Aplicație de gestionare a echipamentelor dintr-o sală

realizat de Ionescu Vlad-Andrei, grupa 331AA

I) Specificare funcționalități de bază și suplimentare

1) Aplicația permite utilizatorului să vizualizeze o colecție de obiecte de tip referință (sală, echipamente, furnizori, utilizatori).

2) Utilizatorul poate adăuga un nou element în listă prin intermediul unei interfețe care permite introducerea parametrilor de inițializare ai unui obiect din listă și adăugarea acestuia la colecție. Sunt implementate soluții de validare a câmpurilor introduse. De exemplu, pentru o sală nouă, se verifică dacă valoarea introdusă pentru capacitate este un număr pozitiv.

3) Utilizatorul poate modifica un element din lista de obiecte. Modificările sunt validate conform restricțiilor specificate pentru fiecare câmp.

4) Utilizatorul poate șterge un obiect din colecția de obiecte. Ștergerea este confirmată de utilizator înainte de a fi efectuată efectiv, printr-un mesaj de confirmare („Sigur doriți să ștergeți acest element?”) și apăsarea butonului „OK”.

II) Descrierea claselor/metodelor/atributelor proiectului

a) Clase pentru entități:

1. Clasa Utilizator

Reprezintă entitatea Utilizator, mapată la tabela utilizator din baza de date.

Atribute : Atributele sunt de tip private și corespund coloanelor din baza de date a tabelei utilizator.

utilizator_id (Integer): ID-ul unic al utilizatorului, generat automat.

nume (String): Numele utilizatorului, NOT NULL, cu o lungime maximă de 50 caractere.

tip_utilizator (String): Tipul utilizatorului, cu o lungime maximă de 50 caractere.

Metode: Sunt de tip public și permit accesul și modificarea valorilor atributelor.

Gettere și settere: getUtilizator_id(), setUtilizator_id(Integer utilizator_id), getNume(), setNume(String nume), getTip_utilizator(), setTip_utilizator(String tip_utilizator).

2. Clasa Sala

Reprezintă entitatea Sala, mapată la tabela sala din baza de date.

Atribute : Atributele sunt de tip private și corespund coloanelor din baza de date a tabelei sala.

sala_id (Integer): ID-ul unic al sălii, generat automat.

nume (String): Numele sălii, NOT NULL, cu o lungime maximă de 50 caractere.

capacitate (Integer): Capacitatea sălii, exprimată ca număr întreg.

Metode: Sunt de tip public și permit accesul și modificarea valorilor atributelor.

Gettere și settere: getSala_id(), setSala_id(Integer sala_id), getNume(), setNume(String nume), getCapacitate(), setCapacitate(Integer capacitate).

3. Clasa Furnizor

Reprezintă entitatea Furnizor, mapată la tabela furnizor.

Atribute:Atributele sunt de tip private și corespund coloanelor din baza de date a tabelei furnizor.

furnizor_id (Integer): ID-ul unic al furnizorului, generat automat.

nume (String): Numele furnizorului, NOT NULL, cu o lungime maximă de 50 caractere.

adresa (String): Adresa furnizorului, cu o lungime maximă de 100 caractere.

telefon (String): Numărul de telefon, cu o lungime maximă de 15 caractere.

Metode: Sunt de tip public și permit accesul și modificarea valorilor atributelor.

Gettere și settere: getFurnizor_id(), setFurnizor_id(Integer furnizor_id), getNume(), setNume(String nume), getAdresa(), setAdresa(String adresa), getTelefon(), setTelefon(String telefon).

4. Clasa Echipament

Reprezintă entitatea Echipament, mapată la tabela echipament.

Atribute : Atributele sunt de tip private și corespund coloanelor din baza de date a tabelei echipament.

echipament_id (Integer): ID-ul unic al echipamentului, generat automat.

nume (String): Numele echipamentului, NOT NULL, cu o lungime maximă de 50 caractere.

Metode: Sunt de tip public și permit accesul și modificarea valorilor atributelor.

Gettere și settere: getEchipament_id(), setEchipament_id(Integer echipament_id), getNume(), setNume(String nume)

b) Interfețe pentru entități: Acestea sunt folosite pentru a abstractiza operațiile CRUD, fără necesitatea de a scrie cod SQL explicit:

1) UtilizatorRepository
Extinde JpaRepository<Utilizator, Integer> și oferă metode implicite pentru salvare, ștergere, actualizare și căutare a utilizatorilor.

2) SalaRepository
Extinde JpaRepository<Sala, Integer> și oferă metode implicite pentru salvare, ștergere, actualizare și căutare a sălilor.

3) FurnizorRepository
Extinde JpaRepository<Furnizor, Integer> și oferă metode implicite pentru salvare, ștergere, actualizare și căutare a furnizorilor.

4) EchipamentRepository
Extinde JpaRepository<Echipament, Integer> și oferă metode implicite pentru salvare, ștergere, actualizare și căutare a echipamentelor.

c) Clasele pentru operarea controalelor

1. Clasa UtilizatorController

Reprezintă un controller REST utilizat pentru gestionarea entității Utilizator din baza de date.

Atribute: Clasa are un singur atribut de tip privat care este o dependență injectată automat de framework-ul Spring. Este utilizată pentru a accesa funcționalitățile repository-ului asociat entității Utilizator.

utilizatorRepository(UtilizatorRepository): Instanța repository-ului care permite efectuarea operațiilor CRUD asupra entității Utilizator.

Metode: Sunt de tip public și sunt asociate cu endpoint-uri REST pentru gestionarea operațiilor asupra entității Utilizator.

getAllUtilizatori()

Endpoint asociat: GET /api/utilizatori

Descriere: Returnează lista completă de utilizatori din baza de date.

Implementare: Utilizează metoda findAll() din UtilizatorRepository pentru a prelua toate înregistrările. Rezultatul este returnat sub forma unei liste de obiecte de tip Utilizator.

getUtilizatorById(Integer id)

Endpoint asociat: GET /api/utilizatori/{id}

Descriere: Returnează un utilizator specific, identificat prin ID-ul său unic.

Implementare: Verifică existența utilizatorului în baza de date folosind metoda findById() din UtilizatorRepository. Dacă utilizatorul este găsit, este returnat cu un răspuns HTTP 200 (OK). În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

addUtilizator(Utilizator utilizator)

Endpoint asociat: POST /api/utilizatori

Descriere: Creează un nou utilizator pe baza datelor furnizate prin corpul cererii (JSON).

Implementare: Utilizează metoda save() din UtilizatorRepository pentru a adăuga utilizatorul în baza de date. În cazul unui succes, este returnat un răspuns HTTP 201 (Created). Dacă apare o eroare, este returnat un răspuns HTTP 400 (Bad Request).

updateUtilizator(Integer id, Utilizator utilizatorDetails)

Endpoint asociat: PUT /api/utilizatori/{id}

Descriere: Actualizează informațiile unui utilizator existent pe baza ID-ului primit în URL.

Implementare: Metoda verifică dacă utilizatorul există utilizând findById(). Dacă utilizatorul este găsit, valorile atributelor sunt actualizate folosind metodele setter ale clasei Utilizator, iar modificările sunt salvate cu save(). În caz contrar, este returnat un răspuns HTTP 404 (Not Found).

deleteUtilizator(Integer id)

Endpoint asociat: DELETE /api/utilizatori/{id}

Descriere: Șterge un utilizator specific, identificat prin ID-ul său unic.

Implementare: Metoda verifică existența utilizatorului în baza de date utilizând findById(). Dacă utilizatorul este găsit, este șters folosind deleteById(), iar răspunsul HTTP 204 (No Content) este returnat. În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

2. Clasa SalaController

Reprezintă un controller REST utilizat pentru gestionarea entității Sala din baza de date.

Atribute: Clasa are un singur atribut de tip privat care este o dependență injectată automat de framework-ul Spring. Este utilizată pentru a accesa funcționalitățile repository-ului asociat entității Sala.

salaRepository (SalaRepository): Instanța repository-ului care permite efectuarea operațiilor CRUD asupra entității Sala.

Metode: Sunt de tip public și sunt asociate cu endpoint-uri REST pentru gestionarea operațiilor asupra entității Sala.

getAllSale()

Endpoint asociat: GET /api/sala

Descriere: Returnează lista completă de săli din baza de date.

Implementare: Utilizează metoda findAll() din SalaRepository pentru a prelua toate înregistrările. Rezultatul este returnat sub forma unei liste de obiecte de tip Sala.

getSalaById(Integer id)

Endpoint asociat: GET /api/sala/{id}

Descriere: Returnează o sală specifică, identificată prin ID-ul său unic.

Implementare: Verifică existența sălii în baza de date folosind metoda findById() din SalaRepository. Dacă sala este găsită, este returnată cu un răspuns HTTP 200 (OK). În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

addSala(Sala sala)

Endpoint asociat: POST /api/sala

Descriere: Creează o sală nouă pe baza datelor furnizate prin corpul cererii (JSON).

Implementare: Înainte de salvare, verifică dacă valoarea atributului capacitate este mai mare decât 0. Dacă validarea este satisfăcută, utilizează metoda save() din SalaRepository pentru a adăuga sala în baza de date. În cazul unui succes, este returnat un răspuns HTTP 201 (Created). Dacă apare o eroare sau validarea eșuează, este returnat un răspuns HTTP 400 (Bad Request).

deleteSala(Integer id)

Endpoint asociat: DELETE /api/sala/{id}

Descriere: Șterge o sală specifică, identificată prin ID-ul său unic.

Implementare: Verifică existența sălii în baza de date utilizând metoda findById(). Dacă sala este găsită, este eliminată folosind metoda deleteById() din SalaRepository. Un răspuns HTTP 204 (No Content) este returnat pentru succes. În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

updateSala(Integer id, Sala salaDetails)

Endpoint asociat: PUT /api/sala/{id}

Descriere: Actualizează informațiile unei săli existente pe baza ID-ului primit în URL.

Implementare: Metoda verifică dacă sala există în baza de date utilizând findById(). Dacă sala este găsită, valorile atributelor sunt actualizate cu cele noi (dacă acestea sunt specificate și corecte). Modificările sunt salvate folosind metoda save(). Dacă sala nu este găsită, se returnează un răspuns HTTP 404 (Not Found), însoțit de un mesaj care indică faptul că sala nu există. Dacă introducem un

număr negativ la capacitate, atunci este returnat un răspuns HTTP 400 (Bad Request).

3. Clasa EchipamentController

Reprezintă un controller REST utilizat pentru gestionarea entității Echipament din baza de date.

Atribute: Clasa are un singur atribut de tip privat, care este o dependență injectată automat de framework-ul Spring. Este utilizată pentru a accesa funcționalitățile repository-ului asociat entității Echipament.

echipamentRepository (EchipamentRepository): Instanța repository-ului care permite efectuarea operațiilor CRUD asupra entității Echipament.

Metode: Sunt de tip public și sunt asociate cu endpoint-uri REST pentru gestionarea operațiilor asupra entității Echipament.

getAllEchipamente()

Endpoint asociat: GET /api/echipamente

Descriere: Returnează lista completă de echipamente din baza de date.

Implementare: Utilizează metoda findAll() din EchipamentRepository pentru a prelua toate înregistrările. Rezultatul este returnat sub forma unei liste de obiecte de tip Echipament.

getEchipamentById(Integer id)

Endpoint asociat: GET /api/echipamente/{id}

Descriere: Returnează un echipament specific, identificat prin ID-ul său unic.

Implementare: Verifică existența echipamentului în baza de date folosind metoda findById() din EchipamentRepository. Dacă echipamentul este găsit, este returnat cu un răspuns HTTP 200

(OK). În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

addEchipament(Echipament echipament)

Endpoint asociat: POST /api/echipamente

Descriere: Creează un echipament nou pe baza datelor furnizate prin corpul cererii (JSON).

Implementare: Utilizează metoda save() din EchipamentRepository pentru a adăuga echipamentul în baza de date. În cazul unui succes, este returnat un răspuns HTTP 201 (Created). Dacă apare o eroare, este returnat un răspuns HTTP 400 (Bad Request).

updateEchipament(Integer id, Echipament echipamentDetails)

Endpoint asociat: PUT /api/echipamente/{id}

Descriere: Actualizează informațiile unui echipament existent pe baza ID-ului primit în URL.

Implementare: Metoda verifică dacă echipamentul există utilizând findById(). Dacă echipamentul este găsit, valorile atributelor sunt actualizate cu cele noi, iar modificările sunt salvate folosind metoda save(). În caz contrar, este returnat un răspuns HTTP 404 (Not Found).

deleteEchipament(Integer id)

Endpoint asociat: DELETE /api/echipamente/{id}

Descriere: Șterge un echipament specific, identificat prin ID-ul său unic.

Implementare: Metoda verifică existența echipamentului în baza de date utilizând findById(). Dacă echipamentul este găsit, este eliminat folosind metoda deleteById() din EchipamentRepository. Un răspuns HTTP 204 (No Content) este returnat pentru succes. În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

4. Clasa FurnizorController

Reprezintă un controller REST utilizat pentru gestionarea entității Furnizor din baza de date.

Atribute: Clasa include un atribut privat care este injectat automat de framework-ul Spring. Acesta este utilizat pentru a interacționa cu repository-ul asociat entității Furnizor.

furnizorRepository (FurnizorRepository): Instanța repository-ului care permite efectuarea operațiilor CRUD asupra entității Furnizor.

Metode: Metodele sunt de tip public și sunt asociate cu endpoint-uri REST pentru gestionarea operațiilor asupra entității Furnizor.

getAllFurnizori()

> **Endpoint asociat:** GET /api/furnizori
>
> **Descriere:** Returnează lista completă de furnizori din baza de date.
>
> **Implementare:** Utilizează metoda findAll() din FurnizorRepository pentru a prelua toate înregistrările. Rezultatul este returnat sub forma unei liste de obiecte de tip Furnizor.

getFurnizorById(Integer id)

> **Endpoint asociat:** GET /api/furnizori/{id}
>
> **Descriere:** Returnează un furnizor specific, identificat prin ID-ul său unic.
>
> **Implementare:** Metoda verifică existența furnizorului în baza de date utilizând findById(). Dacă furnizorul este găsit, este returnat cu un răspuns HTTP 200 (OK). În caz contrar, se returnează un răspuns HTTP 404 (Not Found).

addFurnizor(Furnizor furnizor)

> **Endpoint asociat:** POST /api/furnizori

Descriere: Creează un furnizor nou pe baza datelor furnizate prin corpul cererii (JSON).

Implementare: Înainte de salvare, validează formatul numărului de telefon utilizând expresia regulată ^07\\d{8}$. Dacă validarea eșuează, este returnat un răspuns HTTP 400 (Bad Request) cu un mesaj descriptiv. Dacă datele sunt valide, metoda save() din FurnizorRepository este utilizată pentru a salva furnizorul în baza de date. La succes, se returnează un răspuns HTTP 201 (Created). În caz de eroare, se returnează un răspuns HTTP 400 (Bad Request).

updateFurnizor(Integer id, Furnizor furnizorDetails)

Endpoint asociat: PUT /api/furnizori/{id}

Descriere: Actualizează informațiile unui furnizor existent pe baza ID-ului primit în URL.

Implementare: Metoda verifică existența furnizorului în baza de date utilizând findById(). Dacă furnizorul este găsit, validează numărul de telefon utilizând aceeași expresie regulată ca în metoda de adăugare. În cazul unui număr de telefon invalid, returnează un răspuns HTTP 400 (Bad Request). Dacă datele sunt valide, atributelor furnizorului (nume, adresă, telefon) le sunt atribuite noile valori, iar modificările sunt salvate folosind metoda save(). Dacă furnizorul nu este găsit, returnează un răspuns HTTP 404 (Not Found).

deleteFurnizor(Integer id)

Endpoint asociat: DELETE /api/furnizori/{id}

Descriere: Șterge un furnizor specific, identificat prin ID-ul său unic.

Implementare: Verifică existența furnizorului utilizând findById(). Dacă furnizorul este găsit, îl șterge folosind metoda deleteById() din FurnizorRepository și returnează un răspuns HTTP 204 (No

Content). Dacă furnizorul nu este găsit, returnează un răspuns HTTP 404 (Not Found).

5. Clasa HomeController

Reprezintă un controller utilizat pentru gestionarea operațiilor asociate paginii principale a aplicației web.

Atribute: Clasa nu are atribute specifice.

Metode: Clasa are o metoda de tip public și gestionează operațiile asociate cu redirecționarea către pagina principală.

home()

Endpoint asociat: GET /

Descriere: Returnează pagina principală a aplicației web.

Implementare: Metoda mapează cererile de tip GET la URL-ul rădăcină ("/"). Returnează un șir de caractere ("proiect") care corespunde numelui fișierului HTML al paginii principale.

d) Clasa principală ProiectApplication

Reprezintă clasa principală care inițializează și rulează aplicația Spring Boot.

Atribute: Clasa nu are atribute specifice.

Metode: Clasa are o metodă de tip `public`, `static`, care rulează aplicația Spring Boot, inițializând contextul și pornind serverul pentru a gestiona cererile HTTP.

main(String[] args)

III) Descrierea elementelor funcționale oferite prin interfața grafică cu utilizatorul.

Utilizatorul poate selecta tabelul dorit prin apăsarea chenarului corespunzător care afișează numele acestuia. Pentru fiecare tabel este disponibil un formular dedicat adăugării de noi înregistrări. Fiecare tabel

include o coloană dedicată acțiunilor, în care se află un buton pentru ștergerea înregistrărilor și un buton pentru modificarea rândurilor existente. La apăsarea butonului de modificare, datele din rândul respectiv sunt preluate automat și afișate într-un formular dedicat editării. După efectuarea modificărilor, dacă datele introduse sunt valide, acestea vor fi salvate și înlocuite automat în tabel, actualizând înregistrarea corespunzătoare. La apăsarea butonului de ștergere, apare un mesaj de confirmare “Sigur doriți să ștergeți acest element?”. Dacă se apasă ok, atunci datele sunt șterse, altfel nu.

IV) Aspecte privind testarea aplicatiei

Pentru a verifica funcționalitatea corectă a aplicației am efectuat următoarele teste:

1. Testarea formularului destinat adăugării de date

Am adăugat noi înregistrări cu toate câmpurile completate corect. Acestea au fost adăugate în tabele cu succes.

Am adăugat înregistrări cu unul sau mai multe câmpuri incomplete pentru a verifica afișarea mesajelor de eroare. Pentru fiecare tabel, s-a afișat mesajul “Please fill out this field”, așa cum trebuia.

Am introdus date invalide (de exemplu, text într-un câmp numeric) și am verificat comportamentului aplicației. Aplicația nu ne-a permis să introducem date invalide.

Am introdus date valide, dar incorecte (de exemplu, un număr de telefon care nu respectă formatul 07XXXXXXXX sau un număr negativ în câmpul destinat capacității unei săli). Aplicația a permis introducerea acestor date în formular, însă, în momentul încercării de a le salva în tabel, acestea nu au fost adăugate.

2. Testarea funcției de modificare

Am selectat un rând existent pentru modificare și verificarea preluării corecte a datelor în formularul dedicat editării. Datele au fost extrase cu succes.

Am modificat un rând cu date valide și am verificat actualizarea corespunzătoare al acestuia în tabel. Datele au fost modificate cu succes.

Am modificat un rând cu date invalide. Aplicația nu ne-a lăsat să modificăm niciun tabel cu date invalide

Am modificat un rând cu date valide, dar incorecte(de exemplu, un număr de telefon care nu respectă formatul 07XXXXXXXX). Aplicația nu ne-a lăsat să modificăm niciun tabel cu date invalide.

Am modificat un rând fără a aduce nicio schimbare, pentru a verifica dacă datele rămân neschimbate. Datele au rămas neschimbate.

3. Testarea funcției de ștergere

Am șters un rând specific din tabel și am verificat eliminarea sa corectă. Totul a funcționat perfect.

4. Testarea afișării tabelelor

Am verificat afișarea corectă a datelor în tabel după adăugarea, modificarea sau ștergerea unei înregistrări. Totul a funcționat perfect.

V) Idei despre cum poate fi îmbunătățită aplicația

1) Implementarea unei funcții de undo care să anuleze o acțiune recentă.

2) Posibilitatea de a căuta și filtra date în funcție de diferite criterii.

3) Implementarea unui sistem de autenficare și autorizare pentru utilizatori, astfel încât doar cei cu permisiuni adecvate sa poată modifica datele.

4) Când utilizatorul adaugă date valide, dar incorecte să-i apară explicit mesaje de eroare care să-i spună de ce datele respective nu au fost adăugate în tabel.

5) API extern pentru validare: De exemplu, validarea numerelor de telefon printr-un serviciu extern pentru a verifica dacă sunt active.