Proiect AWJ

1. **Introducere**

Proiectul **Administrare Parc Auto** este o aplicație Java concepută pentru a facilita gestionarea eficientă a unui parc auto, oferind două perspective principale: cea a administratorului și cea a clientului. Scopul principal al aplicației este de a simplifica procesul de urmărire a și gestionare a vehiculelor, precum și de a permite interacțiunea cu potențialii clienți interesați de vehiculele disponibile.

1. **Funcționalități**

1.Modul Administrator

Ca administrator, ai acces complet la toate informațiile despre vehicule și poți:

* Vizualiza lista vehiculelor disponibile în parc și eliminarea acestora
* Adauga un vehicul nou în lista cu toate specificațiile necesare
* Modifica informațiile vehiculelor (de exemplu, marca, modelul, anul fabricației
* Gestiona expirarea documentelor (asigurare, ITP și RoVINIETA)
* Urmări și administra programările pentru revizii sau alte operațiuni specifice fiecărei mașini(mai sus)

2.Modul client

Ca utilizator client, poți:

* Vizualiza lista vehiculelor disponibile din parc, inclusiv informații detaliate precum anul de fabricație, kilometrajul și imagini.
* Utiliza un formular de contact pentru a trimite un mesaj administratorului dacă ești interesat de un vehicul specific. Mesajul va fi trimis pe mailul persoanei care administrează aplicația.

3.Conectarea la aplicație se realizează în pagina de Autentificare:

Această pagină este destinată conectării atât clienților, cât și a administratorilor. În cazul în care clientul nu are un cont deja, își poate face unul apăsând pe “Înregistrați-vă!”.

1. **Descrierea claselor**

1. AutentifController

Aceasta clasă gestionează funcționalitățile de autentificare și înregistrare pentru utilizatori.

* Metode:
  + PaginaAutentif(): Redirecționează utilizatorul la pagina de autentificare.
  + PaginaInreg(): Redirecționează utilizatorul la pagina de înregistrare.
  + viewClientPage(): Redirecționează utilizatorul la pagina principală a clientului.
  + authenticate(): Validează credențialele utilizatorului și determină tipul de utilizator („user” sau „client”).
  + inregistrare(): Permite utilizatorului să creeze un cont nou, verificând validitatea datelor introduse.

2. ContactController

Aceasta clasă se ocupă de procesarea mesajelor trimise prin intermediul formularului de contact.

* Metode:
  + showContactForm(): Afișează formularul de contact.
  + submitContactForm(): Procesează mesajele trimise de utilizatori și le trimite prin e-mail către o adresă prestabilită.

3. HomeController

Aceasta clasă gestionează pagina principală a aplicației.

* Metode:
  + home(): Redirecționează utilizatorul la pagina principală.

4. MasinacController

Aceasta clasă oferă funcționalități de vizualizare a mașinilor disponibile pentru utilizatorii de tip client.

* Metode:
  + viewMasini(): Afișează lista de mașini disponibile, ordonată descrescător după ID.

5. MasinaController

Aceasta clasă este utilizată pentru gestionarea mașinilor în cadrul aplicației.

* Metode:
  + getMasini(): Afișează lista tuturor mașinilor existente.
  + creeazaMasina(): Permite crearea unei mașini noi și afișarea formularului de adăugare.
  + programeazaRevizie(): Permite utilizatorului să programeze o revizie tehnică pentru o mașină, validând data.
  + renewITP(), renewRCA(), renewRoVINIETA(): Actualizează data de expirare pentru ITP, RCA sau rovignetă.
  + editMasina(): Permite modificarea informațiilor despre o mașină existentă.
  + deleteMasina(): Permite ștergerea unei mașini din baza de date.

6. Client

Aceasta clasă reprezintă un client al aplicației. Este mapată la tabela "clienti" din baza de date.

* Atribute:
  + id (int): Identificator unic al clientului.
  + email (String): Adresa de email a clientului.
  + numeUser (String): Numele de utilizator al clientului.
  + parola (String): Parola asociată clientului.
* Metode:
  + Getteri și setteri pentru fiecare atribut, folosiți pentru manipularea datelor.

7. Masina

Aceasta clasă reprezintă un vehicul înregistrat în aplicație. Este mapată la tabela "masini" din baza de date.

* Atribute:
  + id (int): Identificator unic al mașinii.
  + marca (String): Marca mașinii (ex: Toyota).
  + model (String): Modelul mașinii (ex: Corolla).
  + inmat (String): Numărul de înmatriculare, unic și obligatoriu.
  + combust (String): Tipul de combustibil (ex: Benzină).
  + kilometraj (int): Kilometri parcurși.
  + anFabricatie (int): Anul fabricației.
  + pozaUrl (String): URL-ul imaginii mașinii.
  + dataRevizie (LocalDate): Data următoarei revizii.
  + itp (LocalDate): Data expirării ITP.
  + rca (LocalDate): Data expirării asigurării RCA.
  + rovinieta (LocalDate): Data expirării rovignetei.
* Metode:
  + Getteri și setteri pentru fiecare atribut, pentru a accesa sau modifica datele vehiculului.

8. MasinaOtd

Aceasta clasă este un DTO (Data Transfer Object) utilizat pentru a valida datele vehiculului la transferul între aplicație și utilizatori.

* Atribute:
  + marca (String): Marca mașinii, obligatorie.
  + model (String): Modelul mașinii, obligatoriu.
  + inmat (String): Numărul de înmatriculare, obligatoriu.
  + combust (String): Tipul de combustibil, obligatoriu.
  + kilometraj (int): Kilometri parcurși, trebuie să fie pozitiv.
  + anFabricatie (int): Anul fabricației, trebuie să fie pozitiv.
  + pozaUrl (String): URL-ul imaginii mașinii.
  + itp (LocalDate): Data expirării ITP.
  + rca (LocalDate): Data expirării RCA.
  + rovinieta (LocalDate): Data expirării rovignetei.
* Metode:
  + Getteri și setteri pentru toate atributele.
  + Constrângeri de validare (ex: @NotEmpty, @Positive) pentru asigurarea calității datelor.

9. Utilizator

Aceasta clasă reprezintă un utilizator al sistemului, care poate avea acces la funcționalitățile aplicației. Este mapată la tabela "utilizatori" din baza de date.

* Atribute:
  + id (Long): Identificator unic al utilizatorului.
  + numeUser (String): Numele de utilizator, unic și obligatoriu.
  + email (String): Adresa de email, unică și obligatorie.
  + parola (String): Parola asociată utilizatorului.
* Metode:
  + Getteri și setteri pentru fiecare atribut.

Descrierea claselor de repozitorii

10. ClientRepozitoriu

Este o interfață care extinde JpaRepository și oferă operații CRUD pentru entitatea Client.

* Metode:
* Optional<Client> findBynumeUser(String numeUser) – Găsește un client după numele utilizatorului.
* Optional<Client> findByEmail(String email) – Găsește un client după adresa de email.

11. MasinaRepozitoriu

Este o interfață care extinde JpaRepository pentru entitatea Masina.

* Metode:
  + Masina findByInmat(String inmat) – Găsește o mașină după numărul unic de înmatriculare.

12. UtilizatorRepozitoriu

Este o interfață care extinde JpaRepository pentru entitatea Utilizator.

* Metode:
  + Optional<Utilizator> findBynumeUser(String numeUser) – Găsește un utilizator după numele utilizatorului.
  + Optional<Utilizator> findByEmail(String email) – Găsește un utilizator după adresa de email.

Descrierea claselor de servicii

13. AutentifService

Este un serviciu care gestionează logica autentificării pentru utilizatori și clienți.

* Metode
  + String authenticate(String identifier, String password) – Verifică identificatorul și parola. Returnează "user" pentru utilizator, "client" pentru client sau "error" dacă autentificarea eșuează.

14. ClientService

Este un serviciu dedicat gestionării operațiunilor legate de clienți.

* Metode
  + void inregClient(Client client) – Salvează un client nou folosind ClientRepozitoriu.

Descrierea clasei principale

15.ParcApplication

Este clasa principală a aplicației, responsabilă pentru pornirea aplicației Spring Boot.

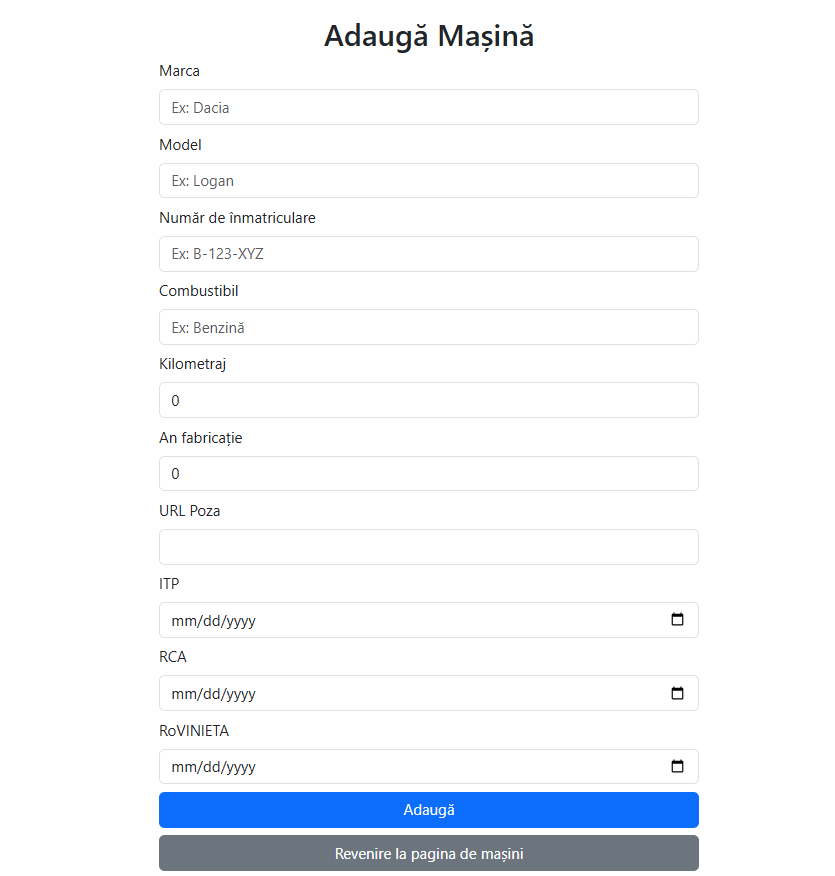
* Metode
  + public static void main(String[] args) – Punctul de intrare în aplicație, apelează metoda run din SpringApplication pentru a inițializa contextul și componentele aplicației.

1. **Descrierea elementelor funcționale oferite prin interfața grafică**

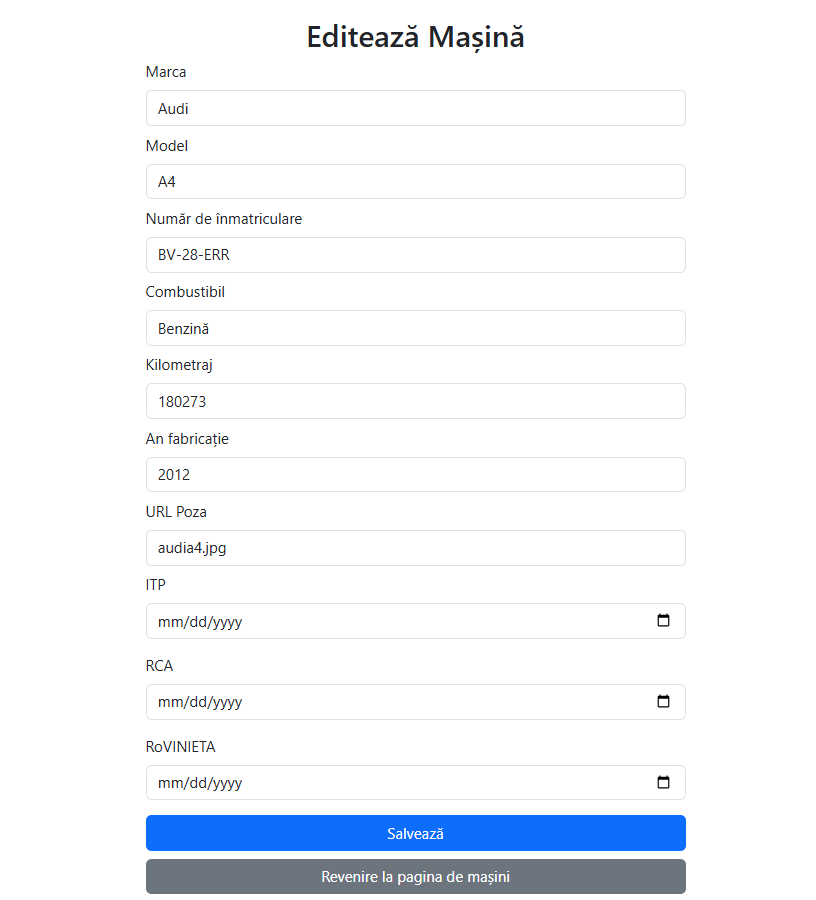
**1.Tabel cu lista mașinilor din parc**  
Interfața oferă un tabel detaliat care afișează toate vehiculele disponibile în parc. Fiecare înregistrare include informații precum marca, modelul, numărul de înmatriculare, anul fabricației și kilometrajul. Administratorul poate vizualiza rapid starea parcului auto.



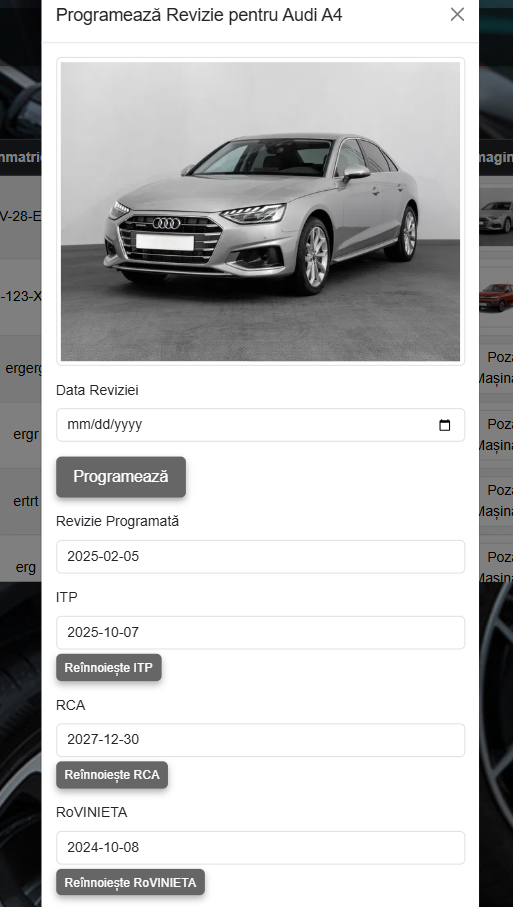
**2.Formular pentru adăugarea unei mașini**  
Permite administratorului să introducă o nouă mașină în parc. Formularul solicită detalii precum marca, modelul, tipul combustibilului, anul fabricației, kilometrajul și alte caracteristici relevante, inclusiv imagini.

****

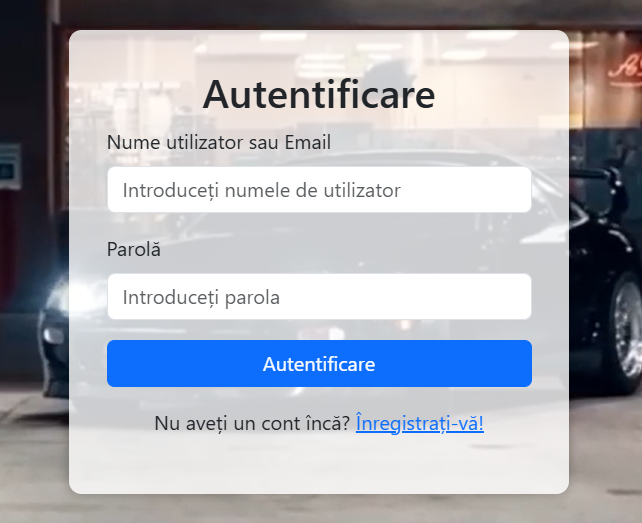
**3.Formular pentru editarea unei mașini existente**  
Administratorul poate modifica datele unei mașini deja înregistrate. Formularul oferă flexibilitate în actualizarea informațiilor precum kilometrajul, starea documentelor sau alte detalii tehnice.

****

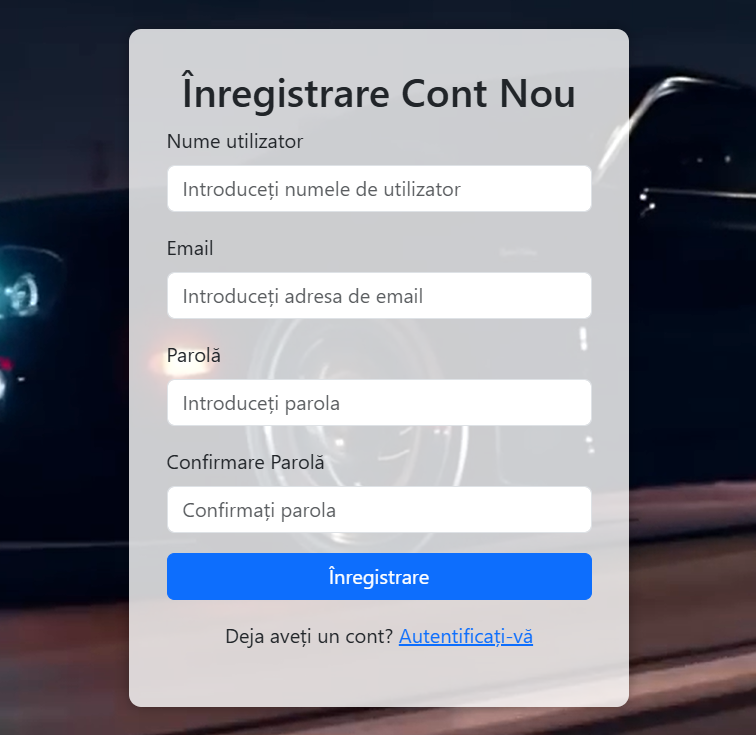
**4.Fereastră pentru vizualizarea programărilor și a datelor de expirare**  
Această interfață afișează următoarea programare de revizie pentru fiecare mașină, împreună cu detalii despre expirarea documentelor esențiale, precum ITP-ul, RCA-ul și RoVINIETA.

****

**5.Formular de autentificare**  
Pagina de autentificare permite accesul atât pentru administratori, cât și pentru clienți. Utilizatorii pot introduce un nume de utilizator sau un e-mail și o parolă pentru a accesa aplicația în funcție de rolul lor.

****

**6.Formular de înregistrare**  
Această funcționalitate este dedicată clienților care nu au deja un cont. Formularul permite completarea datelor necesare pentru a crea un cont nou, inclusiv numele de utilizator, e-mailul și parola.

****

**7.Formular de contact**  
Utilizat de clienții care doresc să trimită mesaje administratorului aplicației. Formularul include o secțiune pentru introducerea mesajului, care va fi trimis către e-mailul administratorului pentru comunicare rapidă și eficientă.

**A screenshot of a contact form

Description automatically generated**

1. **Aspecte privind testarea aplicației**

Testarea aplicației a fost realizată pentru a asigura funcționalitatea și corectitudinea caracteristicilor principale. În procesul de testare, s-au utilizat următoarele metode:

1. **Testarea manuală**
   * Navigarea prin interfața utilizatorului pentru a verifica funcționalitățile esențiale, precum autentificarea, adăugarea și modificarea vehiculelor, și vizualizarea listelor.
   * Testarea paginii de autentificare pentru a valida conectarea și crearea de conturi noi.
2. **Testarea validării datelor**
   * Verificarea mesajelor de eroare și a constrângerilor definite în modelul aplicației (de exemplu, câmpurile obligatorii din MasinaOtd).
   * Asigurarea că valorile incorecte sau incomplete sunt tratate corespunzător.
3. **Testarea funcționalităților backend**
   * Testarea metodelor din servicii precum ClientService și AutentifService pentru a valida operațiunile de salvare și autentificare.
   * Utilizarea manuală a repository-urilor pentru a verifica metodele personalizate, cum ar fi findByInmat sau findByEmail.
4. **Testarea comunicației cu baza de date**
   * Verificarea că toate operațiunile CRUD funcționează corespunzător pentru entitățile definite (Client, Masina, Utilizator).

Această abordare a asigurat o aplicație funcțională și robustă, cu o experiență utilizator fără probleme majore.

1. **5 idei despre cum poate fi imbunatatita aplicatia**

**1.Integrarea unui Sistem de Notificări Automate**

Implementarea unui sistem de notificări automate care să alerteze administratorii şi utilizatorii cu privire la expirarea documentelor vehiculelor (asigurare, ITP, Rovinietă) sau programările pentru revizii. Notificările pot fi trimise prin e-mail sau afişate direct în aplicaţie.

**2.Adăugarea Funcţionalităţii de Filtrare Avansată**

Oferirea de opţiuni avansate pentru filtrarea vehiculelor în funcţie de diferite criterii, cum ar fi preţul, kilometrajul, anul fabricaţiei, tipul de combustibil şi starea tehnică a documentelor. Această funcţie ar îmbunătăţi experienţa utilizatorilor, permiţându-le să găsească mai rapid vehiculul dorit.

**3.Integrarea unei Hărţi pentru Localizarea Vehiculelor**

Adăugarea unei secţiuni de localizare în care fiecare vehicul poate fi asociat cu o locaţie specifică pe o hartă (de exemplu, unde este parcat sau disponibil pentru vizionare). Aceasta ar putea fi utilă în cazul flotelor mari de vehicule.

**4.Istoric pentru Vehicule**

Implementarea unei funcţionalităţi care permite înregistrarea şi vizualizarea istoricului unui vehicul, inclusiv reparaţii efectuate, programări anterioare, revizii sau modificări în documente. Aceasta ar creşte transparenţa şi ar ajuta clienţii să ia decizii informate.

**5.Modul pentru Rapoarte Detaliate**

Dezvoltarea unui modul care permite generarea automata a rapoartelor, atât pentru administratori, cât şi pentru clienţi. Administratorii ar putea genera rapoarte despre starea parcului auto (ex. vehicule încărcate şi descarcate, documente care expiră curând), în timp ce clienţii ar putea primi rapoarte personalizate despre vehiculele de interes.