Aplicație pentru gestionarea spectacolelor de stand-up

Student: **Necula Mihai**
Grupa: **331AB**

Cuprins

1 Date generale

Tehnologii folosite: Java, Spring Boot, Spring MVC, Thymeleaf, JDBC (JdbcTemplate), H2, Maven, HTML, CSS, JavaScript.

2 Specificarea funcționalităților

2.1 Funcționalități de bază

2.1.1 Autentificare utilizatori

- Login cu email și parolă.
- Rol „client” și rol „admin” (admin cu email/parolă predefinite).
- Validare pe server pentru câmpuri goale, email invalid și credențiale greșite.

2.1.2 Înregistrare clienți

- Creare cont cu nume, prenume, email, telefon și parolă.
- Validări: câmpuri obligatorii, email valid, telefon cu 10 cifre, parolă minim 4 caractere, email unic.

2.1.3 Gestionare spectacole (admin)

- Listare spectacole existente.
- Adăugare spectacol nou cu:
  - titlu, locație, dată și oră, preț bilet, durată (minute);
  - asociere artiști (1–3 artiști).
- Ștergere spectacole.

2.1.4 Gestionare clienți (admin)

- Listare clienți.
- Editare client (nume, prenume, email, telefon, parolă), cu validare.
- Ștergere client.

2.1.5 Gestionare artiști (admin)

- Listare artiști.
- Editare artist (nume, prenume, email, telefon, data nașterii), cu validare.
- Ștergere artist.
- Gestionare participări (relația artist–spectacol).

2.1.6 Gestionare locații

- Listare locații (nume sală, oraș).
- Ștergere locație.

2.1.7 Cumpărare bilete

- Selectare oraș, apoi spectacol, apoi rând (1–10) și loc (1–20).
- Verificare disponibilitate și creare bilet.
- Afișare mesaje de succes sau eroare.

2.1.8 Vizualizare bilete

- Listă de bilete pentru un client, cu:
  - titlul spectacolului, locație, dată spectacol, rând, loc, preț, dată achiziție.

2.1.9 Listă spectacole pentru client

- Tabel cu spectacole, în care este afișat statusul:
  - „Urmează” pentru spectacole viitoare;
  - „S-a desfășurat” pentru spectacole trecute.
- Statusul este calculat în SQL, pe baza datei spectacolului.

2.2 Funcționalități suplimentare

2.2.1 Filtrare spectacole după oraș

- În pagina clientului, lista de spectacole poate fi filtrată după oraș folosind un dropdown și JavaScript.

2.2.2 Recomandări de spectacole

- Pagină de recomandări pentru client, bazată pe istoricul de bilete achiziționate (metoda ‘getRecommendedShows‘ din ‘ShowService‘).

2.2.3 Rapoarte pentru admin

- Spectacole populare (după numărul de bilete vândute).
- Clienți frecvenți (VIP) – clienți cu cele mai multe participări.
- Performanța artiștilor – număr de spectacole per artist.
- Istoric bilete pentru un client (pe baza unui ‘clientId‘ introdus în dashboard).

2.2.4 Validare server-side completă

- Toate formularele importante (login, signup, editare client, editare artist, adăugare spectacol, cumpărare bilet) sunt validate pe server cu mesaje explicite.
- Atributele HTML ‘required‘ au fost eliminate pentru a nu depinde de validarea din browser.

3 Descrierea claselor și layerelor

3.1 Layer de prezentare – LoginController

Clasa ‘LoginController‘ este un controller Spring MVC care gestionează rutele principale ale aplicației.

- Rute pentru autentificare și înregistrare
  - ‘GET /login‘, ‘POST /login‘ - Afișează formularul de login și procesează autentificarea.
  - ‘GET /signup‘, ‘POST /signup‘ - Afișează formularul de înregistrare client și salvează un client nou după validare.
  - ‘GET /logout‘ - Redirecționează utilizatorul la pagina de login.

```
@PostMapping("/login")
public String loginSubmit(@RequestParam String email,
        @RequestParam String parola,
        Model model) {

    if (email == null || email.isBlank() || parola == null || parola.isBlank()) {
        model.addAttribute("error", "Emailul și parola sunt obligatorii.");
        return "login";
    }

    if (!email.contains(s: "@")) {
        model.addAttribute("error", "Email invalid. Te rugăm să introduci un email valid.");
        return "login";
    }

    if ("admin@admin.com".equals(email) && "admin".equals(parola)) {
        return "redirect:/admin";
    }

    Map<String, Object> client = clientService.autentificare(email, parola);

    if (client != null) {

        model.addAttribute("client", client);

        return "redirect:/welcome?email=" + email;
    You, 2 weeks ago • Incarcare
    } else {

        model.addAttribute("error", "Email sau parolă incorectă");

        return "login";

    }
}

@PostMapping("/signup")
public String signupSubmit(@RequestParam String nume,
                           @RequestParam String prenume,
```

```
@PostMapping("/signup")
public String signupSubmit(@RequestParam String nume,
                           @RequestParam String prenume,
                           @RequestParam String email,
                           @RequestParam String telefon,
                           @RequestParam String parola,
                           Model model) {

    if (nume == null || nume.isBlank() || prenume == null || prenume.isBlank() ||
        email == null || email.isBlank() || telefon == null || telefon.isBlank() ||
        parola == null || parola.isBlank()) {
        model.addAttribute("error", "Toate câmpurile sunt obligatorii.");
        return "signup";
    }

    if(telefon.length() != 10) {
        model.addAttribute("error", "Telefonul trebuie să conțină exact 10 cifre.");
        return "signup";
    }

    if(clientService.getClientByEmail(email) != null) {
        model.addAttribute("error", "Email-ul există deja.");
        return "signup";
    }

    if (!email.contains(s: "@")) {
        model.addAttribute("error", "Email invalid. Te rugăm să introduci un email valid.");
        return "signup";
    }

    if (parola.length() < 4) {
        model.addAttribute("error", "Parola trebuie să conțină cel puțin 4 caractere.");
        return "signup";
    }


    clientService.createClient(nume, prenume, email, telefon, parola);

    return "redirect:/login";
}
```

- Rute pentru client
  - ‘GET /welcome‘ - Pagina principală a clientului după autentificare.
  - ‘GET /spectacole‘ - Afișează lista de spectacole cu statusul fiecăruia.
  - ‘GET /recomandari‘ - Afișează lista de spectacole recomandate.
  - ‘GET /cumpara-bilet‘, ‘POST /cumpara-bilet‘ - Form pentru cumpărare bilet și prelucrarea comenzii (validare pe oraș/spectacol/rând/loc).
  - ‘GET /bilete/idClient‘ - Afișează biletele clientului.

```
public String welcome(@RequestParam String email, HttpSession session, Model model) {

    if (email != null) {

        session.setAttribute("email", email);

    }

    else {

        email = (String) session.getAttribute("email");

    }

    Map<String, Object> client = clientService.getClientByEmail(email);

    model.addAttribute("client", client);

    return "welcome";

}

@GetMapping("/recomandari")
public String recomandari(HttpSession session, Model model) {

    String email = (String) session.getAttribute("email");
    if (email == null) {
        return "redirect:/login";
    }

    Map<String, Object> client = clientService.getClientByEmail(email);
    model.addAttribute("client", client);

    Integer clientId = (Integer) client.get(key: "id_client");
    if (clientId != null) {
        model.addAttribute("recommendedShows", showService.getRecommendedShows(clientId));
    }

    return "recommended-shows";
}

@GetMapping("/logout")

public String logout() {

    return "redirect:/login";

}

@GetMapping("/bilete/{idClient}")

public String bilete(@PathVariable int idClient, Model model) {
```

```
}
You, 2 weeks ago • Incarcare
@GetMapping("/spectacole")
public String listaSpectacole(HttpSession session, Model model) {

    String email = (String) session.getAttribute("email");

    if (email == null) {
        return "redirect:/login";
    }

    Map<String, Object> client = clientService.getClientByEmail(email);
    model.addAttribute("client", client);
    model.addAttribute("shows", showService.getAllShows());

    return "shows-client";
}

@PostMapping("/cumpara-bilet")
public String cumparaBiletSubmit(@RequestParam int idClient,
                                 @RequestParam(value = "city", required = false) String city,
                                 @RequestParam(value = "idSpectacol", required = false) Integer idSpectacol,
                                 @RequestParam(value = "rand", required = false) Integer rand,
                                 @RequestParam(value = "loc", required = false) Integer loc,
                                 RedirectAttributes redirectAttributes) {


    if (city == null || city.isBlank()) {
        redirectAttributes.addFlashAttribute("errorMessage", "Te rugăm să selectezi un oraș.");
        return "redirect:/cumpara-bilet";
    }

    if (idSpectacol == null) {
        redirectAttributes.addFlashAttribute("errorMessage", "Te rugăm să selectezi un spectacol.");
        redirectAttributes.addAttribute("city", city);
        return "redirect:/cumpara-bilet";
    }

    if (rand == null) {
        redirectAttributes.addFlashAttribute("errorMessage", "Te rugăm să selectezi un rând.");
        redirectAttributes.addAttribute("city", city);
        redirectAttributes.addAttribute("showId", idSpectacol);
        return "redirect:/cumpara-bilet";
    }

    if (loc == null) {
        redirectAttributes.addFlashAttribute("errorMessage", "Te rugăm să selectezi un loc.");
        redirectAttributes.addAttribute("city", city);
        redirectAttributes.addAttribute("showId", idSpectacol);
        return "redirect:/cumpara-bilet";
    }
```

- Rute pentru admin
  - ‘GET /admin‘ - Dashboard cu statistici: număr clienți, spectacole, bilete.
  - ‘GET /admin/clients‘, ‘GET/POST /admin/clients/edit/id‘, ‘GET /admin/clients/delete/id‘ - Listare, editare și ștergere clienți.

- ‘GET /admin/shows‘, ‘GET/POST /admin/shows/add‘, ‘GET /admin/shows/delete/id‘ - Listare, adăugare și ștergere spectacole.
- ‘GET /admin/locations‘, ‘GET /admin/locations/delete/id‘ - Listare și ștergere locații.
- ‘GET /admin/artists‘, ‘GET/POST /admin/artists/edit/id‘, ‘GET /admin/artists/delete/id‘ - Listare, editare și ștergere artiști.
- ‘GET /admin/tickets‘, ‘GET /admin/tickets/delete/id‘ - Listare și ștergere bilete.
- ‘GET /admin/participations‘, ‘GET /admin/participations/delete/showId/artistId‘ - Listare și ștergere participări artiști–spectacole.

- Rute pentru rapoarte
  - ‘GET /admin/reports/popular‘ - Raport spectacole populare (folosește ‘ShowService.getPopularShows‘).
  - ‘GET /admin/reports/vip‘ - Raport clienți frecvenți (folosește ‘ClientService.getVipClients‘).
  - ‘GET /admin/reports/artists‘ - Raport performanță artiști (folosește ‘ArtistService.getArtistPerformanceReport‘).
  - ‘GET /admin/reports/client-history‘ - Istoric bilete pentru un client; validează ‘clientId‘ și afișează „ID invalid.” dacă lipsește sau este greșit.

```java
@GetMapping("/admin/reports/popular")
public String popularShowsReport(@RequestParam(name = "minTickets", defaultValue = "5") int minTickets,
                                 Model model) {
    List<Map<String, Object>> popularShows = showService.getPopularShows(minTickets);
    model.addAttribute("popularShows", popularShows);
    model.addAttribute("minTickets", minTickets);
    return "popular-shows";
}

@GetMapping("/admin/reports/vip")
public String vipClientsReport(@RequestParam(name = "minShows", defaultValue = "3") int minShows,
                               Model model) {
    List<Map<String, Object>> vipClients = clientService.getVipClients(minShows);
    model.addAttribute("vipClients", vipClients);
    model.addAttribute("minShows", minShows);
    return "vip-clients";
}

@GetMapping("/admin/reports/artists")
public String artistPerformanceReport(Model model) {
    List<Map<String, Object>> artistReport = artistService.getArtistPerformanceReport();
    model.addAttribute("artistReport", artistReport);
    return "artist-performance";
}

@GetMapping("/admin/reports/client-history")
public String clientPurchaseHistory(@RequestParam(value = "clientId", required = false) Integer clientId,
                                    Model model,
                                    RedirectAttributes redirectAttributes) {

    if (clientId == null || clientId <= 0) {
        redirectAttributes.addFlashAttribute("errorMessage", "ID invalid.");
        return "redirect:/admin";
    }

    List<Map<String, Object>> history = clientService.getClientPurchaseHistory(clientId);
    model.addAttribute("history", history);
    model.addAttribute("clientId", clientId);
    return "client-purchase-history";
}
```

3.2 Layer de servicii

ClientService conține logica de acces la date pentru:

- autentificare client (‘autentificare‘);
- creare, actualizare, ștergere client;
- obținerea biletelor unui client;
- interogări pentru rapoarte (clienți VIP, istoric client, număr total de utilizatori, spectacole, bilete).

Folosește ‘JdbcTemplate‘ pentru SQL.

```java
private JdbcTemplate jdbc;

public Map<String, Object> autentificare(String email, String parola) {

    String sql = "SELECT * FROM Clienti WHERE email = ? AND parola = ?";

    try {

        return jdbc.queryForMap(sql, email, parola);

    } catch (Exception e) {

        return null;

    }
}

public Iterable<Map<String, Object>> getBileteClient(int idClient) {

    String sql = """
            SELECT B.id_bilet, S.titlu, L.nume_locatie, B.rand, B.loc, B.data_achizitie, S.pret_bilet, S.data_spectacol
            FROM Bilete B
            JOIN Spectacole S ON B.id_spectacol = S.id_spectacol
            JOIN Locatii L ON S.id_locatie = L.id_locatie
            WHERE B.id_client = ?
            """;

    return jdbc.queryForList(sql, idClient);
}

public List<Map<String, Object>> getVipClients(int minShows) {

    String sql = """
            SELECT c.id_client,
                   c.nume,
                   c.prenume,
                   c.email,
                   COUNT(DISTINCT b.id_spectacol) AS shows_attended,
                   COALESCE(SUM(s.pret_bilet), 0) AS total_spent
            FROM Clienti c
            JOIN Bilete b ON c.id_client = b.id_client
            JOIN Spectacole s ON b.id_spectacol = s.id_spectacol
            WHERE s.data_spectacol > DATEADD(YEAR, -1, CAST(GETDATE() AS date))
            AND s.data_spectacol <= CAST(GETDATE() AS date)
            GROUP BY c.id_client, c.nume, c.prenume, c.email
            HAVING COUNT(DISTINCT b.id_spectacol) >= ?
            ORDER BY total_spent DESC
            """;

    return jdbc.queryForList(sql, minShows);
}
```

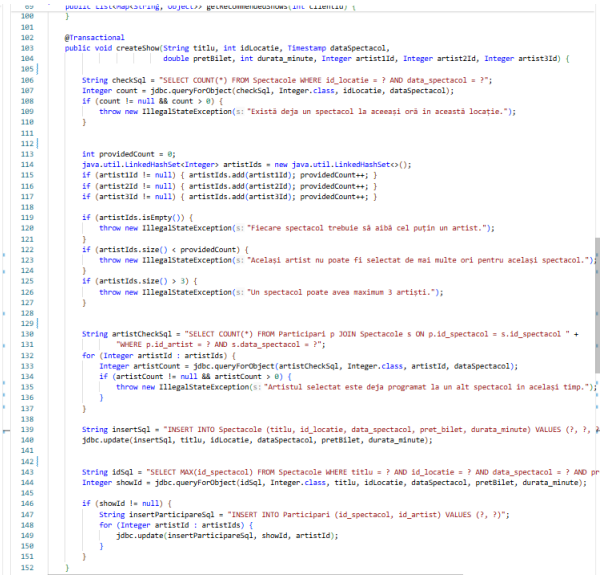
```java
public Iterable<Map<String, Object>> getAllTickets() {
            JOIN Locatii L ON S.id_locatie = L.id_locatie
            """;

    return jdbc.queryForList(sql);
}

public Map<String, Object> getClientByIdSingle(int idClient) {
    String sql = "SELECT * FROM Clienti WHERE id_client = ?";
    return jdbc.queryForMap(sql, idClient);
}

@Transactional
public void deleteClient(int id) {
    String deleteTicketsSql = "DELETE FROM Bilete WHERE id_client = ?";
    jdbc.update(deleteTicketsSql, id);

    String deleteClientSql = "DELETE FROM Clienti WHERE id_client = ?";
    jdbc.update(deleteClientSql, id);
}

@Transactional
public void deleteTicket(int idBilet) {
    String sql = "DELETE FROM Bilete WHERE id_bilet = ?";
    jdbc.update(sql, idBilet);
}

@Transactional
public void createClient(String nume, String prenume, String email, String telefon, String parola) {
    String sql = "INSERT INTO Clienti (nume, prenume, email, telefon, parola) VALUES (?, ?, ?, ?, ?)";
    jdbc.update(sql, nume, prenume, email, telefon, parola);
}

@Transactional
public void updateClient(int idClient, String nume, String prenume, String email, String telefon, String parola) {
    String sql = "UPDATE Clienti SET nume = ?, prenume = ?, email = ?, telefon = ?, parola = ? WHERE id_client = ?";
    jdbc.update(sql, nume, prenume, email, telefon, parola, idClient);
}

@Transactional
public void createTicket(int idClient, int idSpectacol, int rand, int loc) {
    String checkSql = "SELECT COUNT(*) FROM Bilete WHERE id_spectacol = ? AND rand = ? AND loc = ?";
    Integer count = jdbc.queryForObject(checkSql, Integer.class, idSpectacol, rand, loc);

    if (count != null && count > 0) {
        throw new IllegalStateException(s: "Locul selectat este deja ocupat pentru acest spectacol.");
    }

    String insertSql = "INSERT INTO Bilete (id_client, id_spectacol, rand, loc, data_achizitie) VALUES (?, ?, ?, ?, CURRENT_
    jdbc.update(insertSql, idClient, idSpectacol, rand, loc);
}

}
```

ShowService se ocupa de spectacole:

- ‘getAllShows‘ – returnează toate spectacolele, inclusiv cu status (Urmează / S-a desfășurat).

- ‘getUpcomingShows‘ – doar spectacolele viitoare, pentru cumpărare bilete.
- ‘getPopularShows‘ – raport pentru spectacole populare.
- ‘getRecommendedShows(clientId)‘ – recomandări de spectacole pe baza istoricului unui client.
- ‘createShow‘ – creează un spectacol nou în baza de date.
- ‘deleteShow‘ – șterge un spectacol.

```
public Iterable<Map<String, Object>> getAllShows() {
    String sql = """
            SELECT s.*, l.*,
                   CASE
                       WHEN s.data_spectacol > CURRENT_TIMESTAMP THEN 'Urmează'
                       ELSE 'S-a desfășurat'
                   END AS status,
                   (l.capacitate - (
                       SELECT COUNT(*) FROM Bilete b
                       WHERE b.id_spectacol = s.id_spectacol
                   )) AS remaining_seats
            FROM Spectacole s
            JOIN Locatii l ON s.id_locatie = l.id_locatie
            """;
    return jdbc.queryForList(sql);
}

public Iterable<Map<String, Object>> getUpcomingShows() {
    String sql = """
            SELECT s.*, l.*,
                   (l.capacitate - (
                       SELECT COUNT(*) FROM Bilete b
                       WHERE b.id_spectacol = s.id_spectacol
                   )) AS remaining_seats
            FROM Spectacole s
            JOIN Locatii l ON s.id_locatie = l.id_locatie
            WHERE s.data_spectacol > CURRENT_TIMESTAMP
            ORDER BY s.data_spectacol ASC
            """;
    return jdbc.queryForList(sql);
}

public List<Map<String, Object>> getPopularShows(int minTickets) {
    String sql = """
            SELECT s.id_spectacol,
                   s.titlu,
                   l.nume_locatie,
                   l.oras,
                   COUNT(b.id_bilet) AS ticket_count,
                   COALESCE(SUM(s.pret_bilet), 0) AS total_revenue
            FROM Spectacole s
            LEFT JOIN Bilete b ON s.id_spectacol = b.id_spectacol
            JOIN Locatii l ON s.id_locatie = l.id_locatie
            WHERE s.data_spectacol > CURRENT_TIMESTAMP
            GROUP BY s.id_spectacol, s.titlu, l.nume_locatie, l.oras
            HAVING COUNT(b.id_bilet) >= ?
            ORDER BY ticket_count DESC, total_revenue DESC
            """;
    return jdbc.queryForList(sql, minTickets);
}

public List<Map<String, Object>> getRecommendedShows(int clientId) {
    String sql = """
```



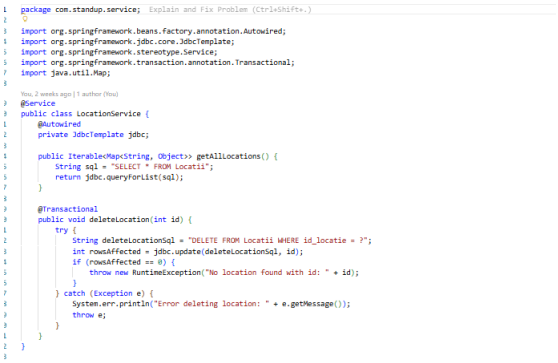
```
    }

    @Transactional
    public void createShow(String titlu, int idLocatie, Timestamp dataSpectacol,
                           double pretBilet, int durata_minute, Integer artist1Id, Integer artist2Id, Integer artist3Id) {

        String checkSql = "SELECT COUNT(*) FROM Spectacole WHERE id_locatie = ? AND data_spectacol = ?";
        Integer count = jdbc.queryForObject(checkSql, Integer.class, idLocatie, dataSpectacol);
        if (count != null && count > 0) {
            throw new IllegalStateException(s: "Există deja un spectacol la aceeași oră în această locație.");
        }


        int providedCount = 0;
        java.util.LinkedHashSet<Integer> artistIds = new java.util.LinkedHashSet<>();
        if (artist1Id != null) { artistIds.add(artist1Id); providedCount++; }
        if (artist2Id != null) { artistIds.add(artist2Id); providedCount++; }
        if (artist3Id != null) { artistIds.add(artist3Id); providedCount++; }

        if (artistIds.isEmpty()) {
            throw new IllegalStateException(s: "Fiecare spectacol trebuie să aibă cel puțin un artist.");
        }
        if (artistIds.size() < providedCount) {
            throw new IllegalStateException(s: "Același artist nu poate fi selectat de mai multe ori pentru același spectacol.");
        }
        if (artistIds.size() > 3) {
            throw new IllegalStateException(s: "Un spectacol poate avea maximum 3 artiști.");
        }


        String artistCheckSql = "SELECT COUNT(*) FROM Participari p JOIN Spectacole s ON p.id_spectacol = s.id_spectacol " +
                "WHERE p.id_artist = ? AND s.data_spectacol = ?";
        for (Integer artistId : artistIds) {
            Integer artistCount = jdbc.queryForObject(artistCheckSql, Integer.class, artistId, dataSpectacol);
            if (artistCount != null && artistCount > 0) {
                throw new IllegalStateException(s: "Artistul selectat este deja programat la un alt spectacol în același timp.");
            }
        }

        String insertSql = "INSERT INTO Spectacole (titlu, id_locatie, data_spectacol, pret_bilet, durata_minute) VALUES (?, ?, #
        jdbc.update(insertSql, titlu, idLocatie, dataSpectacol, pretBilet, durata_minute);


        String idSql = "SELECT MAX(id_spectacol) FROM Spectacole WHERE titlu = ? AND id_locatie = ? AND data_spectacol = ? AND pr
        Integer showId = jdbc.queryForObject(idSql, Integer.class, titlu, idLocatie, dataSpectacol, pretBilet, durata_minute);

        if (showId != null) {
            String insertParticipareSql = "INSERT INTO Participari (id_spectacol, id_artist) VALUES (?, ?)";
            for (Integer artistId : artistIds) {
                jdbc.update(insertParticipareSql, showId, artistId);
            }
        }
    }
```

ArtistService gestionează artiștii și participările:

- ‘getAllArtists‘, ‘getArtistById‘, ‘updateArtist‘, ‘deleteArtist‘;
- ‘getAllParticipations‘, ‘deleteParticipation‘;
- ‘getArtistPerformanceReport‘ – raport cu numărul de spectacole per artist.

LocationService lucreaza cu locatiile

- ‘getAllLocations‘, ‘deleteLocation‘.



```
package com.standup.service;  Explain and Fix Problem (Ctrl+Shift+.)

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.Map;

You, 2 weeks ago | 1 author (You)
@Service
public class LocationService {
    @Autowired
    private JdbcTemplate jdbc;

    public Iterable<Map<String, Object>> getAllLocations() {
        String sql = "SELECT * FROM Locatii";
        return jdbc.queryForList(sql);
    }

    @Transactional
    public void deleteLocation(int id) {
        try {
            String deleteLocationSql = "DELETE FROM Locatii WHERE id_locatie = ?";
            int rowsAffected = jdbc.update(deleteLocationSql, id);
            if (rowsAffected == 0) {
                throw new RuntimeException("No location found with id: " + id);
            }
        } catch (Exception e) {
            System.err.println("Error deleting location: " + e.getMessage());
            throw e;
        }
    }
}
```

4 Entități principale

- **Client:** id, nume, prenume, email, telefon, parolă.
- **Artist:** id, nume, prenume, email, telefon, data nașterii.
- **Spectacol:** id, titlu, locație, dată, preț, durată, status.
- **Locație:** id, nume, oraș.
- **Bilet:** id, client, spectacol, rând, loc, data achiziției.

5 Interfața grafică

5.1 Interfața client

5.1.1 Login('login.html')

- Form simplu cu email și parolă.
- Mesaj de eroare afișat sub formular în caz de date greșite sau câmpuri lipsă.

5.1.2 Inregistrare('signup.hml')

- Formular cu nume, prenume, email, telefon, parolă.
- Mesaje de eroare clare pentru: câmpuri obligatorii, email invalid, telefon invalid, parolă prea scurtă, email deja folosit.

5.1.3 Pagina principală client ('welcome.html')

- Meniu cu link-uri spre: spectacole, recomandări, cumpărare bilete, biletele mele.

5.1.4 Lista de spectacole ('shows-client.html')

- Tabel cu: titlu, locație, oraș, dată, status (Urmează / S-a desfășurat).
- Dropdown pentru oraș; JavaScript ascunde spectacolele din alte orașe.

5.1.5 Cumpărare bilete ('cumpara-bilet.html')

- Form cu: select oraș, select spectacol (filtrat după oraș), select rând și loc.
- Mesaje de eroare: „Te rugăm să selectezi un oraș.", „Te rugăm să selectezi un spectacol.", „Te rugăm să selectezi un rând.", „Te rugăm să selectezi un loc."

- Mesaj de succes la achiziția reușită.

5.1.6 Lista de bilete (‘bilete.html‘)

- Tabel cu biletele clientului și detalii despre spectacole și locuri.

5.2 Interfața administrator

5.2.1 Admin Dashboard (‘admin.html‘)

- - Meniu lateral cu acces la: clienți, spectacole, locații, artiști, bilete, participări, rapoarte.
- Carduri statistice: total clienți, total spectacole, total bilete.
- Carduri rapoarte: spectacole populare, clienți frecvenți, performanță artiști, istoric client (form pentru ‘clientId‘, cu mesaj „ID invalid.”).

5.2.2 Gestionare clienți (‘clients.html‘, ‘client-edit.html‘)

- Listă de clienți cu opțiuni de editare și ștergere (cu confirmare).
- Pagină de editare cu form complet și mesaje de eroare/succes.

5.2.3 Gestionare artiști (‘artists.html‘, ‘artist-edit.html‘, ‘participations.html‘)

- Listă artiști, butoane pentru editare/ștergere.
- Pagină de editare cu form pentru nume, prenume, email, telefon, data nașterii + mesaje de validare.
- Pagină pentru participări (asocieri artist–spectacol).

5.2.4 Gestionare spectacole (‘shows.html‘, ‘show-add.html‘)

- Listă spectacole cu opțiuni de ștergere.
- Form de adăugare spectacol: titlu, locație (select), dată/oră (datetime-local), preț bilet, durată (minute), select pentru artiști (primul obligatoriu logic, 2 și 3 opționali).
- Mesaje de eroare explicite pentru fiecare câmp obligatoriu.

5.2.5 Rapoarte (‘popular-shows.html‘, ‘vip-clients.html‘, ‘artist-performance.html‘, ‘client-purchase-history.html‘)

- Tabele cu date agregate pentru decizii de business (spectacole cu multe bilete vândute, clienți activi, artiști cu multe spectacole, istoricul unui client).

6 Testarea aplicației

6.1 Testare manuală

6.1.1 Formulare

- date corecte (flux de succes);
- câmpuri goale (mesaje „Toate câmpurile sunt obligatorii.” sau mesaje specifice);
- date invalide:
  - email fără ‘@‘;
  - telefon cu mai puțin de 10 cifre;
  - preț negativ;
  - durată zero sau negativă;
  - data spectacolului în trecut;
  - oraș/spectacol/rând/loc neselectate la cumpărare bilet.

6.1.2 Fluxuri funcționale

- Creare client nou + login cu acel client.
- Creare spectacol nou și verificare în: lista de spectacole pentru client, lista spectacolelor pentru admin.
- Cumpărare bilet și vizualizarea lui în pagina de bilete și în istoricul clientului (admin).
- Editare client și artist, verificând actualizarea datelor în liste.
- Ștergere entități (clienți, artiști, spectacole, locații, bilete) și confirmarea dispariției lor din liste.

6.1.3 Rapoarte

- Generarea rapoartelor după inserarea de date de test:
  - spectacole populare: verificare ordonare corectă după bilete vândute;
  - clienți frecvenți: verificare că un client cu multe bilete apare în raport;
  - performanță artiști: verificare număr spectacole per artist;
  - istoric client: verificare listă bilete după 'clientId' și comportament la 'clientId' invalid.

6.2 Testare de regresie

- După modificarea logicii de validare pe server și eliminarea atributelor HTML 'required', au fost retestate:
  - login și signup;
  - editarea clienților și artiștilor;
  - adăugarea spectacolelor;
  - cumpărarea de bilete.
- Scopul a fost verificarea:
  - corectitudinii mesajelor de eroare;
  - faptului că aplicația nu mai depinde de validarea din browser;
  - faptului că fluxurile vechi continuă să funcționeze corect.

7 Idei de îmbunătățire

- Introducerea unui sistem complet de roluri și permisiuni
  - Integrarea cu Spring Security.
  - Definirea unor roluri în baza de date (admin, client, operator etc.) și control fin al accesului la rute.
- Paginare și căutare avansată
  - Paginare în listele mari (clienți, spectacole, artiști, bilete).
  - Căutare după nume, oraș, dată.
  - Filtre multiple (ex. după preț, durată, status).
- Notificări prin email.

  - Trimiterea unui email de confirmare la cumpărarea biletului.
  - Notificări înainte de spectacol (reminder cu data și ora).

- Interfață modernă.
  - Folosirea unui framework UI modern (ex. Bootstrap avansat sau un frontend SPA).
  - Tema dark/light pentru interfață.
  - Elemente vizuale pentru statusul spectacolelor și pentru rapoarte (grafice).

- Export rapoarte.
  - Exportul rapoartelor în format PDF sau Excel.
  - Posibilitatea de printare directă a rapoartelor și a biletelor.