***Implementarea unei aplicatii de gestiune***

***a competitiilor sportive***

1. **Scopul lucrarii:**

**Implementarea unei aplicatii web pentru gestionarea competitilor sportive presupune dezvoltarea unei platforme care sa permita administrarea eficienta a datelor legate de evenimentele sportive, participanti, meciuri, echipe, jucatori si alte aspecte specifice acestora. Aplicatia va fi conectata la o baza de date PostgreSQL, care va contine 12 tabele interconectate, fiecare dedicat unui aspect specific al competitilor (de exemplu, echipe, jucatori, meciuri, spectatori, competitii, locatii, sponsori etc.), unde au loc diverse relatii (one-to-one, one-to-many, many-to-many). Operatiile de manipulare a datelor vor include inserarea, actualizarea, stergerea si interogarea acestora, asigurandu-se astfel gestionarea corecta si rapida a informatiilor in cadrul platformei. Aceste tabele vor fi structurate pentru a facilita accesul rapid si eficient la date, iar interfata web va oferi o experienta intuitiva utilizatorilor. Fiind o aplicatie web, am folosit 'Thymeleaf' pentru a crea interfata.**

**Schema a tabelelor:**

**O imagine care conține text, captură de ecran, diagramă, Paralel

Descriere generată automat**

1. **Functionalitati:**

**Aceasta aplicatie continue o varietate de functionalitati:**

1. **Permisiunea de a te loga in aplicatie cu email si parola. Tin sa mentionez ca parola este si ea stocata in baza de date printr-o functie de hashing, astfel incat creez si o masura de securitate, exact ca in problemele din viata de zi cu zi. Mai jos se poate vedea implementarea acesteia:**

**O imagine care conține software, Software multimedia, text, Software de grafică

Descriere generată automat**

1. **Optiunea de a te putea inregistra, alegand unul din cele 3 roluri puse la dispozitie. Aceasta operatie executa in baza mea de date o noua inregistrare. Exista evident, si un mecanism de validare a parolei, care te constrange sa alegi o parola cu minim 10 caractere si minim 2 numere.**
2. **Implementarea a 3 roluri de logare, precum manager, echipa sau spectator, unde fiecare are o anumita functionalitate (spectatorul poate vedea pe site biletele la meciurile dintr-o anumita data, echipa isi poate vedea lotul si echipamentele de care dispun, managerul isi poate vedea toate competitiile).**
3. **Creearea mecanismului de a accesa o anumita pagina numai cand esti logat, prin intermediul ‘HTTPSession’ implementat cu ajutorul ‘Jakarta Servlet’. Astfel, atunci cand tu doresti sa accesezi pagina principala a managerilor, dart u nu esti logat, sa nu ti se permita, redirectionandu-te inapoi la pagina de login.**
4. **Afisarea anumitor statistici pe pagina principala, cum ar fi ce face fiecare rol, care este cel mai predominant sport in competitii, competitia cu cei mai multi sponsori etc.**

**e) Crearea mai multor API-uri REST pentru a permite altor servere sa extraga informatii de pe site-ul meu sau sa efectueze anumite modificari, prin intermediul cererilor HTTP de tip GET sau POST.**

**Explicatie:**

**• API-uri REST se refera la interfetele de programare a aplicatiilor care urmeaza arhitectura REST (Representational State Transfer).**

**• Cererilor HTTP de tip GET sau POST specifica tipurile de metode HTTP utilizate: GET pentru extragerea datelor si POST pentru trimiterea de date sau efectuarea unor modificari pe server. In cadrul unui astfel de sistem, serverele externe ar putea solicita informatii de pe site-ul tau (de exemplu, printr-o cerere GET), sau ar putea trimite date pentru a modifica informatiile (prin cereri POST). Acest lucru ar permite integrarea cu diverse aplicatii sau servicii externe.**

**Poze din aplicatia de depanare ‘Postman’:**

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

Descriere generată automat**

1. **Testarea metodei de a extrage spectatorul care merge la cele mai multe meciuri, cu ajutorul Junit si Mockito.**

**Mockito este o biblioteca care ne permite sa realizam testele prin abordarea ‘mock’. Un ‘mock’ reprezinta o simulare cat mai buna a unui obiect. Noi in aplicatie putem observa cum declaram aceste obiecte, iar in functie ‘setUp’ care foloseste ‘@BeforeEach’(executa acest block inaintea testarii) instantiem aceste obiecte.**

**Poze cu clasa ‘HomeControllerTest’:**

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

Descriere generată automat**

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

Descriere generată automat**

1. **Explicarea fiecarei cereri SQL:**

**In ceea ce priveste lucrul cu baza de date, am folosit ‘jdbcTemplate’ pentru a putea lucra cu cereri create de mine, si nu pentru a folosi framework-uri bazate pe ORM (Object Relational Mapping) precum ‘Hibernate’. Aplicatia mea foloseste:**

1. **3 operatii de inserare, specifice fiecarui rol, unde fiecare cerere este de tip ‘INSERT’, adica introduce o noua linie in baza mea de date. Cererea contine si parametru variabil, mai exact id-ul fiecarui rol este determinat la rulare, datorita constrangerii de integritate ‘IDENTITY’ impusa in baza mea de date (pentru fiecare inregistrare, incrementeaza automat ID-ul pana se ating 1000 de inregistrari distincte – asa am ales eu pragul, poate fi schimbat).**

**Exemplu:**

**O imagine care conține text, captură de ecran, software, Font

Descriere generată automat**

**Se poate observa ca toate campurile specifice coloanelor din tabela noastra de manageri continue parametrii variabili, care vor fi alesi prin intermediul interfetei implementate cu ajutorul ‘Thymeleaf’. Analog si pentru celelalte roluri.**

**Pe langa operatiile de inserare, mai am si doua operatii de modificare, ambele in cadrul competitiilor, care ii permite managerului sa modifice atat numele unei competitii, cat si data sustinerii acesteia.**

**Implementarea acestora:**

**O imagine care conține text, captură de ecran, Font

Descriere generată automat**

**Din cauza constrangerii de integritate pe coloana ‘data\_competitie’ de tip ‘timestamp without varying zone’, care se afla in antiteza cu tipul din HTML ‘datetime-local’, a trebuit sa realizez o conversie mai rafinata, exemplificata in urmatoarea poza:**

**O imagine care conține text, captură de ecran

Descriere generată automat**

**Se poate observa in clar discrepanta intre ‘pattern’-uri.**

**Mai sunt prezente si 3 operatii de stergere, care ii permit unui manager sa isi anuleze o competitie, sau unui spectator sa isi stearga un bilet la un meci. Ambele actiuni realizeaza o redirectionare catre o alta pagina, unde mai esti intrebat daca vrei cu siguranta sa faci actiune. Daca da, se realizeaza cu success si vei fi redirectionat inapoi, daca nu vei fi doar redirectionat inapoi.**

1. **6 interogari simple, care presupun operatia de ‘JOIN’ intre 2 sau mai multe tabele.**

**Primele 3 constau in realizarea paginii principale a fiecarui rol. Spectatorul isi poate vedea numele, pretul biletelor si data meciului prin urmatoarea cerere:**

***SELECT   
 s.id\_spectator AS spectatorId,  
 m.id\_meci AS matchId,  
 s.nume\_spectator AS spectatorName,   
 ms.pret\_bilet AS ticketPrice,   
 m.data\_sustinerii AS matchDate  
FROM   
 spectatori s   
JOIN   
 meciurispectatori ms ON s.id\_spectator = ms.id\_spectator   
JOIN   
 meciuri m ON ms.id\_meci = m.id\_meci   
WHERE   
 s.id\_spectator = ?;***

**Echipele isi pot vedea numele, lotul, jucatorii si culoarea echipamentul prin urmatoarea cerere:**

***SELECT   
 e.nume\_echipa AS team\_name,  
 e.lot\_echipa AS team\_size,  
 j.nume\_jucator AS player\_name,  
 mp.culoare\_echipament AS equipment\_color  
FROM   
 echipe e  
LEFT JOIN   
 jucatori j ON e.id\_echipa = j.id\_echipa  
LEFT JOIN   
 meciurijucatori mp ON j.id\_jucator = mp.id\_jucator  
WHERE   
 e.id\_echipa = ?***

**Managerul isi poate vedea numele competitiilor, data competitiilor, numele orasului locatiei si numele sportului prin urmatoarea cerere:**

***SELECT c.id\_competitie,   
 c.nume\_competitie,   
 c.data\_competitie,   
 c.id\_locatie,   
 c.id\_sport,   
 c.id\_manager,   
 (SELECT nume\_oras   
 FROM (SELECT l.nume\_oras   
 FROM locatii l   
 WHERE l.id\_locatie = c.id\_locatie) AS location\_subquery) AS nume\_oras,   
 (SELECT nume\_sport   
 FROM (SELECT s.nume\_sport   
 FROM sporturi s   
 WHERE s.id\_sport = c.id\_sport) AS sport\_subquery) AS nume\_sport  
FROM competitii c  
WHERE c.id\_manager = ?  
ORDER BY c.id\_competitie ASC***

**Cu toate ca am folosit si subcereri, am folosit ‘JOIN’-uri si voi cataloga aceasta cerere ca fiind simpla.**

**Pe pagina principala, se pot observa cateva statistici. Numele competitiei care gazduieste cele mai multe meciuri este aleasa cu ajutorul cererii:**

***SELECT c.nume\_competitie  
FROM competitii c  
JOIN meciuri m ON c.id\_competitie = m.id\_competitie  
GROUP BY c.id\_competitie, c.nume\_competitie  
ORDER BY COUNT(m.id\_meci) DESC  
LIMIT 1***

**Numele sportului care are cei mai multi sponsori este ales cu ajutorul cererii:**

***SELECT s.nume\_sport  
FROM sporturi s  
JOIN competitii c ON s.id\_sport = c.id\_sport  
JOIN sponsorizari sp ON c.id\_competitie = sp.id\_competitie  
GROUP BY s.id\_sport, s.nume\_sport  
ORDER BY COUNT(sp.id\_sponsor) DESC  
LIMIT 1***

**Numele si prenumele managerului cu cele mai multe competitii este ales cu ajutorul cererii:**

***SELECT m.nume\_manager || ' ' || m.prenume\_manager AS full\_name   
 FROM manageri m   
 JOIN (  
 SELECT id\_manager, COUNT(\*) AS num\_competitii   
 FROM competitii   
 GROUP BY id\_manager   
 ORDER BY num\_competitii DESC   
 LIMIT 1  
) c   
ON m.id\_manager = c.id\_manager***

**Pentru simplitate, am folosit ‘LIMIT 1’ pentru a extrage doar o inregistrare din fiecare tabela.**

1. **4 interogari complexe, care reprezinta statisticile aflate pe pagina principala**

**Cel mai popular sport, din cadrul tuturor competitiilor, este dat de cererea:**

***SELECT s.nume\_sport  
FROM sporturi s  
WHERE s.id\_sport = (  
 SELECT id\_sport  
 FROM (  
 SELECT id\_sport, COUNT(\*) AS competition\_count  
 FROM competitii  
 GROUP BY id\_sport  
 ) subquery  
 WHERE competition\_count = (  
 SELECT MAX(competition\_count)  
 FROM (  
 SELECT COUNT(\*) AS competition\_count  
 FROM competitii  
 GROUP BY id\_sport  
 ) inner\_subquery  
 )  
)  
LIMIT 1***

**Sportul care are cei mai multi sustinatori (sponsori) este dat de cererea:**

***SELECT c.nume\_competitie  
FROM competitii c  
WHERE c.id\_competitie = (  
 SELECT id\_competitie  
 FROM (  
 SELECT id\_competitie, COUNT(\*) AS sponsor\_count  
 FROM sponsorizari  
 GROUP BY id\_competitie  
 ) subquery  
 WHERE sponsor\_count = (  
 SELECT MAX(sponsor\_count)  
 FROM (  
 SELECT COUNT(\*) AS sponsor\_count  
 FROM sponsorizari  
 GROUP BY id\_competitie  
 ) inner\_subquery  
 )  
)  
LIMIT 1***

**Numele orasului care tine cele mai multe competitii sunt date de cererea:  
  
*SELECT l.nume\_oras  
FROM locatii l  
WHERE l.id\_locatie = (  
 SELECT id\_locatie  
 FROM (  
 SELECT id\_locatie, COUNT(\*) AS competition\_count  
 FROM competitii  
 GROUP BY id\_locatie  
 ) subquery  
 WHERE competition\_count = (  
 SELECT MAX(competition\_count)  
 FROM (  
 SELECT COUNT(\*) AS competition\_count  
 FROM competitii  
 GROUP BY id\_locatie  
 ) inner\_subquery  
 )  
)  
LIMIT 1***

**Spectatorul cu cele mai multe tichete este dat de cererea:**

***SELECT s.nume\_spectator, s.prenume\_spectator, COUNT(ms.id\_meci) AS nr\_meciuri  
FROM meciurispectatori ms  
JOIN spectatori s ON ms.id\_spectator = s.id\_spectator  
GROUP BY s.id\_spectator, s.nume\_spectator, s.prenume\_spectator  
HAVING COUNT(ms.id\_meci) = (  
 SELECT MAX(nr\_meciuri)  
 FROM (  
 SELECT COUNT(ms1.id\_meci) AS nr\_meciuri  
 FROM meciurispectatori ms1  
 GROUP BY ms1.id\_spectator  
 )  
) LIMIT 1***

**Toate aceste cereri folosesc ‘LIMIT 1’, pentru a extrage doar o inregistrare din baza noastra de date.**

1. **Descrierea fiecarei tabele din baza de date (clase):**

**Dupa cum am mentionat, baza mea de date consta in 12 tabele, unde se pot observa coloanele fiecareia in diagrama pusa la inceput:**

1. **Competitii – tabela care reprezinta fiecare competitive in parte. Are relatii cu mai multe tabele (locatii, sporturi, manageri, meciuri) de tip one-to-many, si o relatie de tip many-to-many cu tabela sponsori, prin intermediul tabelei sponsorizari**
2. **Meciuri – tabela reprezinta partidele din cadrul fiecarei competitii, se leaga de tabela spectator prin intermediul tabelei meciurispectatori.**
3. **Spectatori – tabela care reprezinta oamenii ce vor veni la meciurile din cadrul unei competitii**
4. **Jucatori – tabela care reprezinta jucatorii ce vor juca intr-un meci, se leaga si cu tabela echipe, fiecare jucator fiind la o echipa**
5. **Echipe – tabela care reprezinta echipele ce fac parte intr-o anumita competitive**
6. **Manager – tabela care reprezinta oamenii ce manageriaza o anumita competitie**

**Evident, deoarece am folosit modelul Spring MVC (Model-View-Controller), fiecare tabela este ilustrata ca o clasa in aplicatia noastra (modele ce se afla in package-ul ‘models’). In aplicatie se poate vedea clar cum am reprezentat fiecare tabela prin intermediul POJO-urilor (Plain Old Java Object). Se putea folosi si conceptul de ‘record’, care ar facilita functionalitatea si arm ai reduce cateva linii de cod.**

1. **Interfata grafica:**

**Interfata grafica a fost realizata in ‘Thymeleaf’ un motor de template (template engine) care permite utilizarea HTML si CSS. Pe langa estetica pe care trebuie sa o aibe orice aplicatie, am folosit aceste limbaje pentru a creea si metode de POST sau de GET, prin intermediul functiilor implementate in ‘backend’. Astfel, la apasarea unui buton de exemplu, se executa o anumita actiune (de exemplu anularea unui bilet de catre un spectator).**

**Pagina de pornire este singura care foloseste Javascript, pentru a realiza functionalitatea de meniu, unde poti alege daca vrei sa te loghezi sau vrei sa iti creezi cont pe aplicatie.**

1. **Testarea aplicatiei**

**In cadrul functionalitatilor implementate, am utilizat testarea unitara a API-urilor pentru a demonstra modul de functionare al metodei getTopSpectators, care are rolul de a returna persoana cu cele mai multe bilete achizitionate pentru evenimente sportive. Testarea unitara este esentiala pentru verificarea functionalitatii izolate a componentelor aplicatiei, asigurand corectitudinea logicii implementate.**

**De asemenea, am integrat utilizarea Mockito, un cadru dedicat simularii (*mocking*) in testarea unitara. Acesta permite testarea metodei fara a depinde de rularea efectiva a aplicatiei, prin inlocuirea componentelor externe sau a dependintelor cu obiecte simulate. Conceptul de *mock* faciliteaza izolarea metodei getTopSpectators si evaluarea comportamentului acesteia intr-un mediu controlat, contribuind astfel la identificarea rapida a potentialelor probleme si la imbunatatirea calitatii codului.**

1. **Cateva idei care pot permite imbunatatirea aplicatiei:**
2. **Daca ar fi sa folosim aceasta aplicatie in lumea reala, ar trebui sa fac fiecare pagina web ‘responsive’ si putin mai complexa la design.**
3. **Folosirea record-urilor in locul POJO-urilor in cadrul modelelor, deoarece permite o citire si o implementare mai usoara.**
4. **Implementarea unor functionalitati de validare a anumitor campuri mai avansate, de exemplu sa folosesc regex-uri la verificarea email-urilor.**
5. **Pentru creearea de conturi de manageri sau de echipe, sa se ceara anumite verificari, pentru ca nu oricine poate face acest lucru. Am putea de exemplu ca atunci cand se creeaza un cont de acest gen, sa verificam noi manual daca acea persoana ar trebui sa aibe un astfel de cont.**
6. **Cu toate ca am destul de multe tabele, ar trebui sa mai gandesc cateva operatii intre ele pe care le poate efectua cineva, deoarece eu am implementat doar functionalitati de baza**
7. **O implementare a verificarii email-ului mai in amanunt, deoarece cea facuta in HTML poate permite cateva cazuri de mail-uri invalide (am putea folosi regex-uri de exemplu)**
8. **Posibila implementare a unul rol numit ‘admin’ ca atunci cand cineva vrea sa isi faca un cont cu privilegii mai mari (de ex. manager) sa fie validat de catre un administrator al aplicatiei, pe baza mai multor criterii.**