

LUCRARE DE LICENȚĂ

**Coordonator:**

**Absolvent:**

Chițu Raluca-Oana

Brașov

Februarie 2025



LUCRARE DE LICENȚĂ

Tehnologii web în realizarea aplicației Complex turistic Luana Nature

**Absolvent:** Chițu Raluca-Oana

**Coordonator:** Conf.Dr. Sângeorzean Livia

Brașov

Februarie 2025

CUPRINS

[Capitolul 1 Introducere 4](#_Toc187423107)

[1.1 Prezentarea lucrării 4](#_Toc187423108)

[1.2 Motivația alegerii temei 4](#_Toc187423109)

[1.3 Aplicații similare 4](#_Toc187423110)

[1.4. Structura lucrării 5](#_Toc187423111)

[Capitolul 2 Dezvoltarea aplicației 6](#_Toc187423112)

[2.1 Medii de dezvoltare și sisteme de gestionare a bazelor de date 6](#_Toc187423113)

[2.1.1 IntelliJ IDEA – liderul IDE pentru Java 6](#_Toc187423114)

[2.1.2 PostgreSQL – cea mai avansată bază de date relațională open-source din lume 7](#_Toc187423115)

[2.2. Dezvoltarea server-side (*Back-End)* 8](#_Toc187423116)

[2.2.1 Spring – Spring Boot 8](#_Toc187423117)

# Capitolul 1 Introducere

## 1.1 Prezentarea lucrării

Cine nu și-ar dori un spațiu în care munca de birou zilnică să fie combinată cu natura? Într-o lume a vitezei, în care alergăm după mici momente de respiro și relaxare, tânjim după ieșirea din lumea betonelor și a poluării care ne înconjoară.

Aplicația Luana Nature este în fapt un liant între complexul turistic cu același nume și cei care doresc o oază de relaxare în care pot gestiona activitățiile zilnice de birou, în liniște. În sine, această aplicație oferă utilizatorilor săi posibilitatea de a vizualiza toate beneficiile oferite de locația precedent menționată, precum și posibilitatea realizării unei rezervări online. Totodată, utilizatorii pot trimite mesaje diverse sau întrebări. De asemenea, acești pot evalua locația în sine prin intermediul opțiunii de review-uri.

O altă facilitate oferită de aplicație este aceea de achiziționare a produselor oferite de magazinul local aflat în complex. Fiecare utilizator își poate vizualiza comanda înainte de trimitere și totodată, poate vedea în timp real prețul aferent produselor.

Toate facilitățile oferite de aplicație (creeare unei rezervări, trimitere de review-uri sau mesaje, comandă de produse) se realizează pe bază de cont.

Aplicația oferă de asemenea acces utilizatorilor cu drept de administrator, aceștia având dreptul de a: vizualiza utilizatorii, review-urile, comenzile, rezervările.

## 1.2 Motivația alegerii temei

La baza alegerii temei pentru creeare aplicației a stat dorința de a pune în valoare un complex turistic care iese din tiparele cunoscute. Mediul online este mediul prin care oamenii au acces în cel mai facil și rapid mod la locuri și zone nou dezvoltate. Practic, prin realizarea acestui web site, se dorește promovarea complexului turistic Luana Nature și prezentarea lui în fața celor interesați de astfel de spații naturale.

## 1.3 Aplicații similare

În lumea online există numeroase site-uri web care promovează complexuri turistice. Există atât platforme de promovare, cât și pagini web specifice unei singure locații. Din prima categorie, poate cea mai cunoscută este Booking.com, iar pe lângă aceasta câteva exemple ar fi: DirectBooking.ro, TuristInfo.ro, TravelMinit.ro, Trivago.ro, Agoda.com. Din cea de-a doua categorie putem exemplifica prin următoarele: Cheile Grădiștei Fundata, Mona Caraiman, Serenity Resort, Europark Fundata.

Toate aceste pagini web, indiferent de categorie, au următoarele elemente cheie:

* Utilizatorii pot vizualiza facilitățiile oferite (cazare, restaurant, activități etc.)
* Există opțiunea de a realiza rezervări online
* Scurtă prezentare a obiectivului turistic
* Date de contact
* Galerie foto și video

## 1.4. Structura lucrării

Primul capitol prezintă, pe scurt, scopul aplicației, motivarea alegerii acestui tip de aplicație, precum și a altor aplicații asemănătoare, existente deja.

Cel de-al doilea capitol face referire la principalele elemente tehnologice utilizate în cadrul aplicației, atât tool-uri, cât și limbajele de programare utilizate.

În cel de-al treilea capitol este presentată specific partea de structură a aplicației, arhitecturam dar și partea de structurare a codului.

Capitolul patru prezintă, în fapt, ghidul aplicației, fiind prezentată partea grafică a acesteia și modul de utilizare.

În capitolul cinci, ultim, se regăsesc concluziile și prezentarea unor funcționalității viitoare.

# Capitolul 2 Tehnologii web utilizate

## 2.1 Medii de dezvoltare și sisteme de gestionare a bazelor de date

### 2.1.1 IntelliJ IDEA – liderul IDE pentru Java

A black square with white letters on it

Description automatically generated

IntelliJ IDEA reprezintă un mediu de dezvoltare integrat (IDE) utilizat pentru crearea de aplicații software. A fost creat de compania JetBrains și este unul dintre cele mai populare IDE-uri, fiind preferat pentru programarea în limbaje precum Java, în primul rând, Kotlin, Groovy, Scala, în al doilea rând.

Principalele caracteristici ale IntelliJ IDEA sunt:

* **Oferă unul dintre cele mai puternice editoare de cod din industrie** – capacitate de înțelegere a codului ceea ce permite detectarea din timp a erorilor, sugerarea opțiunilor de completare a codului, refactorizarea acestuia și detectarea duplicatelor;
* **Capacitate de completarea a codului** - având un model avansat de învățare profundă, permite îmbunătățirea eficenței codării prin sugestii și prezicere a unor linii întregi de cod;
* **Asistent AI** – utilizat pentru scrierea automată a comentariilor, generare de mesaje de confirmare, sugestii de nume etc.;
* **Furnizează suport pentru cele mai populare framework-uri dedicate dezvoltării de aplicații moderne și microservicii** – oferă asistență pentru Spring și SpringBoot, Jakarta EE, JPA, Micronaut, Quarkus, Helidon etc.
* **Oferă instrumente de tip built**-**in și integrări –** debugger (util pentru urmărirea codului pas cu pas), profiler (pentru date privind timpul de execuție și alocare de memorie), suport pentru baze de date, HTTP client (utilizat pentru cerereri de tip HTTP în cadrul dezvoltării serviciilor de tip web);
* **Cloud și distribuție –** Kubernets (sistem open-source pentru implementarea, scalarea și gestionarea aplicațiilor continerizate), Docker (descărcare și construire de imagini, rulare și depanare aplicații), aplicații server (Tomcat, Jboss, Glassfish etc. – pornire și oprire servere locale, conectare la servere la distanță);
* **Dezvoltare web –** suport pentru o gamă largă de tehnologii frontend precum JavaScript, HTML, CSS, React, Node.js etc;
* **Version control -** colaborează cu cele mai cunoscute sisteme de control a versiunilor (GitHub, GitLub, Mercurial etc.) și, totodată, oferă posibilitatea de anulare a modificărilor și recuperarea fișierelor șterse accidental (Local History).

IntelliJ IDEA se comercializează atât în variant gratuită (*Community Edition)* , cât și în varianta plătită (*Ultimate*).

### 2.1.2 PostgreSQL – cea mai avansată bază de date relațională open-source din lume

A blue elephant with white outline

Description automatically generated

PostgreSQL este un sistem puternic de gestionare a bazelor de date care folosește și extinde limbajul SQL. Combină numeroase caracteristici care stochează și extind în siguranță cele mai complicate volume de date. Orginile sale datează încă din 1986 fiind dezvoltat de Universitatea California din Berkeley, Statele Unite ale Americii.

PostgreSQL rulează pe toate sistemele de operare majore, iar reputația sa se datorează arhitecturii sale, fiabilității, integrității datelor, extensibilității și setului robust de caracteristici. De asemenea, este dedicat comunității open-source, oferind constant soluții performante și inovatoare.

Totodată, este un sistem care încearcă să se conformeze standarului SQL și de aceea, începând cu versiunea 16 lansată în septembrie 2023, acesta este conform cu 170 din cele 177 caracteristici obilgatorii pentru conformitatea SQL:2023 Core.

Din punct de vedere al caracteristiciilor oferite, PostgreSQL oferă:

* **Tipuri de date -** primitive (integer, numeric, String, Boolean), structurate (dată/oră, interval/multi-interval), de tip document (JSON, XML), geometrice (punct, linie, cerc, poligon);
* **Integritate date -** chei primare, chei străine, constrângeri de excludere, evitare date nule, date unice, blocaje
* **Concurență și performanță –** userii pot lucra în mod simultan, fără alterarea datelor și cu performanță ridicată;
* **Fiabilitate și recuperare în caz de dezastru**
* **Securitate -**  inclusiv autentificare de tip multi-factor cu certificat;
* **Extensibilitate** – funcții stocate și proceduri, limbaje procedurale (PL/pgSQL, Phyton, Perl)
* **Internaționalizare** – suport pentru seturi internaționale de caractere, corelații care nu țin cont de majuscule, căutarea textului integral.

PostgreSQL are multe caracteristici dedicate să ajute dezvoltatorii să creeze aplicații, administratorii să protejeze integritatea datelor, indiferent de dimensiunea setului de date.

Pe lângă faptul că este un sistem gratuit, acesta este și foarte extensibil. Spre exemplu, se pot crea tipuri de date și funcții personalizate. De asemenea, oferă posibilitatea scrierii de cod în diferite limbaje de programare, fără să fie nevoie de reconstrucția bazei de date.

## 2.2. Dezvoltarea server-side (*Back-End)*

### 2.2.1 Spring – Spring Boot

#### 2.2.1.1 Introducere



Spring este cel mai popular framework folosit pentru a dezvolta aplicații pe baza limbajului Java. Este puternic și extrem de flexibil, fiind axat pe aplicații de tip web.

Unul din principalele avantaje ale Sprin este că se ocupă de majoritatea părților de nivel scăzut ale construirii unei aplicații, permițând focusul pe partea de bussines logic.

Spring este cunoscut pentru capacitățiile sale de Inversion of Control (IoC) și Dependency Injection (DI) care simplifică crearea de aplicații modulare și testabile. Caracteristicile cheie includ Spring MVC pentru dezvoltarea web, Spring Boot pentru configurarea rapidă a aplicațiilor și Spring Security pentru autentificare și autorizare robuste. Cu un ecosistem bogat care acoperă Spring Data pentru interacțiunile cu bazele de date și Spring Cloud pentru crearea de microservicii, Spring susține soluții de întreprindere scalabile și rezistente, făcându-l un cadru esențial pentru dezvoltatorii de toate nivelurile de experiență.

Din punc t de vedere al elementelor care fac Spring să strălucească, acestea ar fi:

* **Utilitate** – oferă multiple opțiuni de configurare și astfel este ușor de folosit de către dezvoltatori;
* **Modularitate** – există opțiunea de a utiliza întreg framework-ul Spring sau doar modulele necesare (Spring Security, Spring Web, Spring Data, Spring Boot, Spring Cloud etc.);
* **Aplicații web securizate** – oferă setări de securitate personalizate pentru autentificare de bază și previne vulnerabilitățiile;
* **Suportă modelul MVC (Model View Controller)** – acest model este folosit pentru aplicații web și permite separarea diferitelor aspecte ale acestora (interfață utilizator, data acces layer, business layer);
* **Comunicare ușoară cu baza de date**;
* **Permite integrarea cu alte framework-uri** – de exemplu Hibernate, operațiuniile de tip CRUD (Create, Read, Update, Delete) fiind mai ușor de implementat;
* **Inversion of control (IoC)** – permite transferul controlului asupra obiectelor către un framework;
* **Dependency injection** **(DI)** – modulele unui proiect pot fi utilizate în mod eficient în alt proiect (de exemplu, paginiile de logare și autentificare). Ideea de DI este aceea de a conecta obiectele între ele, astfel încât un obiect primește dependețele necesare de la o sursă externă în loc să le creeze el însuși;
* **Testare ușoară a unei aplicații**.



Spring Boot reprezină o extindere a framework-ului Spring și a fost dezvoltat pentru a simplifica procesul de configurare și de a permite crearea mai rapidă a aplicațiilor. În Spring Boot totul este auto-configurat.

Principalele caracteristici ale Spring Boot sunt:

* **Evitarea configurării grele prezentă în Spring, prin fișiere de tip XML:**
* **Incorporarea server-ului Apache Tomcat și altele asemenea (Jetty, Undertow)** – acesta reprezintă un server web care deservește aplicații de Java de tip web. Practic, este un program care ajută aplicațiile web Java să funcționeze și să fie accesibile utilizatorilor prin intermediul unui browser;
* **Util în arhitectura bazată pe microservicii** – fiecare serviciu care rulează într-o aplicație are propriul său proces, astfel serviciile fiind independente între ele. De exemplu, într-o aplicație de tip magazin online, microserviciile pot fi: serviciul de comandă, serviciul de gestionare useri, serviciul de vizualizare produse etc.;
* **Crește productivitatea și reduce timpul de dezvoltare a aplicațiilor;**
* **Integrare ușoară a altor module Spring –** precum Spring JDBC, Spring Security, Spring Web
* **Facilitează conectarea la baze de date -**  prin utilizarea unor simple adnotări (@Entity, @Table, @ Column), Spring Boot permite accesul și chiar crearea de tabele în tool-uri precum PostgreSQL, Oracle, MySQL etc.;
* **Permite comasarea adnotărilor -** de exemplu, @SpringBootApplication integrează adnotările din Spring: @Configuration, @ComponentScan, @EnableAutoConfiguration;
* **Oferă posibilitatea creării de endpoint-uri de tip REST (Representational State Transfer) –** practic construirea unei aplicații sau serviciu de tip web, prin care se creează comunicarea între un client și un server.

Spring Boot a fost creat în 2013 și lansat pentru prima dată în aprilie 2014. Versiunile sale pot fi vizualizate la adresa:

<https://mvnrepository.com/artifact/org.springframework.boot/spring-boot>

#### 2.2.1.2 Arhitectură, elemente specifice și funcționare

1. **Arhitectura Spring Boot**

Are la bază patru nivele principale și anume:

* **Presentation layer**: reprezeintă partea de vizualizare sau, mai simplu spus, partea pe care o vede utilizatorul (*frontend*);
* **Data access layer**: reprezintă nivelul unde au loc operațiunile legate de baza de date – CRUD (*Create, Read, Update, Delete*)
* **Service Layer:** conține clasele de servicii și folosește serviciile furnizate de data access layer;
* **Integration layer:** se referă la diferite tipuri de servicii web, practic orice serviciu disponibil online și utilizează sistemul de mesagerie XML);

Pe lângă acestea, mai există clasele de utilitate, cele de validare și cele de vizualizare.

A diagram of a spring boot flow

Description automatically generated

Arhitectura Spring Boot

1. **Terminologie Spring Boot**

|  |  |
| --- | --- |
| TERMEN | DESCRIERERE |
| Actuator | Modul ce oferă funcționalități pregătite pentru producție, cum ar fi verificarea stării (health checks), metrici și puncte de monitorizare. |
| Bean | O componentă gestionată de containerul Spring, creată de obicei cu ajutorul anotărilor @Bean, @Component sau @Service. |
| Banner | Textul ASCII sau graficul afișat la pornirea aplicației, care poate fi configurat prin banner.txt. |
| Clasa SpringApplication | |  | | --- | |  |  |  | | --- | | Punctul de intrare pentru lansarea unei aplicații Spring Boot, folosită de obicei în metoda main. | |
| CommandLineRunner | O interfață utilizată pentru a rula cod specific după pornirea aplicației Spring Boot, implementată de obicei în bean-uri personalizate. |
| Controller | O clasă notată cu @Controller sau @RestController care gestionează cererile HTTP și răspunsurile. |
| DevTools | Un modul Spring Boot pentru productivitatea dezvoltatorilor, care permite reporniri automate și reîncărcări live în timpul dezvoltării. |
| Embedded Server | |  | | --- | |  |  |  | | --- | | Un server web integrat (cum ar fi Tomcat, Jetty sau Undertow) care elimină necesitatea unui server extern pentru a rula aplicația. | |
| Endpoint | |  | | --- | |  |  |  | | --- | | O cale URL (de obicei expusă de un controller) sau o funcționalitate a Actuator care oferă funcții specifice (ex. /api/v1/resource, /actuator/info). | |
| Entity | O clasă notată cu @Entity ce reprezintă un tabel din baza de date în JPA. |
| Fișier de proprietăți/YAML | Fișiere de configurare (application.properties sau application.yml) pentru setările aplicației și configurații specifice mediului. |
| Health Indicators | Funcționalități ale Spring Boot Actuator care raportează starea aplicației (ex. conectivitate la baza de date, spațiu pe disc). |
| Injecție de Dependențe (DI) | Procesul prin care Spring Boot furnizează automat bean-uri sau servicii către alte componente (prin @Autowired sau injecție prin constructor). |
| Logging | Gestionat prin SLF4J și Logback în mod implicit, configurabil prin application.properties. |
| Profile | Un mecanism pentru definirea configurațiilor specifice mediului, activate prin spring.profiles.active (ex. dev, test, prod). |
| Repository | O interfață notată cu @Repository, folosită pentru operații CRUD și interacțiuni cu baza de date, de obicei extinsă din JpaRepository. |
| Server încorporat | Un server web integrat (cum ar fi Tomcat, Jetty sau Undertow) care elimină necesitatea unui server extern pentru a rula aplicația. |
| Spring Boot Auto-Configuration | Configurare automată a bean-urilor Spring pe baza dependențelor și proprietăților existente în classpath. |
| Spring Boot CLI | Un instrument de linie de comandă pentru rularea aplicațiilor Spring Boot și scripturilor scrise în Groovy. |
| Spring Boot Initializr | Un instrument online pentru generarea de proiecte Spring Boot cu dependențe preconfigurate (disponibil la [start.spring.io](https://start.spring.io)). |
| Spring Boot Starter | Dependențe preconfigurate care simplifică gestionarea dependențelor (ex. spring-boot-starter-web, spring-boot-starter-data-jpa). |
| Spring Cloud | O suită de instrumente pentru construirea microserviciilor, cu funcții precum descoperirea serviciilor, configurarea și echilibrarea încărcării. |
| Spring Data JPA | Modul ce simplifică accesul și gestionarea bazelor de date prin intermediul repository-urilor JPA cu suport încorporat pentru operații CRUD. |
| Spring MVC | Un framework pentru construirea serviciilor RESTful sau a aplicațiilor web, utilizând anotări precum @Controller și @RestController. |
| Spring Security | Un modul Spring pentru securizarea aplicațiilor, oferind autentificare, autorizare și protecție CSRF. |
| Thymeleaf | Un motor de șabloane pe partea de server, utilizat frecvent în Spring Boot pentru a genera conținut HTML dinamic. |

**Tabel 1 – Terminologia specifică Spring Boot**

1. **Adnotări specifice Spring Boot**

O adnotare (în engleză: annotation) este un construct specific limbajului Java (și altor limbaje derivate), folosit pentru a atașa metadate (informații suplimentare) la codul sursă, cum ar fi clase, metode, câmpuri, sau parametri. Adnotările nu afectează direct logica aplicației, dar pot influența comportamentul programului prin intermediul procesării la rulare (runtime) sau la compilare.

|  |  |
| --- | --- |
| ADNOTARE | DESCRIERE |
| @Autowired | Permite injecția automată a unei dependențe gestionate de Spring. |
| @Bean | Definește un bean care va fi gestionat de Spring. |
| @Column | Specifică detalii despre o coloană din tabelul asociat cu o entitate JPA. |
| @Component | |  | | --- | |  |  |  | | --- | | Marchează o clasă ca fiind un bean gestionat de containerul Spring (componentă generică). | |
| @ComponentScan | Specifică pachetele care trebuie scanate pentru componente și bean-uri. |
| @Configuration | Marchează o clasă ca sursă de definiții pentru bean-uri Spring. |
| @DeleteMapping | Maparea unei metode la o cerere HTTP de tip DELETE. |
| @Entity | Definește o clasă ca fiind o entitate JPA, care va fi mapată la un tabel din baza de date. |
| @EnableAutoConfiguration | Activează mecanismul de configurare automată din Spring Boot. |
| @GeneratedValue | |  | | --- | |  |  |  | | --- | | Indică modul în care valoarea cheii primare este generată (ex. automat, secvențial etc.). | |
| @GetMapping | Maparea unei metode la o cerere HTTP de tip GET. |
| @Id | Marchează câmpul ca fiind cheia primară a unei entități JPA. |
| @PathVariable | Extrage o variabilă de cale din URL (ex. /api/{id}). |
| @PostMapping | Maparea unei metode la o cerere HTTP de tip POST. |
| @Primary | |  | | --- | |  |  |  | | --- | | Indică bean-ul implicit care să fie utilizat în cazul în care există mai multe potriviri. | |
| @Profile | |  | | --- | |  |  |  | | --- | | Specifică mediul în care un bean sau o configurație va fi activă (ex. dev, prod). | |
| @PutMapping | Maparea unei metode la o cerere HTTP de tip PUT. |
| @Qualifier | Specifică ce bean să fie injectat atunci când există mai multe candidaturi posibile. |
| @Repository | Marchează o clasă ca fiind responsabilă pentru interacțiunea cu baza de date. |
| @RequestBody | Convertește corpul unei cereri HTTP într-un obiect Java. |
| @RestController | Combina @Controller și @ResponseBody, returnând date JSON sau XML pentru cererile HTTP. |
| @RestTemplate | Creează un obiect de tip RestTemplate pentru a face apeluri HTTP către alte servicii web. |
| @Service | O specializare a @Component, utilizată pentru a marca clasele care conțin logica de business. |
| @SpringBootApplication | Indică punctul de intrare principal al unei aplicații Spring Boot. Combina @Configuration, @EnableAutoConfiguration și @ComponentScan. |
| @Table | Specifică numele tabelului asociat cu o entitate JPA. |
| @Transactional | Marchează o metodă sau o clasă pentru a executa operații într-o tranzacție. |
| @Value | Injecția unei valori din proprietăți externe (application.properties sau application.yml). |

**Tabel 2 – Adnotări specifice Spring Boot**

1. **Crearea unui proiect**

Initițializarea unui proiect care utilizează Spring Boot se poate face prin ceea ce se numește **spring initializr** (<https://start.spring.io/> ).

Metoda reprezintă un mod de a seta foarte rapid un proiect nou, fără a fi nevoie de configurare manuală.

A screenshot of a computer

Description automatically generated

Asftel, spring initializr este divizat în două parți. Partea din stânga se referă la setările privind tipul de proiect, limbajul de programare, versiunea de Spring Boot și datele de tip metadata:

* **Gradle – Groovy, Gradle – Kotlin, Maven** - sunt instrumente de automatizare a construirii proiectelor în ecosistemul Java. Ele ajută la gestionarea dependințelor, compilarea codului, rularea testelor și construirea aplicațiilor;
* **3.x.x / 3.x.x SNAPSHOT –** reprezintă versiunile cele mai recente ale Spring Boot, varianta snapshot fiind o versiune în dezvoltare;
* **Group** - este un identificator unic pentru grupul sau organizația care deține proiectul;
* **Artifact** - identificator unic pentru proiectul sau modulul efectiv;
* **Name –** denumirea proiectului;
* **Description** – scurtă descriere a proiectului ;
* **Package name** – pachetul în care sunt incluse clasele Java;
* **Packaging** - pecifică tipul de fișier care va fi creat în urma procesului de construcție. Indică ce tip de artefact va fi produs (JAR, WAR, EAR etc.) ;
* **Java** – versiunea de Java ce urmează a fi folosită.

Partea din dreapta oferă posibilitatea de a adăugare automată a dependințelor care se doresc a fi utilizate în proiect, cum ar fi: Spring Web, Thymeleaf, Spring Security, Spring Data Jpa etc.

A screenshot of a computer

Description automatically generated

În continuare, prin selectarea butonului A white rectangular sign with black text

Description automatically generated se poate vedea o pre-vizualizare a proiectului:

A screenshot of a computer

Description automatically generated

Butonul A black and white rectangle with black text

Description automatically generated este utilizat pentru generarea proiectului sub formă de arhiva .zip, acesta fiind ulterior deschisa în IntelliJ IDEA.

### 2.2.2 PostgreSQL



PostgreSQL reprezintă un sistem de baze de date open-source care acceptă interogări de tipul celor relaționale (SQL) și celor non-relaționale (JSON). Acest sistem este oferit în mod gratuit. De asemenea, este folosit în cazul construirii de site-uri web dinamice și de aplicații web. Acesta poate fi folosit împreună cu cele mai cunoscute limbaje de programare: Java, Python, C/C++, Node.js, Go, Ruby, Perl.

O listă cu funcționalitățile care se pot regăsi în PostgreSQL este următoarea:

* Tipuri de date
  + Primitive: Integer, Numeric, String, Boolean
  + Structurate: Data/ora, Array, Interval / Multi interval, UUID
  + Documente: JSON/JSONB, XML, Cheie-valoare (Hstore)
  + Geometrie: Punct, Linie, Cerc, Poligon
  + Personalizări: Compus, Tipuri personalizate
* Integritate date
  + UNIC, NU NUL
  + Chei primare
  + Chei străine
  + Constrângeri de excludere
  + Blocaje explicite, blocaje consultative
* Concurență, performanță
  + Indexare: B-tree, Multicolumn, Expresii, Parțial
  + Indexare avansată: GiST, SP-Gist, KNN Gist, GIN, BRIN, indexuri de acoperire, filtre Bloom
  + Planificator / optimizator sofisticat de interogări, scanări numai cu index, statistici pe mai multe coloane
  + Tranzacții, tranzacții imbricate (prin puncte de salvare)
  + Controlul simultaneității în mai multe versiuni (MVCC)
  + Paralelizarea interogărilor de citire și construirea de indexuri B-tree
  + Partiționarea tabelelor
  + Toate nivelurile de izolare a tranzacțiilor definite în standardul SQL, inclusiv Serializable
  + Compilarea just-in-time (JIT) a expresiilor
* Fiabilitate, recuperare în caz de dezastru
  + Înregistrare de tip Write-ahead Logging (WAL)
  + Replicare: Asincronă, sincronă, logică
  + PITR (Point-in-time-recovery), standby activ
  + Spații de tabele
* Securitate
  + Autentificare: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, certificat și altele
  + Sistem robust de control al accesului
  + Securitate la nivel de coloană și rând
  + Autentificare multi-factor cu certificate și o metodă suplimentară
* Extensibilitate
  + Funcții și proceduri stocate
  + Limbaje procedurale: PL/pgSQL, Perl, Python și Tcl. Există și alte limbaje disponibile prin extensii, de exemplu Java, JavaScript (V8), R, Lua și Rust
  + Constructori SQL/JSON, funcții de interogare, expresii de cale și JSON\_TABLE
  + Învelitori de date străine: conectați-vă la alte baze de date sau fluxuri cu o interfață SQL standard
  + Interfață de stocare personalizabilă pentru tabele
  + Multe extensii care oferă funcționalități suplimentare, inclusiv PostGIS
* Internaționalizare, căutare de text
  + Suport pentru seturi internaționale de caractere, de exemplu prin colajele ICU
  + Colerații care nu țin cont de majuscule și accente
  + Căutarea textului integral

## 2.3. Dezvoltarea client-side (*Front-End)*

### 2.3.2 HTML – Hypertext Markup Language

HTML este un acronim pentru HyperText Markup Language (Limbaj de Marcare Hypertext). Este limbajul standard folosit pentru a crea și structura conținutul pe web. HTML îi indică browserului web cum să afișeze text, linkuri, imagini și alte forme de conținut multimedia pe o pagină web. Acesta stabilește structura de bază a unui site web, iar limbaje precum CSS și JavaScript adaugă stiluri și interactivitate pentru a îmbunătăți aspectul și funcționalitatea site-ului.

HTML este punctul de plecare pentru crearea site-urilor web. Înțelegerea sa este crucială pentru orice rol în dezvoltarea sau designul web.

Printre beneficiile sale se numără:

|  |  |
| --- | --- |
| BENEFICIU | DESCRIERE |
| Ușor de învățat și utilizat | HTML este prietenos pentru începători și reprezintă o introducere ideală în lumea dezvoltării web. |
| Limbaj universal al web-ului | Este standardul global pentru structura paginilor web și esențial pentru gestionarea conținutului online. |
| Platformă independentă | Funcționează pe orice sistem de operare sau platformă, fiind extrem de versatil. |
| Permite adăugarea de multimedia | Integrarea imaginilor, videoclipurilor, fișierelor audio și a hyperlink-urilor pentru îmbunătățirea experienței utilizatorilor. |
| Oportunități de carieră | Deschide uși către roluri diverse (dezvoltator web, manager de conținut, UX/UI designer), cu o cerere în creștere constantă. |
| Personalizarea conținutului | Permite editarea și personalizarea paginilor web, șabloanelor sau emailurilor pentru a răspunde unor cerințe specifice. |
| Bazele dezvoltării web | Reprezintă piatra de temelie pentru dezvoltarea web, oferind o tranziție ușoară spre tehnologii avansate precum CSS și JavaScript. |

**HTML Element și HTML Tags**

HTML Element și HTML Tags sunt elemente cheie ale HTML-ului, fiind strâns legate una de cealalată, deși distincte ca și concept.

HTML Element reprezintă structura completă, care include:

* Eticheta de deschidere
* Conținutul (dacă există)
* Eticheta de închidere

HTML Tag este cuvântul-cheie sau numele scris între paranteze unghiulare (< >), care indică browserului tipul de conținut pe care să îl afișeze.

Pe scurt, elementul HTML include întregul context, iar eticheta este componenta sa principală.

A screen shot of a computer

Description automatically generated

**Structura paginii HTML**

Structura de bază a unei pagini HTML este prezentată mai jos. Aceasta conține elementele de bază esențiale (și anume declarația doctype, elementele HTML, head, title și body) pe baza cărora sunt create toate paginile web.

A screen shot of a computer program

Description automatically generated

* **<!DOCTYPE html> -** reprezintă declarația de tip a documentului, nu o etichetă. Indică faptul că documentul este de tip HTML5.
* **<html lang=”en”> -** acesta este elementul rădăcină al unui document HTML. Toate celelalte elemente sunt conținute în interiorul acestuia.
* **<head> -** conține elementele „din spatele scenei” ale unei pagini web. Aceste elemente nu sunt vizibile pe interfața paginii. Exemple de elemente din interiorul <head> includ:
* **<title> -** definește titlul afișat pe fila browserului.
* **<meta charset= “UTF-8”> -** oferă informații precum setul de caractere utilizat sau setările viewport-ului.
* **<body> -** include tot conținutul vizibil al unei pagini web. Cu alte cuvinte, ceea ce este inclus în <body> va fi afișat de browser pe interfața utilizatorului.

Un document HTML poate fi creat utilizând un editor de text, prin salvarea acestuia cu extensia *.html* sau *.htm* Odată cu salvarea acestuia, poate fi deschis ca pagină web în orice browser.

### 2.3.3 CSS – Cascading Style Sheets

### 2.3.4 JavaScript – JS

### 2.3.3 Bootstrap – Framework bazat pe HTML + CSS +JP



Bootstrap este un framework front-end gratuit utilizat pentru o dezvoltare web mai rapidă și mai ușoară.

Bootstrap include modele de design bazate pe HTML și CSS pentru crearea de formulare, butoane, tabele, navigare, modale, caruseluri de imagini și multe altele, precum și plugin-uri de JavaScript opționale. De asemenea, este web design responsive, adică design-ul este ajustat în mod automat pentru a se mula pe toate tipurile de dispozitive, de la smartphones la desktop-uri de dimensiuni mari.

Avantajele utilizării framework-ului Bootstrap sunt următoarele :

* **Dezvoltare rapidă**  - prin componentele sale deja implementate, permite accelerarea dezvoltării, astfel că finalizarea proiectelor este una mai rapidă ;
* **Responsive** – asigură adaptarea site-urilor web la diverse dispozitive și dimensiuni de ecran, de la smartphon-uri la desktop-uri de dimensiuni mari ;
* **Personalizare flexibilă** -

### 2.2.6 Altele

*2.2.6.1 Maven*

#### 2.2.6.2 Thymeleaf

#### 2.2.6.3 Design.com

#### 2.2.6.4 draw.io

# Capitolul 3 Implementarea aplicației

## 3.1 Structură

Aplicație web este structurată în două părți: cea de frontend și cea de backend. În ceea ce privește partea de frontend, pentru aceasta s-a utilizat framework-ul Bootstrap, care include HTML, CSS și JavaScript. Pentru partea de backend, s-a ales Java Spring Boot.

Referitor la baza de date, opțiunea aleasă a fost PostgreSQL, utilizat pentru stocarea și gestionarea datelor folosite în aplicație.

### 3.1.1 Arhitectură server-side (Back-end)

Arhitectura back-end a aplicației este de tip multi-layer, ceea ce înseamnă că există trei nivele (*straturi*) ale acesteia. Cele trei nivele sunt: presentation layer (*nivelul de prezentare*), business-logic layer (*nivelul logic*), data-acces layer (*nivelul de date*).

**Presentation layer (nivelul de prezentare)**

Nivelul de prezentare este acel nivel accesibil utilizatorilor pe care acești îl pot accesa direct. Practic, reprezintă legătura dintre interfața destinată utilizatorilor și restul aplicației, având rolul de a afișa informații și de a da voie celor care folosesc aplicația să realizeze acțiuni și să introducă date. Responsabilitatea principală a acestui nivel este de a defini și gestiona formatul datelor și criptarea acestora.

În cadrul aplicației, acest nivel este reprezentat de clasele de tip controller, care gestionează cererile primite de la client (browser) și returnează răspunsuri. Cu alte cuvinte, traduce interacțiunile utilizatorului în cereri procesabile de aplicație. Aceste acțiuni reprezintă cereri de tip HTTP (HyperText Transfer Protocol), ele reprezentând ceea ce se numește CRUD: create (POST), read (GET), update(PUT), delete (DELETE). Totodată, acest nivel include și motorul de șabloane Thymeleaf, care reprezintă, în fapt, legătura între front-end și back-end. Acesta este folosit pentru a transforma o pagină HTML statică într-una dinamică, personalizată pentru fiecare utilizator sau cerere.

A screen shot of a computer screen

Description automatically generated

Clasa de tip controller – aferent paginii *Index*

**A black screen with white text

Description automatically generated**

Exemplu fișier HTML cu Thymeleaf inclus

În cele două imagini este ilustrat modul de funcționare al nivelului de prezentare din aplicație.

Prima imagine prezintă o clasă de tip controller, unde adnotarea **@Controller** definește clasa ca fiind un controller, parte din stratul de prezentare. Adnotarea **@GetMapping** marchează metoda ca fiind responsabilă pentru procesarea cererilor HTTP de tip GET, iar **/index** reprezintă punctul de acces (*endpoint*) asociat acestei metode. Acest endpoint permite operațiunea de citire (*READ*) a paginii Index. Thymeleaf utilizează această legătură pentru a conecta front-end-ul cu back-end-ul aplicației.

Cea de-a doua imagine ilustrează modul în care Thymeleaf este utilizat în fișierul HTML pentru a face legătura dintre front-end și back-end. Directivele Thymeleaf, precum **th:href="@{/index}"**, creează o legătură dinamică cu metoda de tip GET din controller, permițând navigarea către endpoint-ul definit.

**Business-Logic Layer (nivelul logic)**

Nivelul logic este nivelul responsabil de funcționarea al aplicației. Acesta acționează ca și intermediar între nivelul de prezentare și cel de acces la date. Totodată, este responsabil cu prelucrarea datelor și manipularea acestora înainte de transmiterea acestora către utilizator sau salvarea lor în baza de date.

Acest nivel asigură îndeplinirea anumitor sarcini precum:

* Validarea datelor de intrare pentru a asigura respectarea regulilor și constrângerilor aferente aplicației
* Efectuarea de operațiuni de manipulare a datelor (de exemplu, calcule)
* Asigurarea punerea în aplicare a regulilor stabilite, cum ar fi de exemplu partea de acces în aplicație și securitatea acesteia
* Gestionează erorile și excepțiile
* Comunică cu nivelul de date în vederea extragerii și stocării datelor.

În cadrul aplicației Complex turistic Luana Nature, acest nivel este definit prin intermediul claselor de tip service. Acestea sunt adnotate cu @Service și prezintă metodele care compun logica aplicației. Un exemplu în acest sens este prezentat în imaginea de mai jos:

A screen shot of a computer code

Description automatically generated

**Data-Access Layer (nivelul de date)**

Nivelul de date reprezintă nivelul care realizează legătura între nivelul logic și sistemul de stocare a datelor. Practic, acest nivel este responsabil pentru efectuarea unor operațiuni precum: conectarea la baza de date și gestionarea acestei conexiuni, mapează datele din sistemul de stocare în obiectele utilizate de aplicație și invers, gestionează erorile și excepțiile legate de accesul la date, generează și execută interogări SQL.

În cadrul aplicației, clasele de tip repository sunt cele aferente acestui nivel, ele fiind de fapt interfețe care extind interfața generică JpaRepository oferită de Spring Data. Ele sunt adnotate cu @Repository. O clasă de acest tip arată astfel:

A screen shot of a computer

Description automatically generated

Cele trei nivele prezentate funcționază în cadrul aplicației Complex turistic Luana Nature așa cum este prezent în imaginea de mai jos:

A diagram of a software development process

Description automatically generated

### 3.1.2 Arhitectură client-side (Front-end)

Arhitectura client-side are la bază modelul MVC (Model-View-Controller), care reprezintă fundamentul construirii aplicațiilor web dezvoltate cu tehnologiile Spring și Spring Boot. Modelul MVC asigură separarea responsabilităților în cadrul aplicației, fiind împărțit în trei componente principale:

* **Componenta Model** – este responsabilă de gestionarea și încapsularea datelor aplicației. Aceasta este, de regulă, compusă din obiecte de tip POJO (Plain Old Java Objects), care reprezintă structurile de date utilizate în aplicație.
* **Componenta View** – se ocupă cu afișarea datelor provenite din Model și generează output sub formă de pagini HTML, care pot fi interpretate de browser-ul utilizatorului. În cazul utilizării framework-ului Thymeleaf, componentele View sunt șabloane HTML dinamice care integrează datele furnizate de Controller.
* **Componenta Controller** – are rolul de a gestiona cererile primite de la utilizator (de exemplu, cererile HTTP) și de a construi un Model adecvat. Acest Model este transmis ulterior componentei View pentru afișarea informațiilor într-un format accesibil utilizatorului.

A diagram of a spring mvc architecture

Description automatically generated

În cadrul aplicației Complex turistic Luana Nature, structura proiectului care stă la baza construcției acesteia, este împărțit pe pachete, iar în componența acestora sunt create clasele necesare dezvoltării aplicației. Astfel, avem un pachet principal  *com.example.Luana\_Nature*, în cadrul căruia sunt definite restul pachetelor care au ca scop organizarea claselor necesare în funcție de tipul lor. Acesta este definit sub directorul *src/main/java.*

Astfel, avem un pachet *model* dedicat claselor folosite pentru definirea entităților. Pachetul *controller*  este cel care conține clasele de tip controller în cadrul cărora sunt definite metodele aferente cererilor de tip CRUD. Pachetul *repository* înglobează toate clasele utilizate pentru comunicarea cu sistemul de gestionare a bazei de date, clase de tip repository. Pachetul *service* conține acele clase care gestionează partea de bussiness layer.

A screenshot of a computer program

Description automatically generated

Pe lângă aceste pachete create manual de dezvoltatorul aplicației, în proiect mai sunt și directoare create automat. Primul este *resources* și în cadrul său sunt create și gestionate atât fișierele de tip .html, .css, .js (*folder-ul templates)* cât și fișierele adiționale necesare în proiect (*de exemplu, poze sau fișiere de tip .txt – salvate în folder-ul static*) .

A screenshot of a computer

Description automatically generated

Cel de-al doilea director automat este cel aferent claselor de test, în care dezvoltatorul va scrie metodele de test utilizate pentru verificarea funcționării aplicației.

A screenshot of a computer

Description automatically generated

### 3.1.3 Baza de date

Sistemul de gestionare a datelor folosit este PostgreSQL, iar structura bazei de date este prezentată în cele ce urmează. Astfel, schema acesteia se poate vizualiza în imaginea de mai jos, aceasta prezentând și relațiile dintre entități.

Din schema prezentată se pot observa mai multe entități esențiale pentru dezvoltarea aplicației și anume:

1. **User** - reprezintă un utilizator al aplicației și conține informații fundamentale despre acesta, cum ar fi numele, adresa de email și username-ul. Aceste informații sunt esențiale pentru identificarea și autentificarea utilizatorilor în sistem.
2. **Rezervations** - este responsabilă pentru stocarea datelor referitoare la rezervările făcute de utilizatori. Aceasta include informații precum data sosirii, data plecării, tipul de cazare, catering, activități , precum și legătura cu entitatea User pentru a ști cine a făcut respectiva rezervare.
3. **Reviews** - permite utilizatorilor să adauge recenzii pentru serviciile oferite de aplicație. Fiecare recenzie este asociată unui utilizator. În această entitate sunt stocate informații precum rating-ul acordat (de exemplu, stele). Astfel, Reviews ajută la obținerea unui feedback valoros și la îmbunătățirea serviciilor oferite.
4. **Contact** - poate conține informații referitoare la modalitățile prin care utilizatorii pot lua legătura cu administrația aplicației. Acestea pot include detalii precum numărul de telefon, adresa de email, adresa fizică sau orice alte informații utile pentru suportul clienților. Este de asemenea conectată la User.
5. **Products** - reprezintă produsele disponibile în aplicație în vederea achiziționării lor. Aceasta conține informații detaliate despre fiecare produs, precum denumirea, descrierea, prețul și disponibilitatea.
6. **Orders** - reprezintă comenzile plasate de utilizatori pentru produsele sau serviciile disponibile în aplicație. Aceasta conține informații despre produsul comandat, utilizatorul care a plasat comanda, data comenzii și starea acesteia. Orders ajută la urmărirea tranzcțiilor și gestionarea fluxului de comenzi din aplicație. Inlcude, de asemenea, legături către entitățile User și Products, pentru a asocia fiecare comandă cu utilizatorul respectiv și produsele comandate.

## 3.2 Specificații funcționale

Aplicația Complex turistic Luana Nature este realizată în vederea promovării locației cu același nume, aceasta contribuind la expunerea în mediul online a facilitățiilor oferite de complexul turistic Luana Nature.

### 3.2.1 Actori

Actorii reprezintă entitățiile care utilizează aplicația. Acești sunt: Vizitator, Utilizator. Vizitatorul este acel utilizator care are drepturi doar de vizualizare, pentru acesta fiind restricționat accesul la crearea de rezervări, review-uri sau comenzi de produse. Utilizatorul este entitatea care nu are nici un fel de restricție. Pentru a prezenta schema pentru fiecare dintre aceste entități s-au utilizat diagrame de tip UML.

A diagram of a diagram

Description automatically generated

A diagram of a person's structure

Description automatically generated

### 3.2.2 Implementare în Spring Boot

**Fișierele de configurare**

**pom.xml**

Primul pas în implementarea proiectului îl reprezintă setarea fișierelor de configurare. Acestea sunt *pom.xml* și *application.properties,* ambele create în mod automat prin intițierea proiectului cu *spring initializr.*

Fișierul pom.xml este specific proiectelor care utilizează Maven, iar în acesta sunt trecute toate dependențele necesare aplicației. Fără menționarea acestor dependețe, proiectul nu poate funcționa. În cazul aplicației de față s-au utilizat următoarele dependențe:

* **Spring Boot Starter Web** – oferă funcționalitățile de bază pentru dezvoltarea unei aplicații de tip web, inclusiv serverul Tomcat încorporat (*un server web open-source care permite rularea aplicațiilor web construite cu Java);*
* **Spring Boot Starter Data JPA** – permite interacțiunea cu baza de date folosind ORM (Object-Relational Mapping), aceasta fiind o tehnică de programare care facilitează legătura dintre obiectele unui limbaj de programare și tabele bazei de date;
* **Spring Boot Starter Thymeleaf** – facilitează utilizarea șabloanelor Thymeleaf pentru generarea dinamică a paginilor HTML.;
* **Spring Boot Starter Mail** – folosită pentru trimiterea automată de e-mailuri, de exemplu, notificări de confirmare a unei rezervări ;
* **Driver PostgreSQL** – asigură conectivitatea cu baza de date PostgreSQL, utilizată pentru stocarea datelor aplicației ;
* **Lombok** – simplifică codul prin generarea automată a metodelor de tip getter, setter, constructori ;
* **JUnit** – o bibliotecă pentru testarea aplicației, folosită pentru verificarea și validarea funcționalităților implementate.

O astfel de dependență este definită cu ajutorul elementul <*dependency>:*

*A screen shot of a computer

Description automatically generated*

De asemenea, fișierul pom.xml mai conține date despre denumirea aplicației, versiunea acesteia și descrierea ei.

A screen shot of a computer code

Description automatically generated

**application.properties**

În acest fișier se setează portul pe care va rula aplicația (*de obicei 8080*) și de asemenea, conține setările pentru baza de date, email, Thymeleaf. Astfel, în aplicație s-au folosit:

1. Setarea bazei de date: denumire, username și parolă

**spring.datasource.url=jdbc:postgresql://localhost:5432/luananature**

**spring.datasource.username=postgres**

**spring.datasource.password=4891**

1. Setarea pentru JPA

**spring.jpa.hibernate.ddl-auto=update  
spring.jpa.hibernate.show-sql=true  
logging.level.org.springframework.security=DEBUG**

1. Setarea pentru Thymeleaf

**spring.thymeleaf.prefix=classpath:/templates/  
spring.thymeleaf.suffix=.html  
spring.thymeleaf.cache=false**

1. Setarea pentru serviciul de email

**spring.mail.host=smtp.gmail.com  
spring.mail.port=587  
spring.mail.username=luananature2024@googlemail.com  
spring.mail.password=zyty qrir cdzz bufb   
spring.mail.properties.mail.smtp.auth=true  
spring.mail.properties.mail.smtp.starttls.enable=true**

**SpringBoot Application**

Pentru a seta o aplicație dedicată Spring Boot este necesară utilizarea adnotării @SpringBootApplication în clasa principală care reprezintă punctul de start al proiectului.

A screen shot of a computer program

Description automatically generated

**Clasele de entități**

Clasele de entități sau modelele, reprezintă acele clase care evidențiază tabelele din baza de date. Acestea conțin parametrii care sunt, de fapt, coloanele din tabele. Totodată, în cadrul acestora se pot specifica relații între entități și se pot seta chei străine1 între acestea.

A screen shot of a computer program

Description automatically generated

Parametrii *name, email, phone, company, arrivalDate, departureDate, numberOfPersons, accomodationType, cateringType, cateringMentions, drinkType, message, activity, period* reprezintă denumirea coloanele cu aceleași nume în baza de date. Pe lângă acestea mai există parametrul  *reservationId*  care reprezintă cheia primară din tabela rezervărilor. Totodată, clasa mai conține relația cu entitatea User.

Pe lângă parametrii, clasa are specificat pachetul din care face parte (*package com.example.Luana\_Nature.model*) și librările necesare atât pentru folosirea parametrilor de tip dată, cât și pentru implementarea adnotărilor și implicit, a funcționalităților acestora.

* **import jakarta.persistence.\*** - pentru adnotările @Entity, @Table, @Id, @GeneratedValue, @ManyToOne,@JoinColumn
* **import lombok.\*** - pentru adnotările @AllArgsConstructor, @NoArgsConstructor, @Data
* **import java.time.LocalDate** – utilizată la definirea parametrilor *arrivalDate și departureDate.*

În aceea ce privește adnotările folosite, funcționalitările acestora sunt descrise în cele ce urmează.

**@Data** **(din Lombok)** – face parte din librăria Lombok și este folosită pentru generarea automată a metodelor de tip getter(), setter(), toString(), equals(), hashCode(). Totodată, poate genera un constructor care acceptă parametrii de tip final și nenuli. Practic, printr utilizarea acestei adnotări, nu mai este necesară scrierea manuală a metodelor amintite. Asfel, se evită scrierea codului repetitiv (*boilerplate).*

**@AllArgsConstructor (din Lombok)** - generează automat un constructor care primește ca parametri toate câmpurile din clasă, inclusiv câmpurile final (*dacă sunt prezente*).

**@NoArgsConstructor (din Lombok)** - generează un constructor implicit, fără parametri. Acest lucru este util mai ales pentru entitățile JPA, care necesită un constructor fără parametri pentru a funcționa corect.

**@Entity (din Jakarta Persistence**) - marchează o clasă ca fiind o entitate JPA (Java Persistence API). Aceasta înseamnă că obiectele acestei clase vor fi stocate într-o bază de date relațională.

**@Table (din Jakarta Persistence) -** este utilizată pentru a lega o entitate Java la o tabelă din baza de date

**@Id (din Jakarta Persistence**) – setează câmpul ca fiiind cheia primară a entității **@GeneratedValue (din Jakarta Persistence**) - marchează câmpul cheii primare ca fiind generat automat de baza de date

**@ManyToOne (din Jakarta Persistence**) - definește o relație de tipul mulți-la-unu între entități **@JoinColumn (din Jakarta Persistence**) - este folosită pentru a specifica coloana din tabela bazei de date care va ține referința către cheia primară a entității la care se face referire.

**Clasele de tip repository**

Clasele de tip repository sunt componente care gestionează interacțiunea cu baza de date, fiind o parte esențială a arhitecturii de tip Data Access Layer (DAL). Acestea sunt folosite pentru a efectua operațiuni de citire, scriere, actualizare și ștergere (operațiile CRUD) pe entitățile definite într-o aplicație. Prin ele, se elimină scrierea manuală a interogărilor SQL. Însă, în cadrul lor se pot scrie interogări customizate. O astfel de clasă este prezentată în imaginea de mai jos:

