biblioteca continuă să fie un exemplu de simplitate, eficiență și integrare excelentă în cadrul tehnologiilor moderne.

2.5.2 Caracteristici Thymeleaf

Thymeleaf este o bibliotecă cu numeroase caracteristici care contribuie la popularitatea sa în dezvoltarea aplicațiilor web Java. Una dintre principalele sale calități este suportul pentru şabioane naturale (Natural Templates), care sunt fișiere HTML complet valide. Acestea pot fi deschise și editate direct în browsere sau editori de text, simplificând astfel procesul de dezvoltare, deoarece pot fi vizualizate fără a rula aplicația pe server.

Biblioteca este compatibilă cu standardele moderne HTML5, ceea ce permite dezvoltarea de pagini conforme cu specificațiile actuale. În plus, Thymeleaf poate fi extins pentru a lucra și cu alte formate, cum ar fi XML, text, JavaScript sau CSS. Flexibilitatea sa permite utilizarea atât pentru generarea de conținut pe partea serverului (server-side rendering), cât și pentru generarea de conținut static sau preprocesat pe partea clientului, ceea ce îl face potrivit pentru o varietate de aplicații [25].

O altă caracteristică importantă este integrarea perfectă cu ecosistemul Spring, în special cu Spring MVC și Spring Boot. Aceasta permite rezolvarea facilă a datelor din model și oferă suport nativ pentru mesaje, validări și generarea de URL-uri dinamice. Thymeleaf include, de asemenea, suport pentru internaționalizare, ceea ce îl face ideal pentru aplicațiile multilingve.

Pentru o personalizare avansată, biblioteca oferă posibilitatea de a crea dialecte proprii, extinzând funcționalitatea standard pentru a răspunde unor cerințe specifice. Thymeleaf include directive care permit procesarea condițională și iterativă a datelor, cum ar fi afișarea dinamică pe baza condițiilor (th:if, th:unless) sau iterarea prin liste (th:each) [25].

Acstea caracteristici fac din Thymeleaf o alegere excelentă pentru aplicațiile Java web, oferind un echilibru între ușurința utilizării și flexibilitatea avansată necesară în proiectele moderne.

2.5.3 Thymeleaf vs Angular

Thymeleaf și Angular diferă semnificativ în modul în care procesează și livră datele către utilizator. Thymeleaf este un motor de şabioane server-side, ceea ce înseamnă că procesarea şabioanelor și inserarea datelor au loc pe server, rezultând un HTML final care este trimis către client (browser). În contrast, Angular procesează datele pe partea clientului, generând interfață dinamică direct în browser, folosind JavaScript și HTML.

Din punctul de vedere al paradigmelor utilizate, Thymeleaf este centrat pe generarea statică de pagini HTML și este mai potrivit pentru aplicații care nu necesită o interacțiune intensă a utilizatorului. Pe de altă parte, Angular urmează paradigma SPA (Single-Page Application), care permite actualizarea dinamică a interfeței fără reîncărcarea paginii, fiind ideal pentru aplicații interactive și complexe [26].

Rolul lor în arhitectura unei aplicații este, de asemenea, diferit. Thymeleaf este utilizat în principal pentru renderizarea inițială a paginilor HTML și face parte integrantă din arhitectura server-side, în timp ce Angular este folosit exclusiv pentru gestionarea interfeței utilizatorului pe partea clientului, separând complet frontend-ul de backend.

Integrarea cu backend-ul este un alt aspect distinct. Thymeleaf este strâns legat de backend-ul Java, având o integrare excelentă cu Spring Framework și permitând accesul direct la datele din model. Angular, în schimb, comunică cu backend-ul prin API-uri REST sau GraphQL, folosind HTTP pentru a sincroniza datele, ceea ce îl face mai flexibil pentru aplicațiile distribuite [26].

În ceea ce privește performanța și scalabilitatea, Thymeleaf poate deveni mai lent pe măsură ce numărul utilizatorilor crește, deoarece procesarea şablonelor are loc pe server. Angular reduce această încărcare a serverului prin mutarea procesării interfeței pe partea clientului, deși performanța poate fi afectată pe dispozitive mai lente sau în browsere mai vechi.

2.5.4 Avantaje Thymeleaf

Thymeleaf este un motor de şablonare server-side extrem de popular în ecosistemul Java, datorită integrării sale naturale cu Spring Framework. Este construit pentru a funcționa perfect cu Spring MVC și Spring Boot, permitând utilizarea directă a datelor din model, gestionarea facilă a mesageriei și validărilor, precum și generarea de URL-uri dinamice. Această compatibilitate face din Thymeleaf o alegere preferată pentru dezvoltatorii care lucrează în acest mediu [25].

Un alt avantaj major este conceptul de „Natural Templates”. Fișierele HTML utilizate în Thymeleaf sunt complet valide și pot fi deschise și editate direct în browsere sau editori, ceea ce simplifică testarea și dezvoltarea. Dezvoltatorii pot vedea și ajusta şablonanele fără a depinde de un server, reducând timpul și complexitatea procesului de implementare.

Securitatea este integrată nativ în Thymeleaf, toate datele fiind escapate automat pentru a preveni atacurile XSS (cross-site scripting). În plus, suportul pentru internaționalizare (i18n) îl face ideal pentru aplicații multilingve, asigurând traducerea dinamică a textelor și gestionarea localizării. De asemenea, reutilizarea codului este facilitată prin fragmente de şablonare, ceea ce contribuie la organizarea clară a aplicației și reduce redundanța [25].

Thymeleaf oferă și flexibilitate în generarea de conținut, fiind capabil să proceseze HTML5, XML, text, JavaScript sau CSS. Cu o curbă de învățare scurtă și o configurare simplă, este ușor de utilizat chiar și de către dezvoltatorii mai puțin experimentați, ceea ce îl face o soluție practică pentru aplicațiile web tradiționale de dimensiuni mici și medii.

2.5.5 Concluzie

Thymeleaf se remarcă prin integrarea sa naturală cu Spring Framework, ușurința utilizării și caracteristici care reduc timpul de dezvoltare, cum ar fi şablonanele naturale și securitatea nativă. Aceste avantaje îl fac o soluție practică și eficientă pentru aplicațiile server-side de dimensiuni mici și medii, care nu necesită interacțiuni complexe ale utilizatorilor. În ansamblu, Thymeleaf oferă un echilibru ideal între simplitate, flexibilitate și putere, fiind o alegere potrivită pentru echipele care dezvoltă aplicații web bazate pe Java.

2.6 BOOTSTRAP 5

Bootstrap 5 este cea mai recentă versiune a uneia dintre cele mai populare framework-uri front-end utilizate în dezvoltarea web. Lansat oficial în mai 2021, Bootstrap 5 aduce îmbunătățiri semnificative care reflectă cerințele moderne ale industriei, cum ar fi renunțarea la jQuery, un sistem de grilă modernizat și suport extins pentru accesibilitate și personalizare. Cu un design rafinat, performanță îmbunătățită și o documentație bine structurată, Bootstrap 5 reprezintă un instrument indispensabil pentru dezvoltatorii care doresc să creeze aplicații atractive, responsive și eficiente.

2.6.1 Istoria Bootstrap 5

Bootstrap este unul dintre cele mai populare framework-uri front-end, folosit pentru crearea interfețelor web responsive și moderne. Lansat inițial în 2011 de Mark Otto și Jacob Thornton de la Twitter, proiectul a evoluat constant, devenind un standard în industria dezvoltării web. Bootstrap 5, lansat oficial pe 5 mai 2021, reprezintă o etapă semnificativă în istoria acestui framework, datorită numeroaselor îmbunătățiri aduse [27].

Un aspect important al Bootstrap 5 este renunțarea la jQuery, ceea ce face framework-ul mai ușor și mai rapid. Acest lucru a fost posibil prin utilizarea funcționalităților native din JavaScript modern. Sistemul de grilă a fost și el modernizat, utilizând CSS Grid și Flexbox, ceea ce permite crearea unor layout-uri mai complexe și oferă un control mai bun asupra componentelor.

Accesibilitatea a fost o prioritate în această versiune, cu îmbunătățiri semnificative care respectă standardele internaționale și facilitează utilizarea de către persoane cu dizabilități. Suportul pentru CSS personalizat a fost extins, prin introducerea variabilelor CSS, ceea ce face personalizarea temelor mult mai simplă și mai flexibilă.

Designul general al Bootstrap 5 a fost rafinat, eliminându-se componente considerate depășite, cum ar fi card deck-urile. În același timp, au fost introduse îmbunătățiri

care aliniază framework-ul la tendințele moderne de design. Documentația a fost revizuită, devenind mai clară și mai detaliată, astfel încât să sprijine mai bine dezvoltatorii în utilizarea eficientă a acestui instrument [28].

Bootstrap 5 a fost rapid adoptat de comunitatea dezvoltatorilor datorită adaptării sale la standardele contemporane și flexibilității pe care o oferă. Fără dependență de jQuery și cu un set de caracteristici moderne, framework-ul continuă să fie unul dintre cele mai utilizate instrumente pentru dezvoltarea front-end.

2.6.2 Caracteristici Bootstrap 5

Bootstrap 5 aduce îmbunătățiri semnificative care reflectă cerințele moderne de dezvoltare web. Una dintre cele mai importante schimbări este renunțarea la jQuery, ceea ce face framework-ul mai ușor și mai rapid, utilizând funcționalități native din JavaScript modern. De asemenea, sistemul de grilă a fost modernizat prin adoptarea CSS Grid și Flexbox, oferind o flexibilitate mai mare în gestionarea layout-urilor complexe și o mai bună adaptare la diverse rezoluții de ecran [29].

Framework-ul pune un accent mai mare pe accesibilitate, respectând standardele internaționale pentru utilizatorii cu dizabilități. Componentele sunt optimizate pentru cititoarele de ecran și navigarea prin tastatură, iar introducerea variabilelor CSS simplifică personalizarea temelor. Acest lucru permite dezvoltatorilor să creeze design-uri mai coerente și să adapteze cu ușurință aspectul aplicațiilor la nevoile specifice.

Designul general al Bootstrap 5 a fost rafinat, eliminând elementele considerate depășite și introducând îmbunătățiri pentru componente existente, cum ar fi formularele, butoanele și meniurile. În plus, suportul pentru limbile scrise de la dreapta la stânga (RTL) și documentația reorganizată și mai detaliată fac framework-ul accesibil unei audiențe mai largi de dezvoltatori [29].

Performanța generală a fost îmbunătățită prin reducerea dimensiunii framework-ului și eliminarea suportului pentru browserele vechi, cum ar fi Internet Explorer. Aceste schimbări, împreună cu noile funcționalități și o abordare mai modernă, consolidează poziția Bootstrap 5 ca unul dintre cele mai puternice instrumente pentru dezvoltarea de interfețe responsive și atrăgătoare.

2.6.3 Avantajele Bootstrap 5

Bootstrap 5 oferă numeroase avantaje, care îl fac un instrument esențial pentru dezvoltarea aplicațiilor web moderne. Eliminarea dependenței de jQuery îmbunătățește performanța generală și reduce dimensiunea framework-ului, utilizând în schimb funcționalități native din JavaScript modern. În plus, noul sistem de grilă bazat pe CSS Grid și Flexbox oferă flexibilitate ridicată în crearea layout-urilor, permitând gestionarea mai ușoară a structurilor complexe și asigurând un design responsive [29].

Un alt avantaj important este personalizarea simplificată, datorită introducerii variabilelor CSS. Acest lucru le permite dezvoltatorilor să ajusteze culori, spațieri și alte stiluri rapid, fără a modifica codul de bază. De asemenea, framework-ul este mai accesibil, respectând standardele WCAG și optimizând componentele pentru citoarele de ecran și navigarea prin tastatură, ceea ce îl face potrivit pentru o gamă mai largă de utilizatori, inclusiv cei cu dizabilități.

Bootstrap 5 extinde și suportul pentru aplicații internaționale, oferind compatibilitate nativă pentru limbile care se citesc de la dreapta la stânga (RTL). În plus, designul general a fost rafinat, cu actualizări estetice pentru formulare, butoane și alte componente, ceea ce contribuie la crearea unor interfețe moderne și atrăgătoare. Documentația bine organizată și detaliată simplifică adoptarea framework-ului chiar și pentru începători, oferind ghiduri clare și exemple practice [29].

Aceste caracteristici, împreună cu sprijinul unei comunități active și accesul la resurse bogate, fac din Bootstrap 5 o soluție flexibilă, eficientă și ușor de utilizat pentru proiectele front-end de orice dimensiune.

2.6.4 Concluzie

Bootstrap 5 marchează un pas important în evoluția framework-ului, adaptându-se la tendințele și cerințele contemporane ale dezvoltării web. Caracteristici precum flexibilitatea crescută, accesibilitatea îmbunătățită și suportul pentru limbile RTL îl fac potrivit pentru o gamă variată de aplicații, indiferent de complexitate. Cu o comunitate activă și resurse extinse, Bootstrap 5 continuă să fie o alegere de încredere pentru dezvoltatorii front-end, oferind un echilibru ideal între ușurința utilizării și puterea funcționalităților.

3 CERINȚA ȘI SPECIFICAȚIILE PROIECTULUI

Proiectul constă în dezvoltarea unei aplicații web care să gestioneze planurile de învățământ și fișele de disciplină utilizând tehnologii moderne precum Java, Spring Boot, MySQL, Apache POI și Thymeleaf. Aplicația va oferi funcționalități de import, gestionare și vizualizare a datelor pentru trei tipuri de utilizatori: administratori, cadre didactice și studenți.

3.1 CERINȚE FUNCȚIONALE

Aplicația trebuie să ofere funcționalități specifice pentru trei tipuri de utilizatori: administratori, cadre didactice și studenți. Administratorii vor putea importa planuri de învățământ din fișiere Excel, care vor fi validate și mapate într-un model relațional SQL pentru stocare. Acest model va conține generații (de exemplu, 2020–2024 sau 2021–2025), ani universitari și detalii despre discipline, cum ar fi denumirea, numărul de credite, orele alocate și tipurile de activități (curs, seminar, laborator).

Sistemul trebuie să permită administratorilor să copieze fișele de disciplină recente într-o nouă generație la începutul unui nou an universitar. Totodată, aceștia vor avea posibilitatea de a aloca cadre didactice la disciplinele din planurile de învățământ. Cadrele didactice autentificate vor putea vizualiza fișele de disciplină asociate și vor avea opțiunea de a încărca noi fișe, actualizate sau modificate. Studenții vor putea accesa fișele de disciplină organizate pe generație, ciclu de studiu (Licență sau Master) și an de studiu. De asemenea, aceștia vor avea posibilitatea de a descărca fișele în format Word pentru utilizare offline.

3.2 CERINȚE NEFUNCȚIONALE

Aplicația trebuie să fie performantă, astfel încât importul fișierelor Excel și generarea fișelor Word să se realizeze în mai puțin de 3 secunde pentru fișiere standard. Sistemul trebuie să ofere un timp de răspuns sub 2 secunde pentru vizualizarea fișelor de disciplină. Securitatea este un aspect esențial, iar aplicația trebuie să stocheze parolele utilizatorilor în mod criptat. În plus, accesul la date și funcționalități trebuie să fie restricționat pe baza rolurilor utilizatorilor.

Pentru a sprijini creșterea utilizării, aplicația trebuie să fie scalabilă și să poată gestiona simultan cel puțin 500 de utilizatori activi și datele pentru 10 generații complete. Sistemul trebuie să fie disponibil 99,9% din timp și să includă mecanisme de back-up automat pentru baza de date și fișierele încărcate. Interfața aplicației trebuie să fie responsivă, optimizată pentru dispozitive mobile, tablete și desktop-uri, iar navigarea între module să fie intuitivă.

3.3 CERINȚE TEHNICE

Backend-ul aplicației va fi dezvoltat în Java, utilizând Spring Boot pentru implementarea serviciilor REST. Aceste servicii vor comunica cu o bază de date MySQL,

care va gestiona structuri relaționale pentru generații, discipline, cadre didactice, studenți și fișe de disciplină. Frontend-ul aplicației poate fi realizat cu Thymeleaf pentru redare server-side.

Aplicația va folosi biblioteca Apache POI pentru a citi și scrie fișiere Excel, respectiv Word. Fișierele Word asociate fișelor de disciplină vor fi stocate într-un server de fișiere dedicat, legat de baza de date pentru a asigura integrarea datelor.

Pentru portabilitate, sistemul trebuie să fie compatibil cu medii de rulare locale sau cloud și să poată fi implementat într-un container Docker. De asemenea, aplicația va fi configurabilă pentru scalabilitate și suport pentru volume mari de date. Interfața utilizatorului trebuie să fie proiectată astfel încât să ofere o experiență unitară și prietenoasă, indiferent de dispozitivul utilizat.

3.4 FUNCȚIONALITATEA PROIECTULUI

Aplicația web este destinată gestionării eficiente a planurilor de învățământ și a fișelor de disciplină ale unei instituții de învățământ superior. Aceasta este împărțită în module funcționale specifice fiecărui tip de utilizator: Administrator, Cadru Didactic și Student, fiecare având acces la un set de instrumente personalizate în funcție de rol.

Administratorul se poate autentifica în sistem pentru a accesa un panou de control dedicat. Prin intermediul acestuia, poate încărca fișiere Excel care conțin planurile de învățământ. Datele sunt procesate și integrate într-un model relational SQL, organizat pe generații și ani universitari. În plus, administratorul are posibilitatea de a aloca cadre didactice pentru discipline specifice și de a actualiza generațiile, copierea fișelor recente într-o nouă generație fiind automatizată pentru tranziția între ani universitari.

Cadrele didactice pot accesa aplicația printr-un cont personalizat, autenticându-se pentru a vizualiza informațiile relevante. Sistemul le permite să consulte fișele de disciplină asociate materiilor pe care le predau, acestea fiind organizate pe generații și ani universitari. În cazul în care fișele necesită actualizări, cadrele didactice pot încărca versiuni noi ale acestora, care sunt salvate și asociate automat în baza de date, menținând consistența informațiilor.

Studenții au acces la fișele de disciplină ale cursurilor din generația și ciclul lor de studiu, organizate pe ani universitari pentru o navigare facilă. Aplicația oferă posibilitatea de a filtra fișele în funcție de generație, ciclu de studiu (Licență sau Master) și an, astfel încât studenții să găsească rapid informațiile necesare despre cursuri, structura lor și resursele asociate.

În ansamblu, fluxul funcțional al aplicației începe cu importul planurilor de învățământ de către administrator, continuă cu gestionarea și actualizarea fișelor de disciplină de către cadrele didactice și se finalizează cu accesul studenților la informațiile

relevante. Aplicația este concepută pentru a elimina gestionarea manuală a documentelor, a reduce riscul erorilor și a asigura accesibilitate și eficiență pentru toate părțile implicate.

3.5 ANALIZA DE RISC

Din punct de vedere tehnic, există riscul ca tehnologiile utilizate, precum Spring Boot, Apache POI, MySQL și Thymeleaf, să nu fie perfect compatibile, ceea ce ar putea conduce la întârzieri sau funcționalități limitate. Pentru a preveni acest lucru, este esențială realizarea unui prototip pentru testarea integrării componentelor înainte de dezvoltarea completă. De asemenea, procesarea fișierelor mari, cum ar fi cele în format Excel sau Word, poate afecta performanța aplicației. Optimizarea procesării prin utilizarea eficientă a Apache POI și efectuarea de teste de stres poate atenua acest risc. Totodată, erorile în procesarea fișierelor, cauzate de formate sau structuri necorespunzătoare, pot duce la date incomplete sau incorecte. Acest risc poate fi redus prin implementarea unor validări riguroase pentru fișierele încărcate.

Din perspectiva managementului, proiectul poate fi afectat de depășirea termenelor de livrare din cauza complexității tehnice sau a resurselor insuficiente. Stabilirea unui plan detaliat cu etape intermediare clare și monitorizarea constantă a progresului poate diminua aceste riscuri. În cazul lipsei de personal calificat sau a infrastructurii adecvate, alocarea suplimentară de resurse sau externalizarea unor sarcini poate reprezenta o soluție eficientă.

Din punct de vedere al calității, o interfață dificil de utilizat sau neintuitivă poate descuraja utilizatorii și genera feedback negativ. Implicitarea utilizatorilor finali în procesul de proiectare și efectuarea de teste de utilizabilitate pot contribui la crearea unei experiențe mai bune. De asemenea, erorile neașteptate în funcționalitate pot afecta negativ utilizatorii și pot cauza pierderi de date. Testarea riguroasă, atât unitară, cât și de integrare și end-to-end, înainte de lansare, este esențială pentru reducerea acestui risc.

Din perspectiva operațională, lipsa unui suport tehnic post-lansare poate întârzi rezolvarea problemelor identificate, ceea ce poate duce la nemulțumirea utilizatorilor. Un plan de suport tehnic bine definit și alocarea de resurse pentru menenanță sunt măsuri necesare. În plus, dependența de infrastructură instabilă, precum servere sau rețele, poate face aplicația inaccesibilă. Utilizarea unei infrastructuri redundante și monitorizarea performanței pot minimiza acest risc.

Din punct de vedere legal, nerespectarea reglementărilor privind protecția datelor, cum ar fi GDPR, poate atrage sancțiuni financiare și daune reputaționale. Proiectarea aplicației conform reglementărilor de confidențialitate și protecția datelor reprezintă o măsură preventivă esențială.

În concluzie, succesul proiectului depinde de capacitatea de a identifica și gestiona riscurile în mod proactiv. Prin implementarea unor procese riguroase de planificare, testare și monitorizare, impactul riscurilor poate fi redus semnificativ.

4 PROIECTARE

4.1 SCHEMA BLOC

Schema arhitecturală a aplicației web din figura 4 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.

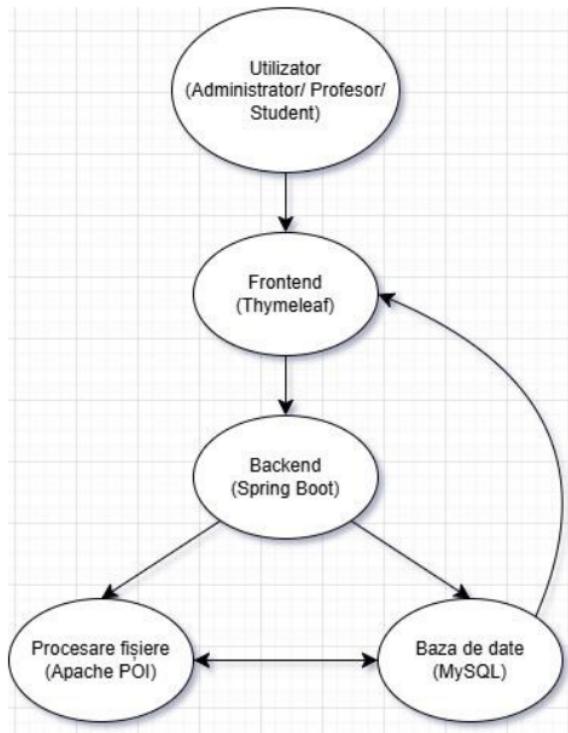


Figura 4 - Schema aplicației web

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 din figura 5 descrie structura statică a sistemului, evidențierând principalele clase și relațiile dintre ele.

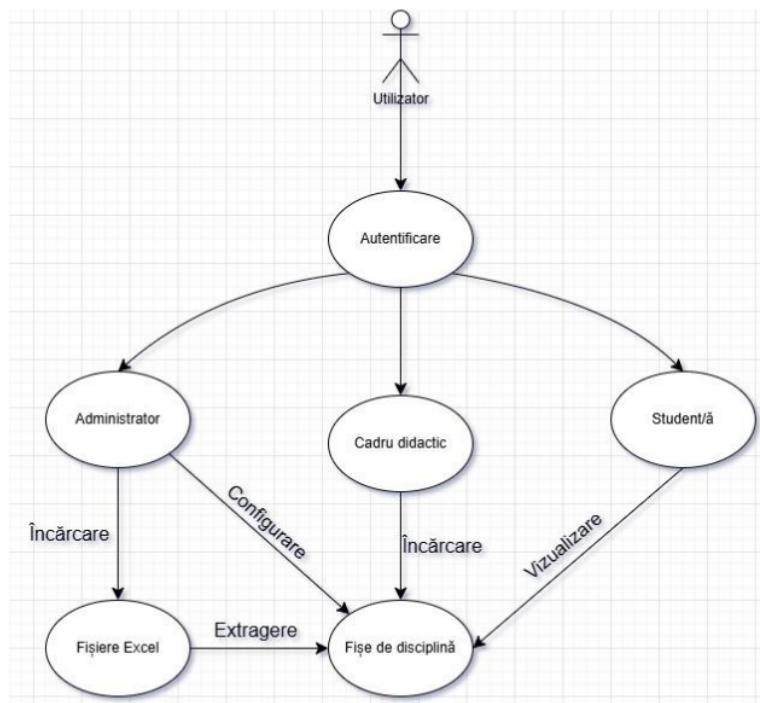


Figura 5 - Schema UML

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 detalile 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, ciclu 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 compozitie 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**, permitând 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.

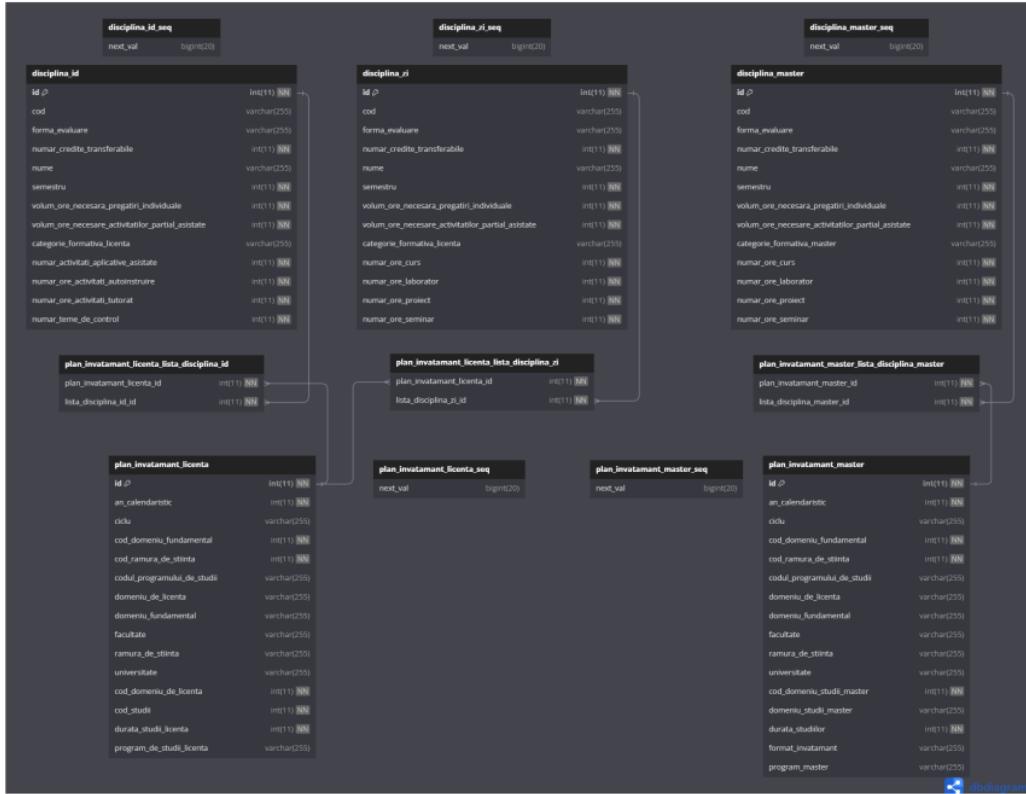


Figura 6 – Schema bazei de date

Tabelele disciplinelor din figura 6 conțin informații despre disciplinele din programele de licență și masterat, respectiv. Toate includ atrbute precum identificatorul unic al disciplinei (id), codul disciplinei (cod), forma de evaluare (de exemplu, examen sau colocviu), numărul de credite transferabile (nc), numele disciplinei și semestrul în care se desfășoară. De asemenea, sunt incluse detalii despre orele necesare, cum ar fi orele de curs, laborator sau proiect. Tabelul *discipline_id* (învățământ la distanță) este specific programele de licență și conține detalii suplimentare precum numărul de ore de activități asistate sau numărul de teme de control. În mod similar, tabelul *discipline_master* este destinat programele de masterat și conține informații adaptate acestora, mai specific în categoria formativă.

Tabele planurilor de învățământ reprezintă planurile de învățământ pentru programele de licență și masterat. Fiecare rând din aceste tabele descrie un plan specific, identificat printr-un ID unic (id). Sunt incluse informații despre anul calendaristic în care este valabil planul (an_calendaristic), codul unic al planului (cod_studii), codul și denumirea domeniului fundamental, codul ramurii de știință, durata studiilor (în ani), precum și informații despre facultatea și universitatea care oferă programul.

Legătura dintre planurile de învățământ și disciplinele incluse este realizată prin tabelele de asociere `plan_invatamant_licenta_lista_disciplina_id` și `plan_invatamant_master_lista_disciplina_master`. Aceste tabele conțin referințe către planurile de învățământ și disciplinele corespunzătoare, asigurând astfel că fiecare plan este asociat cu disciplinele corecte.

Modelul de bază de date poate fi utilizat pentru a gestiona și organiza planurile de studiu și disciplinele asociate. Administratorii pot adăuga sau modifica planurile și disciplinele. Profesorii pot consulta informații despre disciplinele pe care le predau, iar studenții pot accesa detalii despre disciplinele incluse în planurile lor de învățământ.

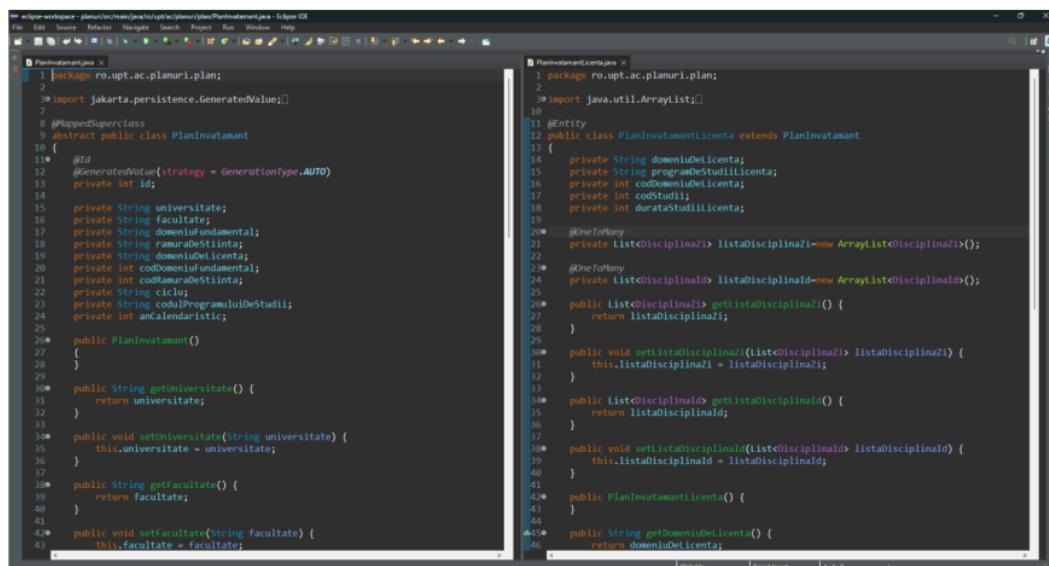
5 IMPLEMENTAREA

5.1 Planuri

Proiectul este o aplicație bazată pe **Spring Boot**, care gestionează planurile de învățământ pentru studii de licență și masterat. Aceasta utilizează **Spring Data JPA** pentru interacțiunea cu baza de date și **Thymeleaf** pentru generarea interfeței utilizatorului.

5.1.1 Clasa PlanInvatamant

Clasa PlanInvatamant din figura 7 este abstractă și reprezintă baza comună pentru entitățile PlanInvatamantLicenta și PlanInvatamantMaster. Aceasta conține proprietăți generale, precum numele universității, facultatea, domeniul fundamental, ciclul de studii, codurile asociate și anul calendaristic. Este decorată cu anotarea `@MappedSuperclass`, ceea ce înseamnă că nu este mapată direct într-o tabelă din baza de date, dar proprietățile sale sunt moștenite de clasele derivate. `@Id` și `@GeneratedValue(strategy = GenerationType.AUTO)` indică faptul că valoarea pentru id va fi generată automat de către baza de date, folosind o strategie specificată (AUTO este standard, baza de date decide cum să genereze valorile).



```

1 package ro.upt.ac.planuri.plan;
2
3 import jakarta.persistence.GeneratedValue;
4
5 @MappedSuperclass
6 abstract public class PlanInvatamant {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.AUTO)
10    private int id;
11
12    private String universitate;
13    private String facultate;
14    private String domeniulFundamental;
15    private String domeniulStiinta;
16    private String domeniulLicenta;
17    private int codDomeniuFundamental;
18    private int codDomeniuStiinta;
19    private String ciclu;
20    private String codulProgramuluiDeStudii;
21    private int anCalendaristic;
22
23    public PlanInvatamant() {
24    }
25
26    public String getUniversitate() {
27        return universitate;
28    }
29
30    public void setUniversitate(String universitate) {
31        this.universitate = universitate;
32    }
33
34    public String getFacultate() {
35        return facultate;
36    }
37
38    public void setFacultate(String facultate) {
39        this.facultate = facultate;
40    }
41
42    public PlanInvatamantLicenta() {
43    }
44
45    public String getDomeniuDeLicenta() {
46        return domeniulLicenta;
47    }
48}

```

```

1 package ro.upt.ac.planuri.plan;
2
3 import java.util.ArrayList;
4
5 @Entity
6 public class PlanInvatamantLicenta extends PlanInvatamant {
7
8     private String domeniulDeLicenta;
9     private String programDeStudiiLicenta;
10    private int codDomeniuDeLicenta;
11    private int codStudii;
12    private int durataStudiiLicenta;
13
14    @OneToMany
15    private List<DisciplinaZi> listaDisciplinaZi=new ArrayList<DisciplinaZi>();
16
17    @OneToMany
18    private List<DisciplinaId> listaDisciplinaId=new ArrayList<DisciplinaId>();
19
20    public List<DisciplinaZi> getListaDisciplinaZi() {
21        return listaDisciplinaZi;
22    }
23
24    public void setListaDisciplinaZi(List<DisciplinaZi> listaDisciplinaZi) {
25        this.listaDisciplinaZi = listaDisciplinaZi;
26    }
27
28    public List<DisciplinaId> getListaDisciplinaId() {
29        return listaDisciplinaId;
30    }
31
32    public void setListaDisciplinaId(List<DisciplinaId> listaDisciplinaId) {
33        this.listaDisciplinaId = listaDisciplinaId;
34    }
35
36    public PlanInvatamantLicenta() {
37    }
38
39    public String getDomeniuDeLicenta() {
40        return domeniulDeLicenta;
41    }
42}

```

Figura 7 – PlanInvatamant.java și PlanInvatamantLicenta.java

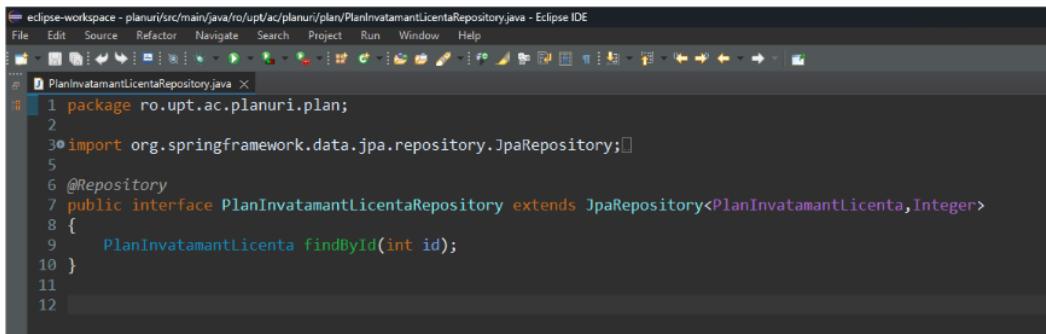
Clasa PlanInvatamant extinde clasa PlanInvatamant și adaugă atribute specifice pentru planurile de licență, cum ar fi programul de studii, durata studiilor și listele disciplinelor asociate. Relațiile dintre planurile de licență și discipline sunt definite prin anotări `@OneToMany`, care leagă planurile cu entitățile DisciplinaZi și DisciplinaId, iar

@Entity indică faptul că această clasă este o entitate JPA (Java Persistence API) și va fi mapată la o tabelă în baza de date.

Similară clasei PlanInvatamantLicenta, PlanInvatamantMaster extinde PlanInvatamant și definește atributele necesare pentru planurile de master, cum ar fi programul de studii și durata. Aceasta include o relație @OneToMany pentru gestionarea disciplinelor asociate.

5.1.2 Repository și Controllere

Repository-urile sunt componente care gestionează operațiile CRUD (Create, Read, Update, Delete) pe entități, utilizând funcționalitățile predefinite ale interfeței JpaRepository. Acestea asigură acces direct la baza de date fără a fi nevoie să scriem manual interogări SQL.



```
eclipse-workspace - planuri/src/main/java/ro/upt/ac/planuri/plan/PlanInvatamantLicentaRepository.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
...
PlanInvatamantLicentaRepository.java ×
1 package ro.upt.ac.planuri.plan;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 @Repository
6 public interface PlanInvatamantLicentaRepository extends JpaRepository<PlanInvatamantLicenta, Integer> {
7     PlanInvatamantLicenta findById(int id);
8 }
9
10
11
12
```

Figura 8 – PlanInvatamantLicentaRepository.java

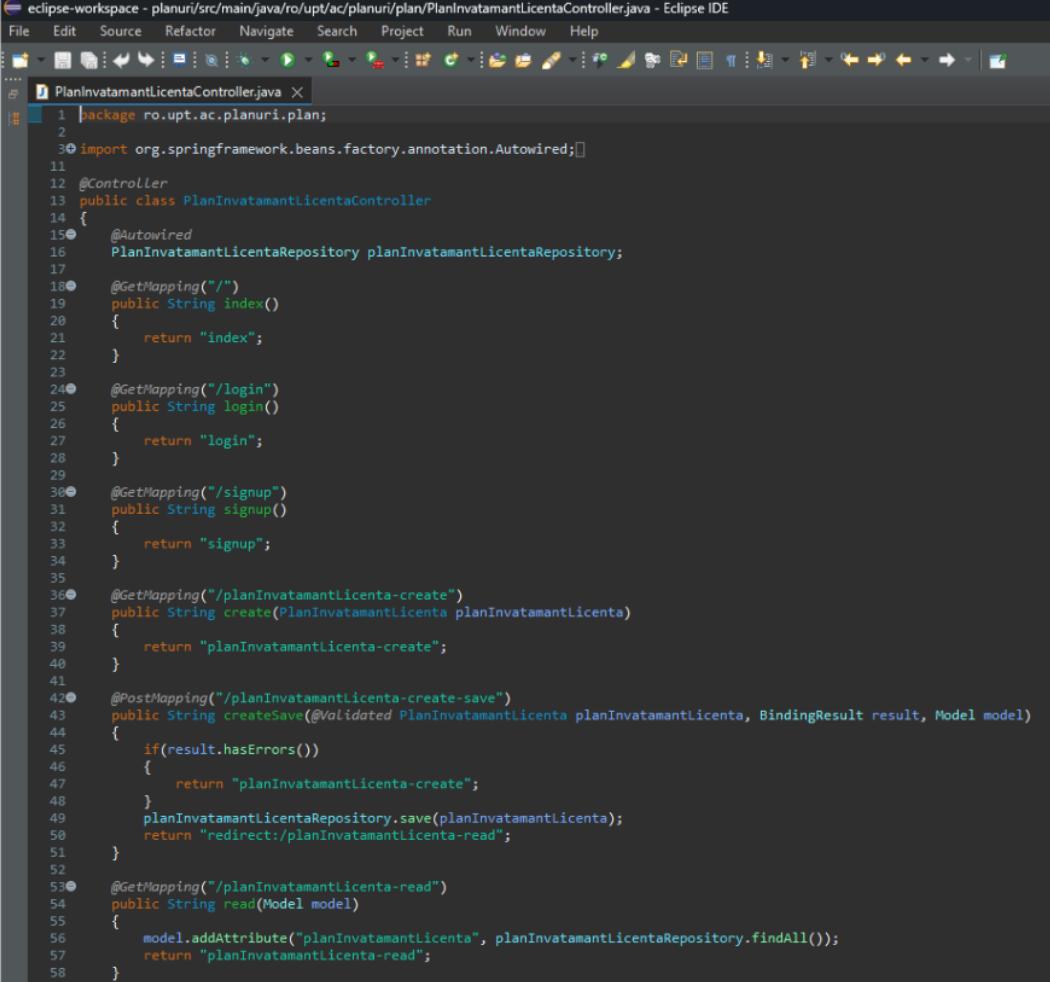
Acest repository din figura 8 este responsabil pentru gestionarea entității PlanInvatamantLicenta. Pe lângă metodele standard (save, findById, findAll, etc.), include o metodă personalizată care găsește un plan de învățământ pe baza ID-ului. PlanInvatamantMaster funcționează în mod similar cu PlanInvatamantLicentaRepository.

Metoda findById(int id) caută un plan de învățământ de licență în tabel, utilizând un identificator (id) și returnează o instanță.

Controller-ele sunt responsabile pentru gestionarea cererilor HTTP și pentru legătura dintre interfața utilizatorului și baza de date. Acestea folosesc repository-urile pentru operații pe date și returnează pagini HTML generate de Thymeleaf.

Controllerul este marcat cu anotarea @Controller, indicând că gestionează cererile HTTP și returnează răspunsuri sub formă de pagini HTML. Pentru a interacționa cu baza de date, folosește un repository injectat automat prin @Autowired, care permite accesarea metodelor standard oferite de JpaRepository. Acest controller este dedicat gestionării

planurilor de licență. Include funcționalități pentru crearea, citirea, editarea și ștergerea planurilor.



The screenshot shows the Eclipse IDE interface with the file 'PlanInvatamantLicentaController.java' open. The code defines a controller class for managing plans. It includes methods for index, login, signup, and creating a new plan. The 'createSave' method handles form validation and saves the new plan to the database. The 'read' method retrieves all plans from the database and adds them to a model for display.

```
1 package ro.upt.ac.planuri.plan;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Controller
6 public class PlanInvatamantLicentaController
7 {
8     @Autowired
9     PlanInvatamantLicentaRepository planInvatamantLicentaRepository;
10
11     @GetMapping("/")
12     public String index()
13     {
14         return "index";
15     }
16
17     @GetMapping("/login")
18     public String login()
19     {
20         return "login";
21     }
22
23     @GetMapping("/signup")
24     public String signup()
25     {
26         return "signup";
27     }
28
29     @GetMapping("/planInvatamantLicenta-create")
30     public String create(PlanInvatamantLicenta planInvatamantLicenta)
31     {
32         return "planInvatamantLicenta-create";
33     }
34
35     @PostMapping("/planInvatamantLicenta-create-save")
36     public String createSave(@Validated PlanInvatamantLicenta planInvatamantLicenta, BindingResult result, Model model)
37     {
38         if(result.hasErrors())
39         {
40             return "planInvatamantLicenta-create";
41         }
42         planInvatamantLicentaRepository.save(planInvatamantLicenta);
43         return "redirect:/planInvatamantLicenta-read";
44     }
45
46     @GetMapping("/planInvatamantLicenta-read")
47     public String read(Model model)
48     {
49         model.addAttribute("planInvatamantLicenta", planInvatamantLicentaRepository.findAll());
50         return "planInvatamantLicenta-read";
51     }
52 }
```

Figura 9 – PlanInvatamantLicentaController.java

Crearea unui plan de învățământ este gestionată prin două rute. Ruta GET /planInvatamantLicenta-create afișează un formular pentru introducerea datelor unui nou plan, iar ruta POST /planInvatamantLicenta-create-save salvează datele completeate în formular în baza de date. Dacă există erori de validare, utilizatorul rămâne pe pagina de creare.

Pentru citirea planurilor de învățământ, ruta GET /planInvatamantLicenta-read returnează toate planurile din baza de date și le afișează pe pagina planInvatamantLicenta-

read.html. Ruta GET /planInvatamantLicenta-view/{id} permite vizualizarea detaliată a unui plan specific, identificat prin ID.

Editarea unui plan este realizată prin două rute. Ruta GET /planInvatamantLicenta-edit/{id} afișează formularul cu datele existente ale planului pentru editare, iar ruta POST /planInvatamantLicenta-update/{id} salvează modificările făcute. În cazul erorilor de validare, utilizatorul rămâne pe pagina de editare.

Stergerea unui plan este gestionată de ruta GET /planInvatamantLicenta-delete/{id}, care găsește planul specific după ID și îl elimină din baza de date.

PlanInvatamantMasterController oferă aceleași funcționalități precum cel pentru licență, dar este adaptat pentru gestionarea planurilor de master. Structura și rutele sunt aproape identice, diferența fiind entitatea gestionată (PlanInvatamantMaster).

5.1.3 Sumar

Arhitectura de mai sus reprezintă o soluție bine structurată pentru gestionarea planurilor de învățământ, utilizând un ecosistem modern de tehnologii Java. Prin arhitectura sa modulară, bazată pe Spring Boot, Spring Data JPA și Thymeleaf, proiectul oferă o bază solidă pentru dezvoltare și extindere ulterioară. Funcționalitățile implementate acoperă toate cerințele esențiale pentru gestionarea planurilor de licență și master, incluzând operații CRUD și relații bine definite între entități.

5.2 Discipline

Acest proiect are ca scop dezvoltarea unui sistem modular și scalabil pentru gestionarea disciplinelor din cadrul unui mediu educațional, cu accent pe flexibilitate, standardizare și eficiență. Arhitectura proiectului este construită în jurul unor enum-uri și clase care definesc structura datelor și logica aferentă, adaptate diverselor tipuri de discipline de învățământ: învățământ la zi, învățământ la distanță licență și masterat.

5.2.1 Clasa Discipline și Enum-uri

Structura entităților este bine definită prin utilizarea unei clase abstracte de bază, Disciplina (figura 10), folosită pentru a defini o entitate generică ce descrie o disciplină universitară într-o aplicație bazată pe JPA (Java Persistence API) care conține proprietățile comune, cum ar fi numele, codul, numărul de credite și forma de evaluare. Anotarea `@MappedSuperclass` indică faptul că *Disciplina* este o clasă de bază care nu va fi mapată direct la o tabelă în baza de date. `@Id` definește id ca fiind cheia primară a entității în contextul JPA, iar `@GeneratedValue(strategy = GenerationType.AUTO)` specifică modul în care se generează automat valorile pentru id. Strategia AUTO lasă JPA să aleagă metoda potrivită bazată pe baza de date utilizată.

```

Disciplina.java
1 package ro.upt.ac.planuri.disciplina;
2
3 import jakarta.persistence.GeneratedValue;
4
5 @Entity
6 abstract public class Disciplina
7 {
8     @Id
9     @GeneratedValue(strategy = GenerationType.AUTO)
10    private int id;
11
12    private String nume;
13    private String cod;
14    private int numarCrediteTransferabile;
15    private String formaDeValoare;
16    private int volumOreNecesareActivitatilorPartialAsigurate;
17    private int volumOreNecesareARezarcitiiIndividualizate;
18    private int semestru;
19
20    public Disciplina()
21    {
22    }
23
24    public int getId()
25    {
26        return id;
27    }
28
29    public void setId(int id)
30    {
31        this.id = id;
32    }
33
34    public String getName()
35    {
36        return nume;
37    }
38
39    public void setName(String nume)
40    {
41        this.nume = nume;
42    }
43
44    public String getCod()
45    {
46
DisciplinaZi.java
1 package ro.upt.ac.planuri.disciplina;
2
3 import jakarta.persistence.Entity;
4
5 @Entity
6 public class DisciplinaZi extends Disciplina
7 {
8     private String categorieFormativaLicenta;
9     private int numarOreCurs;
10    private int numarOreSeminar;
11    private int numarOreLaborator;
12    private int numarOreProiect;
13
14    public DisciplinaZi()
15    {
16    }
17
18    public int getNumarOreCurs()
19    {
20        return numarOreCurs;
21    }
22
23    public void setNumarOreCurs(int numarOreCurs)
24    {
25        this.numarOreCurs = numarOreCurs;
26    }
27
28    public int getNumarOreSeminar()
29    {
30        return numarOreSeminar;
31    }
32
33    public void setNumarOreSeminar(int numarOreSeminar)
34    {
35        this.numarOreSeminar = numarOreSeminar;
36    }
37
38    public int getNumarOreLaborator()
39    {
40        return numarOreLaborator;
41    }
42
43    public void setNumarOreLaborator(int numarOreLaborator)
44    {
45        this.numarOreLaborator = numarOreLaborator;
46
TFormalEvaluare.java
1 package ro.upt.ac.planuri.disciplina;
2
3 public enum TFormalEvaluare
4 {
5     E("Examen","E"),
6     D("Evaluare distribuita","D"),
7     C("Colocviu","C"),
8     P("Proiect autonom cu examinare ca si in caz de proiect","P_D");
9
10    private String numeLung;
11    private String numeScurt;
12
13    TFormalEvaluare(String numeLung, String numeScurt)
14    {
15        this.numeLung = numeLung;
16        this.numeScurt = numeScurt;
17    }
18
19    public String getNumeLung()
20    {
21        return numeLung;
22    }
23
24    public String getNumeScurt()
25    {
26        return numeScurt;
27    }
28
29
30

```

Figura 10 – Disciplina.java, DisciplinaZi.java și TFormalEvaluare.java

Clasele derivate, cum ar fi DisciplinaZi, Disciplinald și DisciplinaMaster, extind această bază pentru a adăuga specificații suplimentare. De exemplu, DisciplinaZi adaugă informații specifice învățământului de zi, cum ar fi numărul de ore de curs, seminar sau laborator. Disciplinald este concepută pentru învățământul la distanță, înlocuind orele de curs și seminar cu ore de autoinstruire și tutorat. DisciplinaMaster, pe de altă parte, include categorii formative specifice masteranzilor și ore de proiect, fiind specifică programelor de masterat. Acest design oferă flexibilitate și posibilitatea de a extinde ușor sistemul pentru alte tipuri de discipline.

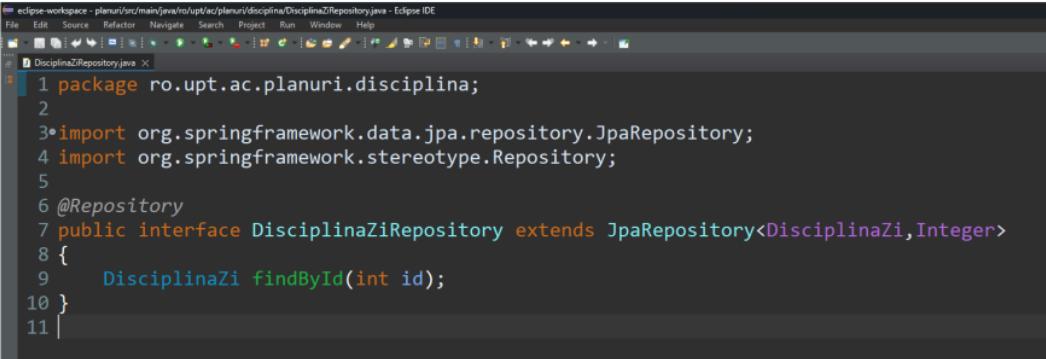
Anotarea `@Entity` declară că aceste clase sunt entități JPA și că instanțele lor vor fi mapate la o tabelă din baza de date. DisciplinaZi, Disciplinald și DisciplinaMaster moștenesc toate atributele și metodele din clasa abstractă Disciplina.

Enum-urile TCategorieFormativaLicenta, TCategorieFormativaMaster și TFormalEvaluare sunt utilizate pentru a standardiza și organiza informațiile despre categoriile formative și formele de evaluare ale disciplinelor. TCategorieFormativaLicenta gestionează categoriile formative pentru disciplinele programelor de licență, cum ar fi "Disciplina complementară" sau "Disciplina fundamentală". La rândul său, TCategorieFormativaMaster este dedicat disciplinelor de masterat, inclusiv categorii precum "Disciplina de aprofundare" sau "Disciplina de sinteză". TFormalEvaluare definește metodele prin care studenții sunt evaluați, de exemplu, examen, colocviu sau proiect autonom. Aceste enum-uri asigură consistența datelor, faciliteză validarea și permit o afișare clară a informațiilor.

Atributele enum-ului sunt numeLung, care oferă descrierea completă a categoriei (ex. "Disciplina complementara"), respectiv numeScurt, care oferă un identificator prescurtat pentru categoria respectivă (ex. "DC"). Aceste atribute sunt utilizate pentru a asocia fiecărei constante o descriere mai detaliată, ceea ce este util în aplicații care implică afișarea de informații către utilizatori.

5.2.2 Repository și Controllere

Pentru interacțiunea cu baza de date, fiecare entitate are asociat un repository dedicat, care extinde JpaRepository, un component Spring care gestionează operațiunile de acces la date. Aceste repository-uri, cum ar fi DisciplinaZiRepository (figura 11), DisciplinaleRepository și DisciplinaMasterRepository, oferă metode standard pentru operațiuni CRUD, eliminând necesitatea scrierii codului SQL manual. Ele pot fi utilizate în cadrul serviciilor și controllerelor pentru a gestiona logic operațiunile asupra datelor.



```
eclipse-workspace - planuri/src/main/java/ro/upt/ac/planuri/disciplina/DisciplinaZiRepository.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DisciplinaZiRepository.java
1 package ro.upt.ac.planuri.disciplina;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4 import org.springframework.stereotype.Repository;
5
6 @Repository
7 public interface DisciplinaZiRepository extends JpaRepository<DisciplinaZi, Integer>
8 {
9     DisciplinaZi findById(int id);
10 }
11 |
```

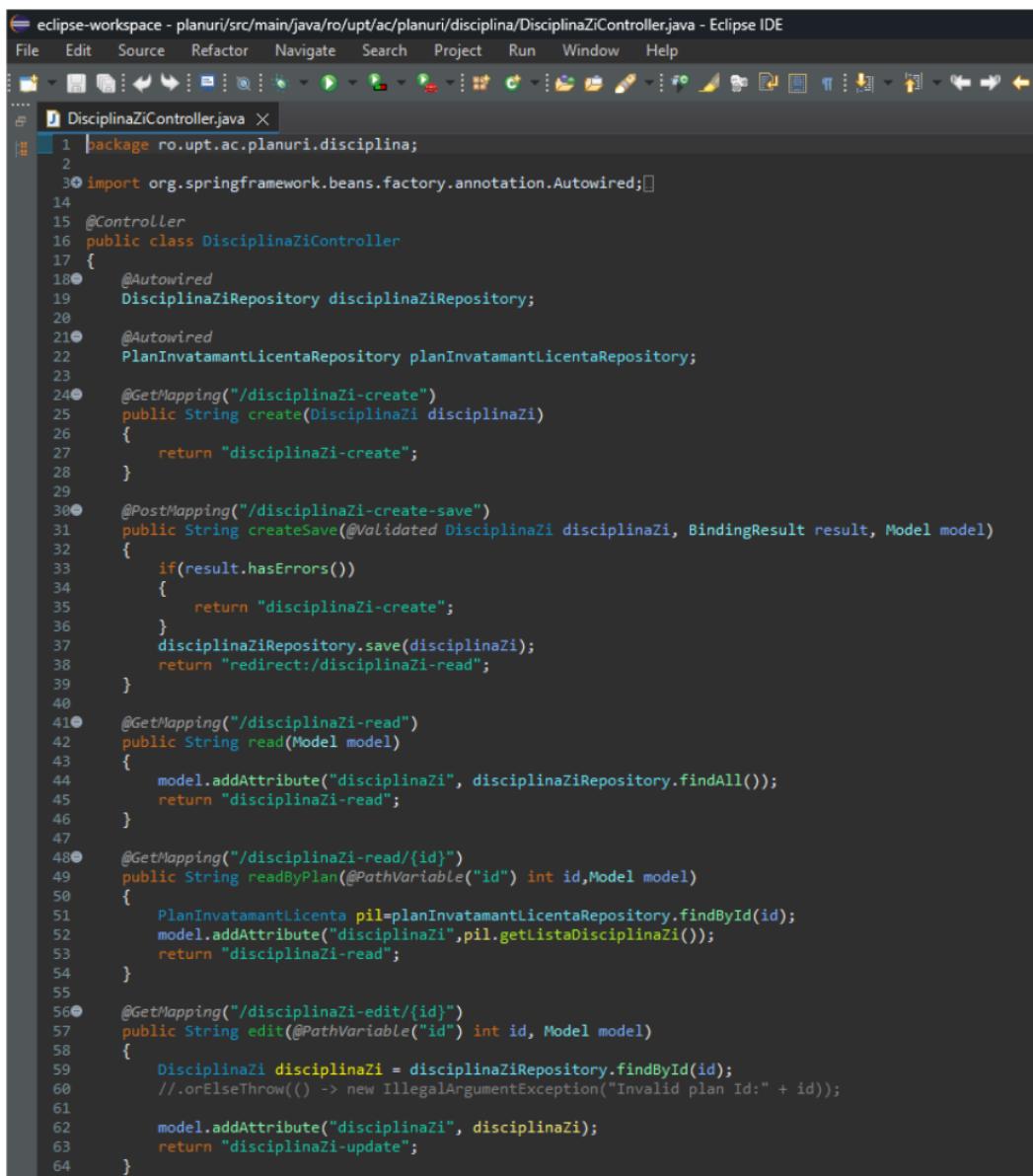
Figura 11 – DisciplinaZiRepository.java

Metoda findById(int id) este interpretată automat de Spring Data JPA, astfel încât nu este nevoie de implementare explicită. Metoda generează o interogare SQL care caută în baza de date o entitate cu id corespunzător.

Controllerele sunt componente care fac legătura între utilizator și sistem. Fiecare tip de disciplină are un controller dedicat. De exemplu, DisciplinaZiController (figura 12) permite crearea, citirea, actualizarea și ștergerea disciplinelor din învățământul de zi. Aceste controllere utilizează repository-urile pentru a accesa baza de date și şablonane HTML pentru a afișa informațiile către utilizatori. În mod similar, DisciplinaleController gestionează disciplinele învățământului la distanță, iar DisciplinaMasterController este responsabil pentru disciplinele programelor de masterat. Această organizare facilitează gestionarea eficientă a cerințelor specifice fiecărui tip de disciplină.

Anotarea `@Controller` declară că această clasă este un controller Spring MVC. Controlează fluxul cererilor HTTP și returnează şablonane HTML pentru interfața utilizator.

Atributele `@Autowired` sunt injectate automat pentru a oferi acces la repository-ile respective care gestionează persistarea entităților.



```

eclipse-workspace - planuri - src/main/java/ro/upt/ac/planuri/disciplina/DisciplinaZiController.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
DisciplinaZiController.java X
1 package ro.upt.ac.planuri.disciplina;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @Controller
6 public class DisciplinaZiController
7 {
8     @Autowired
9     DisciplinaZiRepository disciplinaZiRepository;
10
11     @Autowired
12     PlanInvatamantLicentaRepository planInvatamantLicentaRepository;
13
14     @GetMapping("/disciplinaZi-create")
15     public String create(DisciplinaZi disciplinaZi)
16     {
17         return "disciplinaZi-create";
18     }
19
20     @PostMapping("/disciplinaZi-create-save")
21     public String createSave(@Validated DisciplinaZi disciplinaZi, BindingResult result, Model model)
22     {
23         if(result.hasErrors())
24         {
25             return "disciplinaZi-create";
26         }
27         disciplinaZiRepository.save(disciplinaZi);
28         return "redirect:/disciplinaZi-read";
29     }
30
31     @GetMapping("/disciplinaZi-read")
32     public String read(Model model)
33     {
34         model.addAttribute("disciplinaZi", disciplinaZiRepository.findAll());
35         return "disciplinaZi-read";
36     }
37
38     @GetMapping("/disciplinaZi-read/{id}")
39     public String readByPlan(@PathVariable("id") int id, Model model)
40     {
41         PlanInvatamantlicenta pil=planInvatamantLicentaRepository.findById(id);
42         model.addAttribute("disciplinaZi", pil.getListaDisciplinaZi());
43         return "disciplinaZi-read";
44     }
45
46     @GetMapping("/disciplinaZi-edit/{id}")
47     public String edit(@PathVariable("id") int id, Model model)
48     {
49         DisciplinaZi disciplinaZi = disciplinaZiRepository.findById(id);
50         //orElseThrow(() -> new IllegalArgumentException("Invalid plan Id:" + id));
51         model.addAttribute("disciplinaZi", disciplinaZi);
52         return "disciplinaZi-update";
53     }
54
55 }

```

Figura 12 – DisciplinaZiController.java

Prima metodă, `create`, asociată cu ruta GET `/disciplinaZi-create`, afișează un formular pentru crearea unei noi discipline. Metoda returnează şablonul HTML `disciplinaZi-`

create, care permite utilizatorului să introducă informațiile necesare despre disciplină. Formularul este utilizat ulterior pentru trimiterea datelor către metoda de salvare.

Metoda `createSave`, asociată cu ruta POST `/disciplinaZi-create-save`, primește datele complete de utilizator în formularul de creare. Aceasta folosește validarea prin `@Validated` pentru a verifica dacă datele respectă constrângările definite în clasa entitate. În cazul în care sunt detectate erori de validare, metoda redirecționează utilizatorul înapoi la formularul de creare (`disciplinaZi-create`). Dacă validarea este reușită, disciplina este salvată în baza de date, iar utilizatorul este redirecționat către lista tuturor disciplinelor.

Metoda `read`, asociată cu ruta GET `/disciplinaZi-read`, obține toate disciplinele existente din baza de date utilizând metoda `findAll()` din repository-ul `DisciplinaZiRepository`. Disciplinele extrase sunt adăugate în model sub atributul `disciplinaZi`, iar metoda returnează şablonul HTML `disciplinaZi-read` pentru afișarea listei.

O altă metodă, `readByPlan`, asociată cu ruta GET `/disciplinaZi-read/{id}`, cauță un plan de învățământ licență (`PlanInvatamantLicenta`) pe baza ID-ului specificat. Disciplinele asociate aceluiași plan sunt extrase și adăugate în model sub atributul `disciplinaZi`, după care şablonul `disciplinaZi-read` este returnat pentru afișare.

Metoda `edit`, asociată cu ruta GET `/disciplinaZi-edit/{id}`, este folosită pentru editarea unei discipline existente. Aceasta cauță disciplina cu ID-ul specificat folosind metoda `findById()` din repository. Disciplina este adăugată în model sub atributul `disciplinaZi`, iar metoda returnează şablonul `disciplinaZi-update`, care afișează formularul pentru editare.

În continuare, metoda `update`, asociată cu ruta POST `/disciplinaZi-update/{id}`, primește datele actualizate introduse de utilizator în formularul de editare. Aceste date sunt validate folosind `@Validated`. Dacă sunt detectate erori, utilizatorul este redirecționat înapoi la formularul de editare. În cazul în care validarea este reușită, metoda actualizează disciplina în baza de date și redirecționează utilizatorul către lista disciplinelor.

În cele din urmă, metoda `delete`, asociată cu ruta GET `/disciplinaZi-delete/{id}`, gestionează ștergerea unei discipline din baza de date. Aceasta găsește disciplina pe baza ID-ului specificat utilizând metoda `findById()` și o șterge din baza de date folosind metoda `delete()`. După ștergere, utilizatorul este redirecționat către lista disciplinelor.

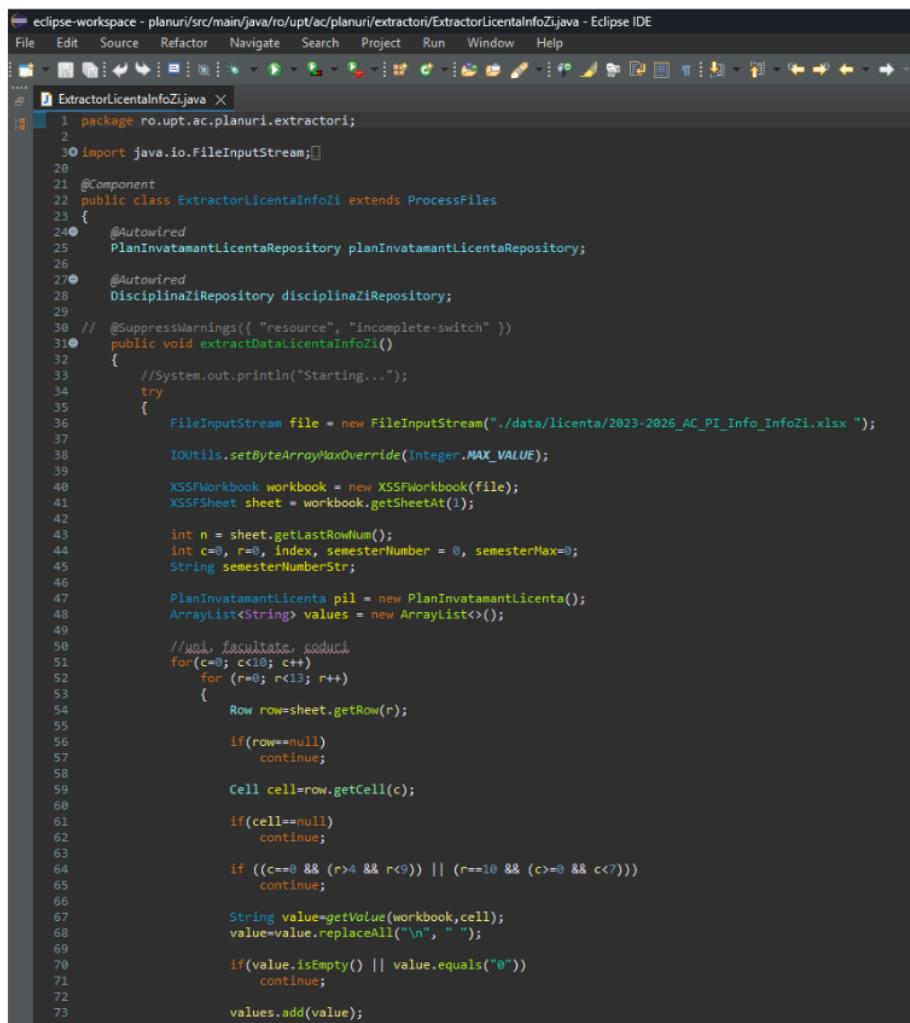
5.2.3 Sumar

Arhitectura proiectului este modulară, scalabilă și bine structurată. Utilizarea unei clase abstracte și a enum-urilor permite reutilizarea și standardizarea codului, reducând redundanța și asigurând consistența. Fiecare tip de disciplină este tratat individual, ceea ce permite extinderea sistemului fără a afecta funcționalitatea existentă. Repository-urile simplifică accesul la date, iar controlerlele oferă o interfață clară pentru utilizatori. În ansamblu, acest design modular și flexibil asigură o bază solidă pentru dezvoltarea ulterioară a proiectului. Dacă sunt necesare detalii suplimentare sau integrarea unei noi

funcționalități, acestea pot fi adăugate cu ușurință datorită arhitecturii clare și bine organizate.

5.3 Extractoare

Acest set de clase Java este conceput pentru extragerea și salvarea datelor din fișiere Excel ce conțin planuri de învățământ. Biblioteca **Apache POI** este utilizată pentru manipularea fișierelor Excel, în timp ce framework-ul **Spring** gestionează componentele aplicației și operațiunile de persistare a datelor în baza de date.



```
eclipse-workspace - planuri/src/main/java/ro/upt/ac/planuri/extractor/ExtractorLicentainfoZi.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
ExtractorLicentainfoZi.java
1 package ro.upt.ac.planuri.extractori;
2
3 import java.io.FileInputStream;
4
5
6 @Component
7 public class ExtractorLicentainfoZi extends ProcessFiles
8 {
9     @Autowired
10     PlanInvatamantLicentaRepository planInvatamantLicentaRepository;
11
12     @Autowired
13     DisciplinaZiRepository disciplinaZiRepository;
14
15     // @SuppressWarnings({ "resource", "incomplete-switch" })
16     public void extractDataLicentainfoZi()
17     {
18         //System.out.println("Starting...");
19         try
20         {
21             FileInputStream file = new FileInputStream("./data/licenta/2023-2026_AC_PI_InfoZi.xlsx");
22
23             IOUtils.setByteArrayMaxOverride(Integer.MAX_VALUE);
24
25             XSSFWorkbook workbook = new XSSFWorkbook(file);
26             XSSFSheet sheet = workbook.getSheetAt(1);
27
28             int n = sheet.getLastRowNum();
29             int c=0, r=0, index, semesterNumber = 0, semesterMax=0;
30             String semesterNumberStr;
31
32             PlanInvatamantLicenta pil = new PlanInvatamantLicenta();
33             ArrayList<String> values = new ArrayList<>();
34
35             //uri, facultate, coduri
36             for(c=0; c<10; c++)
37                 for (r=0; r<15; r++)
38                 {
39                     Row row=sheet.getRow(r);
40
41                     if(row==null)
42                         continue;
43
44                     Cell cell=row.getCell(c);
45
46                     if(cell==null)
47                         continue;
48
49                     if ((c==0 && (r>4 && r<9)) || (r==10 && (c>0 && c<7)))
50                         continue;
51
52                     String value=getValue(workbook,cell);
53                     value=value.replaceAll("\n", " ");
54
55                     if(value.isEmpty() || value.equals("0"))
56                         continue;
57
58                     values.add(value);
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73 }
```

Figura 13 – ExtractorLicentainfoZi.java

Fiecare clasă extractoare, precum cea din figura 13, este o componentă Spring și procesează fișierele Excel asociate unui anumit program de studiu. Astfel, există clase distincte pentru planurile de licență generale, programele de master și diferite specializări precum Informatică de zi, Informatică la distanță sau Calculatoare - Engleză.

Obiectele principale gestionate de aceste clase includ entități pentru planurile de învățământ și discipline. Pentru licență, entitatea principală este PlanInvatamantLicenta, iar disciplinele sunt reprezentate prin DisciplinaZi (pentru Învățământ de zi) și DisciplinaDistanta (pentru Învățământ la distanță). În cazul planurilor de master, se utilizează entitățile PlanInvatamantMaster și DisciplinaMaster, care reflectă specificațiile avansate ale acestui nivel de studiu.

Clasele includ metode pentru procesarea fișierelor Excel și pentru extragerea datelor relevante. Metoda principală de procesare a fișierelor în fiecare clasă definește o listă de fișiere și apelează o metodă auxiliară pentru procesarea individuală a fiecărui fișier. În această etapă, fișierele sunt deschise și analizate rând cu rând pentru a extrage informațiile despre universitate, facultate, coduri, programe de studii și disciplinele aferente fiecărui semestru.

Fiecare disciplină extrasă este salvată în baza de date prin intermediul repository-urilor gestionate de Spring. După completarea listei de discipline, planul de învățământ este, de asemenea, salvat, inclusiv toate informațiile colectate. O metodă auxiliară, getValue, interpretează valorile din celulele Excel, gestionând corect tipuri diverse precum valori numerice, text și formule.

Aceste clase sunt flexibile și scalabile, având o structură modulară care permite adăugarea suportului pentru alte programe de studii sau alte formate de fișiere Excel cu modificări minime. Blocurile try-catch sunt utilizate pentru gestionarea erorilor neașteptate, astfel încât procesarea să continue chiar dacă apar valori invalide sau celule goale.

În concluzie, acest set de clase oferă o soluție robustă pentru procesarea automată a fișierelor Excel asociate planurilor de învățământ, cu un grad ridicat de personalizare și extensibilitate pentru diverse programe și formate de studii.

5.4 User

5.5 Templates

Aplicația „Plan Învățământ” este concepută pentru a oferi o soluție completă și eficientă în gestionarea planurilor de învățământ pentru programele de Licență și Master. Cu un design modern, interfață intuitivă și funcționalități bine integrate, aplicația permite utilizatorilor să navigheze rapid și să gestioneze informațiile necesare în mod organizat. Acest sistem integrează tehnologii precum Thymeleaf, Bootstrap și Spring Boot pentru a

asigura o experiență de utilizare plăcută și funcțională, fiind destinat atât administratorilor, cât și profesorilor și studentilor. În cele ce urmează, sunt detaliate principalele componente ale aplicației și modul în care acestea contribuie la administrarea eficientă a disciplinelor și planurilor academice.

5.5.1 Template Index / Login / Sign-up

Pagina principală reprezintă punctul de pornire al aplicației. Aceasta include o bară de navigare în partea superioară, care oferă acces rapid la diferite secțiuni: pagina principală, planuri de licență și master. De asemenea, sunt disponibile două butoane dedicate utilizatorilor autentificați și celor care doresc să își creeze un cont nou. În centrul paginii se regăsește un mesaj de bun venit care explică funcționalitatea aplicației, alături de un buton ce direcționează utilizatorii către secțiunea planurilor de licență. La finalul paginii, un footer afișează informații despre drepturile de autor și include link-uri către termeni și condiții și politica de confidențialitate. Această pagină se remarcă prin designul său atrăgător, realizat cu ajutorul unui fundal gradient și a unor stiluri personalizate care creează un aspect profesional.

Pagina de înregistrare le oferă utilizatorilor posibilitatea de a-și crea un cont nou. Formularul este mai detaliat decât cel de autentificare, incluzând câmpuri pentru nume complet, adresă de email, parolă și confirmarea parolei. Fiecare câmp este însoțit de validare HTML5, cum ar fi verificarea formatului email-ului și lungimea minimă a parolei. De asemenea, există un checkbox prin care utilizatorii trebuie să accepte termenii și condițiile înainte de a putea finaliza procesul de înregistrare. Formularul este poziționat central într-un container stilizat, iar fundalul gradient completează designul modern al paginii. Footer-ul, similar celor din paginile precedente, oferă informații despre drepturile de autor.

Pagina de autentificare permite utilizatorilor existenți să acceseze aplicația prin introducerea adresei de email și a parolei. Formularul este simplu și intuitiv, fiind poziționat central într-un container alb stilizat, care contrastează cu fundalul gradient al paginii. Formularul include opțiunea de a ține utilizatorul conectat, iar butonul de autentificare este bine evidențiat, având un design responsiv. Footer-ul, similar cu cel al paginii principale, afișează drepturile de autor pentru perioada 2024–2025. Pagina integrează validare automată pentru câmpurile de email și parolă, utilizând funcționalitățile HTML5. Atributele de accesibilitate, cum ar fi aria-label, asigură că pagina este ușor de utilizat de către persoanele cu dizabilități.

5.5.2 Template Planuri

Proiectul implementat este un sistem complet pentru gestionarea planurilor de învățământ pentru Licență și Master, oferind funcționalități precum crearea, editarea, afișarea detaliilor și stergerea planurilor. Toate paginile sunt realizate folosind **Thymeleaf** pentru a popula dinamic datele din backend și **Bootstrap** pentru a asigura un design

modern, responsiv și estetic. De asemenea, **Font Awesome** este utilizat pentru integrarea unor iconuri atractive, care îmbunătățesc experiența utilizatorului.

Figura 14 – planInvatamantLicenta-read.html

Fiecare pagină are o structură consistentă, începând cu un **header comun**, prezentat în figura 14. Acesta include link-uri de navigare către secțiunile principale ale aplicației, precum Acasă, Planuri Licență, Planuri Master. De asemenea, include o bară de căutare pentru filtrarea informațiilor (deși aceasta este momentan doar o machetă) și butoane pentru autentificare, oferind opțiuni de login și sign-up.

În paginile de **creare a planurilor**, utilizatorii pot introduce detalii precum universitate, facultate, domeniu fundamental, ramura de știință, coduri specifice și alte informații academice, inclusiv durata studiilor și anul calendaristic. În plus, acestea includ o zonă interactivă pentru încărcarea documentelor, care permite adminului să încarce fișiere fie prin tragere și plasare, fie prin clic. Această funcționalitate este gestionată printr-un script JavaScript care afișează numele și dimensiunile fișierelor selectate.

Paginile de **editare** sunt similare celor de creare, dar cu câmpuri precompletate folosind valorile existente din baza de date. Atributele **Thymeleaf** precum th:value sunt

utilizate pentru a afișa datele curente ale planului, oferind utilizatorilor posibilitatea de a le modifica. După finalizarea modificărilor, utilizatorii pot salva planul actualizat sau anula modificările și reveni la lista planurilor.

Pentru afișarea detaliilor unui plan, este utilizat un tabel organizat clar, care prezintă toate informațiile importante, cum ar fi universitatea, facultatea, domeniul de licență, programul de studii master, forma de învățământ, durata studiilor, domeniul fundamental, codurile asociate și altele. Acțiunile asociate acestui plan, precum editarea, ștergerea sau afișarea disciplinelor, sunt disponibile prin intermediul unor butoane plasate sub tabel.

Lista planurilor (figura 14) este prezentată sub formă de carduri organizate într-un mod responsiv. Fiecare card include detalii precum numele programului de studii și durata acestuia, iar utilizatorii au la dispoziție opțiuni pentru afișarea detaliilor, editare, ștergere sau vizualizarea disciplinelor asociate. Cardurile sunt stilizate astfel încât să aibă dimensiuni uniforme, indiferent de conținut.

Fiecare pagină include un **footer simplu**, care afișează informații despre drepturile de autor și un link pentru a reveni în partea de sus a paginii.

Acest sistem este bine structurat și oferă o experiență ușor de utilizat. Funcționalitățile sunt complete, iar designul modern asigură o interfață clară și prietenoasă. Integrarea cu Thymeleaf permite manipularea dinamică a datelor prin intermediul backend-ului (Spring Boot), iar utilizarea Bootstrap garantează un aspect vizual atractiv și responsiv. Proiectul poate fi extins cu ușurință, adăugând funcționalități suplimentare sau îmbunătățind cele existente.

5.5.3 Template Discipline

Aplicația „Plan Învățământ” oferă un sistem robust pentru gestionarea disciplinelor, adaptat atât programului de licență, cât și celui de master. Aceasta include funcționalități pentru adăugarea, listarea și editarea disciplinelor, fiecare fiind adaptată specificului programului. Diferențele dintre disciplinele de licență și cele de master sunt minime, dar semnificative. Disciplinele de licență sunt clasificate în categorii precum complementare, în domeniu, fundamentale sau de specialitate, iar cele de master includ categorii avansate, cum ar fi discipline de aprofundare, de cunoaștere avansată, de sinteză sau complementare. În ceea ce privește orele alocate, disciplinele de master acordă mai multă importanță proiectelor detaliate și activităților de cercetare.

Adăugarea unei discipline presupune completarea unui formular detaliat care solicită informații precum numele disciplinei, codul, numărul de credite transferabile, forma de evaluare și orele alocate diferitelor activități, cum ar fi cursurile, seminariile, laboratoarele sau proiectele. De asemenea, utilizatorul trebuie să specifice timpul necesar pentru pregătirea individuală și activitățile parțial asistate. Un alt element important este selectarea categoriei formative corespunzătoare programului de studii (licență sau master), precum și

atribuire disciplinei unui semestru specific. Formularul este proiectat să asigure validarea datelor, utilizând Bootstrap și Thymeleaf, astfel încât utilizatorul să nu poată trimite date incomplete sau incorecte. După completarea formularului, utilizatorul are opțiunea de a salva disciplina sau de a renunța la proces.

Listarea disciplinelor este realizată prin intermediul unui tabel bine structurat, care afișează toate informațiile relevante despre discipline. Coloanele tabelului includ attribute precum numele disciplinei, codul, numărul de credite, forma de evaluare, orele asociate diverselor activități (curs, seminar, laborator, proiect), categoria formativă și semestrul. Fiecare rând din tabel are, de asemenea, butoane care permit editarea sau ștergerea disciplinei respective. În cazul în care nu există discipline în baza de date, pagina afișează un mesaj clar care informează utilizatorul despre această situație. Pe lângă tabel, pagina oferă un buton pentru adăugarea unei discipline noi și include navigație facilă către alte secțiuni ale aplicației, precum și o bară de căutare.

Figura 15 – disciplinaZi-update.html

Editarea unei discipline (figura 15) presupune modificarea informațiilor deja existente. Utilizatorul este redirecționat către un formular similar celui pentru adăugare, dar care este precompletat cu informațiile disciplinei selectate. Toate câmpurile sunt editabile,

iar utilizatorul poate actualiza atribute precum numele disciplinei, codul, creditele, forma de evaluare, orele alocate și categoria formativă. După efectuarea modificărilor, utilizatorul poate alege să salveze schimbările, caz în care datele sunt actualizate în baza de date, sau poate renunța, fiind redirecționat înapoi la lista disciplinelor fără ca modificările să fie salvate.

Toate paginile aplicației sunt construite cu un design responsiv, asigurat de Bootstrap, pentru a oferi o experiență optimă pe orice dispozitiv. Thymeleaf este integrat pentru a permite generarea dinamică a paginilor și gestionarea eficientă a datelor. Validarea datelor este asigurată prin combinația dintre Bootstrap și Thymeleaf, prevenind erorile și completările incorecte. Navigația este clară și intuitivă, fiecare pagină având link-uri către alte secțiuni, o bară de căutare și butoane pentru autentificare.

Sistemul oferă astfel o soluție completă pentru gestionarea disciplinelor academice, fiind ușor de utilizat, flexibil și eficient. Acoperă toate etapele necesare pentru administrarea acestora, de la adăugare până la afișare și editare.

5.5.4 Sumar

Aplicația „Plan Învățământ” reprezintă un instrument robust și versatil pentru gestionarea planurilor academice, fiind proiectată să răspundă cerințelor moderne ale instituțiilor de învățământ superior. Prin integrarea tehnologiilor avansate și a unui design responsiv, sistemul oferă o interfață ușor de utilizat, adaptată nevoilor diferitelor categorii de utilizatori. De la adăugarea și editarea disciplinelor, până la afișarea detaliilor acestora, fiecare funcționalitate este optimizată pentru a oferi o experiență eficientă și fără erori. Proiectul demonstrează o abordare inovatoare în digitalizarea proceselor academice și are un potențial semnificativ de extindere și adaptare la noi cerințe.

6 TESTARE ȘI UTILIZARE

7 CONCLUZII

8 BIBLIOGRAFIE

- [1] *** <https://web.archive.org/web/20050420081440/http://java.sun.com/features/1998/05/birthday.html> [accesat Decembrie, 2024]
- [2] *** Cursuri_PJ_1-7.pdf [Curs Java dr. ing. Raul Robu 2022-2023]
- [3] *** <https://computerhistory.org/profile/james-gosling/> [accesat Decembrie, 2024]
- [4] *** <https://www.geeksforgeeks.org/the-complete-history-of-java-programming-language/> [accesat Decembrie, 2024]
- [5] *** https://www.tutorialspoint.com/java/java_features.htm [accesat Decembrie, 2024]
- [6] *** <https://www.javatpoint.com/java-oops-concepts> [accesat Decembrie, 2024]
- [7] *** https://www.tutorialspoint.com/java/java_oops_concepts.htm [accesat Decembrie, 2024]
- [8] *** <https://docs.oracle.com/en/java/index.html> [accesat Decembrie, 2024]
- [9] *** <https://www.oracle.com/java/technologies/javase-documentation.html> [accesat Decembrie, 2024]
- [10] *** <https://docs.oracle.com/javame/8.3/index.html> [accesat Decembrie, 2024]
- [11] *** <https://docs.oracle.com/javaee/7/index.html> [accesat Decembrie, 2024]
- [12] *** <https://www.javatpoint.com/jpa-tutorial> [accesat Decembrie, 2024]
- [13] *** <https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html> [accesat Decembrie, 2024]
- [14] *** <https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html> [accesat Decembrie, 2024]
- [15] *** <https://docs.oracle.com/en/java/javase/23/security/java-security-overview1.html#GUID-2EF91196-D468-4D0F-8FDC-DA2BEA165D10> [accesat Decembrie, 2024]
- [16] *** <https://www.geeksforgeeks.org/introduction-to-spring-boot/> [accesat Decembrie, 2024]
- [17] *** <https://www.baeldung.com/properties-with-spring> [accesat Decembrie, 2024]
- [18] *** <https://www.geeksforgeeks.org/difference-between-spring-and-spring-boot/> [accesat Decembrie, 2024]
- [19] *** <https://poi.apache.org/changes.html> [accesat Decembrie, 2024]
- [20] *** <https://github.com/apache/poi> [accesat Decembrie, 2024]
- [21] *** <https://poi.apache.org/> [accesat Decembrie, 2024]
- [22] *** <https://dev.mysql.com/doc/refman/8.0/en/history.html> [accesat Decembrie, 2024]
- [23] *** <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html> [accesat Decembrie, 2024]
- [24] *** <https://www.simplilearn.com/tutorials/sql-tutorial/difference-between-sql-and-mysql> [accesat Decembrie, 2024]
- [25] *** <https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html#what-is-thymeleaf> [accesat Decembrie, 2024]
- [26] *** <https://www.websoptimization.com/blog/thymeleaf-vs-angular/> [accesat Decembrie, 2024]
- [27] *** <https://getbootstrap.com/docs/5.3/about/overview/> [accesat Decembrie, 2024]

- [28] *** <https://blog.getbootstrap.com/2021/05/05/bootstrap-5/> [accesat Decembrie, 2024]
- [29] *** <https://getbootstrap.com/docs/5.3/getting-started/introduction/> [accesat Decembrie, 2024]

Mirza Daniel-Ionut

RAPORT PRIVIND ORIGINALITATEA

4
%

INDICE DE
SIMILITUDINE

4
%

SURSE DE PE INTERNET

1
%

PUBLICAȚII

1
%

LUCRĂRILE
STUDENTILOR

SURSE PRINCIPALE

- | | | |
|----------|---|-------------------|
| 1 | dspace.upt.ro
Sursă de pe Internet | 3
% |
| 2 | Submitted to Technical University of Cluj-Napoca
Lucrarea studentului | 1
% |
| 3 | www.websiterating.com
Sursă de pe Internet | <1
% |
| 4 | Babeș-Bolyai University
Publicație | <1
% |
| 5 | www.chingiexpert.ro
Sursă de pe Internet | <1
% |
| 6 | Submitted to Stefan cel Mare University of Suceava
Lucrarea studentului | <1
% |

Excludeți citările

Dezactivat

Excludeți bibliografia

Activat

Excludeți similitudinile < 25 words