**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](#_Toc186935494)

[2 TEHNOLOGII WEB 6](#_Toc186935495)

[2.1 LIMBAJUL DE PROGRAMARE JAVA 6](#_Toc186935496)

[2.1.1 Istoria Java 6](#_Toc186935497)

[2.1.2 Caracteristici esențiale 7](#_Toc186935498)

[2.1.3 Programarea orientata pe obiecte 7](#_Toc186935499)

[2.1.4 Biblioteci și ediții 9](#_Toc186935500)

[2.1.5 Java Persistence API 10](#_Toc186935501)

[2.1.6 Java Database Connectivity 11](#_Toc186935502)

[2.1.7 Securitate 12](#_Toc186935503)

[2.1.8 Exemplu de program 12](#_Toc186935504)

[2.1.9 Concluzie Java 13](#_Toc186935505)

[2.2 BIBLIOTECA SPRING BOOT 14](#_Toc186935506)

[2.2.1 Istorie Spring Boot 14](#_Toc186935507)

[2.2.2 Caracterisitici 15](#_Toc186935508)

[2.2.3 Spring vs Spring Boot 15](#_Toc186935509)

[2.2.4 Avantajele Spring Boot 16](#_Toc186935510)

[2.2.5 Program Spring Boot 16](#_Toc186935511)

[2.2.6 Concluzii Spring Boot 17](#_Toc186935512)

[2.3 BIBLIOTECA APACHE POI 17](#_Toc186935513)

[2.3.1 Istoria POI 18](#_Toc186935514)

[2.3.2 Caracteristicile Apache POI 18](#_Toc186935515)

[2.3.3 Avantaje Apache POI 19](#_Toc186935516)

[2.3.4 Exemplu de program 20](#_Toc186935517)

[2.3.5 Concluzie Apache POI 21](#_Toc186935518)

[2.4 SISTEM DE GESTIONARE DE DATE MYSQL 21](#_Toc186935519)

[2.4.1 Istoria MySQL 22](#_Toc186935520)

[2.4.2 Caracteristici MySQL 22](#_Toc186935521)

[2.4.3 MySQL vs SQL 24](#_Toc186935522)

[2.4.4 Avantajele MySQL 25](#_Toc186935523)

[2.4.5 Concluzie 25](#_Toc186935524)

[2.5 BIBLIOTECA THYMELEAF 25](#_Toc186935525)

[2.5.1 Istorie Thymeleaf 26](#_Toc186935526)

[2.5.2 Caracteristici Thymeleaf 26](#_Toc186935527)

[2.5.3 Thymeleaf vs Angular 27](#_Toc186935528)

[2.5.4 Avantaje Thymeleaf 28](#_Toc186935529)

[2.5.5 Concluzie 28](#_Toc186935530)

[2.6 BOOTSTRAP 5 29](#_Toc186935531)

[2.6.1 Istoria Bootstrap 5 29](#_Toc186935532)

[2.6.2 Caracteristici Bootstrap 5 29](#_Toc186935533)

[2.6.3 Avantajele Bootstrap 5 30](#_Toc186935534)

[2.6.4 Concluzie 31](#_Toc186935535)

[3 CERINȚA ȘI SPECIFICAȚIILE PROIECTULUI 31](#_Toc186935536)

[3.1 CERINȚE FUNCȚIONALE 31](#_Toc186935537)

[3.2 CERINȚE NEFUNCȚIONALE 32](#_Toc186935538)

[3.3 CERINȚE TEHNICE 32](#_Toc186935539)

[3.4 FUNCȚIONALITATEA PROIECTULUI 33](#_Toc186935540)

[3.5 ANALIZA DE RISC 33](#_Toc186935541)

[4 PROIECTARE 34](#_Toc186935542)

[4.1 SCHEMA BLOC 34](#_Toc186935543)

[4.2 DIAGRAMA DE CLASE, SECVENTA SI STARE 36](#_Toc186935544)

[4.3 SCHEMA BAZEI DE DATE 37](#_Toc186935545)

[5 IMPLEMENTAREA 39](#_Toc186935546)

[5.1 DESCRIERE CLASE 39](#_Toc186935547)

[5.1.1 CLASA PlanInvatamant 39](#_Toc186935548)

[5.1.2 CLASA PlanInvatamantController 40](#_Toc186935549)

[5.1.3 CLASA PlanInvatamantRepository 42](#_Toc186935550)

[5.1.4 CLASA Disciplina 43](#_Toc186935551)

[5.1.5 CLASA DisciplinaZi si DisciplinaId 44](#_Toc186935552)

[6 Bibliografie 45](#_Toc186935553)

# INTRODUCERE

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

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

Proiectul include trei module principale:

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

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

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

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

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

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

În capitolul 5

În capitolul 6

În capitolul 7

# TEHNOLOGII WEB

## 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 robusteței, securității și flexibilității sale. 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.

### 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”.

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.

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.

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.

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.

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.

### 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 mentenanța pe termen lung.

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 adopția pe scară largă a limbajului de-a lungul timpului.

### Programarea orientata 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. Acesta este o instanță a unei **clase**, ce poate fi considerată un șablon pentru crearea obiectelor. Clasele definesc atributele (datele) și metodele (funcționalitățile) pe care obiectele le vor avea.

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

**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.

**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 suprascrise în clasele derivate).

În fine, **abstracția** este procesul prin care sunt ascunse detaliile de implementare, expunându-se doar funcționalitățile relevante. Acest lucru este realizat prin utilizarea claselor abstracte 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, permițând adaptarea codului la cerințe noi și simplificând gestionarea comportamentelor complexe.

În fine, OOP contribuie la întreținerea mai ușoară 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 (OOP) 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.

### 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)**.

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.

Java EE (Entreprise 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 persistentei 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.

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.

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.

### 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, permițând dezvoltatorilor să lucreze cu obiecte Java care reprezintă datele stocate în tabelele unei baze de date. În loc să se bazeze pe 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.

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, atâta timp cât se utilizează o implementare compatibilă, cum ar fi Hibernate sau EclipseLink.

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.

### 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, comenzile 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.

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, atâta timp cât există un driver corespunzător.

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.

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.

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 abstracție 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.

### 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.

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 malițioase 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.

### Exemplu de program

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 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!

### 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, conceptele 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 domeniu – enterprise, IoT sau aplicații web – Java rămâne o soluție de încredere pentru dezvoltatori.

## 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.

### 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 configurării automate și ușurinței de utilizare, eliminând multe dintre complexitățile tradiționale ale configurării aplicațiilor Spring.

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 configurări 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, permițând 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.

### Caracterisitici

Spring Boot se remarcă printr-o serie de caracteristici care îl fac ideal pentru dezvoltarea rapidă și eficientă a aplicațiilor Java. În primul rând, configurarea 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 configurării unui server extern.

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 configurare inițială.

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.

### 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 variate ale dezvoltării software, cum ar fi inversiunea controlului (IoC), programarea orientată pe aspecte (AOP), securitatea, accesul la baze de date și multe altele. Totuși, configurarea 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.

În contrast, Spring Boot este o extensie a Spring Framework creată pentru a simplifica dezvoltarea aplicațiilor. Oferă configurare automată, detectând și configurând setările implicite 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.

Diferențele dintre cele două constau în nivelul de complexitate și în scopurile lor principale. Spring Framework necesită configurare 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 configurare. Alegerea între cele două depinde de tipul și complexitatea proiectului.

### Avantajele Spring Boot

Spring Boot se remarcă prin configurarea automată și simplitatea procesului de dezvoltare, ceea ce îl face ideal pentru proiecte rapide și eficiente. Framework-ul elimină necesitatea configurărilor complexe, detectând automat dependențele și aplicând setări implicite optimizate. Generarea proiectelor este rapidă datorită instrumentului Spring Initializr, care creează un schelet de aplicație configurat pentru nevoile specifice ale dezvoltatorilor.

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.

### 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.

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 2 Program Spring Boot

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.

### Concluzii Spring Boot

Spring Boot se distinge prin capacitatea sa 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.

## 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 versatilă 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 sa de a gestiona mari volume de date.

### 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.

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.

### 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 extracția 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 formatări 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.

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 gestiona 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 extracția 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.

### 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.

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.

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.

### Exemplu de program

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

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 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 dependința corespunzătoare în fișierul de configurare al proiectului, cum ar fi *pom.xml* pentru Maven. Gestionarea excepțiilor este esențială, deoarece operațiile de citire și scriere implică riscul de erori legate de I/O.

### 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.

## 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 performanța, 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 limbaje 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 fundamentul majorității aplicațiilor software și sistemelor informatice. 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 performanța și versatilitatea sa.

### 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.

Un moment definitoriu 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.

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.

### Caracteristici MySQL

MySQL este un sistem de gestionare a bazelor de date relaționale cunoscut pentru performanța, 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 operațiuni 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.

Un alt aspect important al MySQL este modelul său de licențiere duală. Acesta este disponibil gratuit sub licența 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.

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. Acesta 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 operațiuni 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.

### 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.

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.

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 lucrul 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 fundamentul teoretic, MySQL reprezintă aplicația practică a acestuia.

### 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 startup-uri, 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.

### 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.

## BIBLIOTECA THYMELEAF

Thymeleaf este un motor de șabloane 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 șabloane 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 șabloane, Thymeleaf este o alegere excelentă pentru dezvoltarea aplicațiilor web tradiționale și multilingve.

### Istorie Thymeleaf

Thymeleaf este o bibliotecă Java dedicată construirii de aplicații web, utilizată în special pentru generarea de șabloane 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, permițând 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 șabloanele 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 șabloane 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 șabloanelor. 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 șabloane 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.

### 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 șabloane 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.

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).

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

### Thymeleaf vs Angular

Thymeleaf și Angular diferă semnificativ în modul în care procesează și livrează datele către utilizator. Thymeleaf este un motor de șabloane server-side, ceea ce înseamnă că procesarea șabloanelor 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 interfața dinamică direct în browser, folosind JavaScript și HTML.

Din punctul de vedere al paradigmei 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.

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 permițând 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.

În ceea ce privește performanța și scalabilitatea, Thymeleaf poate deveni mai lent pe măsură ce numărul utilizatorilor crește, deoarece procesarea șabloanelor 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.

### Avantaje Thymeleaf

Thymeleaf este un motor de șabloane 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, permițând 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.

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 șabloanele 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 șabloane, ceea ce contribuie la organizarea clară a aplicației și reduce redundanța.

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.

### Concluzie

Thymeleaf se remarcă prin integrarea sa naturală cu Spring Framework, ușurința utilizării și caracteristici care reduc timpul de dezvoltare, cum ar fi șabloanele 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.

## BOOTSTRAP 5

Bootstrap 5 este cea mai recentă versiune a unuia 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.

### 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.

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 aliniează 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.

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

### 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.

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 componentele 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.

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.

### 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, permițând gestionarea mai ușoară a structurilor complexe și asigurând un design responsive.

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 cititoarele 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.

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.

### 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.

# CERINȚA ȘI SPECIFICAȚIILE PROIECTULUI

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

## CERINȚE FUNCȚIONALE

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

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

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

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

## CERINȚE NEFUNCȚIONALE

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

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

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

## CERINȚE TEHNICE

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

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

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

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

## FUNCȚIONALITATEA PROIECTULUI

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

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

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

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

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

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

## ANALIZA DE RISC

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

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

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

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

# PROIECTARE

## SCHEMA BLOC

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

O imagine care conține text, diagramă, cerc, linie

Descriere generată automat

Fig 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.

## DIAGRAMA DE CLASE, SECVENTA SI STARE

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

O imagine care conține diagramă, cerc, linie, schiță

Descriere generată automat

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

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

Clasa ***Student*** descrie studenții și conține atributele `id` (identificator unic) și `nume` (numele studentului). Metoda `vizualizeazaPlanuri()` permite studenților să vizualizeze planurile de învățământ, filtrate după generație, 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 compoziție cu *FisaDisciplina*, ceea ce reflectă faptul că un plan de învățământ conține mai multe fișe de disciplină. De asemenea, *FisaDisciplina* este asociată cu *CadruDidactic*, pentru a indica alocarea unui profesor unei discipline. În plus, clasa *Student* este asociată cu *PlanInvatamant*, permițând accesul studenților la planurile de învățământ.

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

## 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.

O imagine care conține text, diagramă, linie, captură de ecran

Descriere generată automat

Fig 6 Schema bazei de date

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

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

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

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

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

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

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

# IMPLEMENTAREA

## DESCRIERE CLASE

### CLASA PlanInvatamant

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 7 Clasa PlanInvatamant

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

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

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

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

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

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

### CLASA PlanInvatamantController

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

O imagine care conține captură de ecran, text, software, Software multimedia

Descriere generată automat

Fig 8 Clasa PlanInvatamantRepository

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

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

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

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

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

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

### CLASA PlanInvatamantRepository

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

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 9 Clasa PlanInvatamantRepository

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

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

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

### CLASA Disciplina

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 10 Clasa Disciplina

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

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

Constructorul implicit este gol, ceea ce permite JPA să creeze instanțe ale clasei fără a specifica inițial valori pentru atribute. În plus, clasa include metode getter și setter pentru fiecare atribut, asigurând accesul și modificarea acestora în conformitate cu principiile de incapsulare. Metodele getter returnează valorile atributelor, iar metodele setter permit atribuirea unor valori noi.

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

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

### CLASA DisciplinaZi si DisciplinaId

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

O imagine care conține text, captură de ecran, software, Software multimedia

Descriere generată automat

Fig 11 Clasa DisciplinaZi și DisciplinaId

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

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

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

# Bibliografie