



**UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA**

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
SPECIALIZAREA CALCULATOARE**

PROIECT
LICITAȚIE DE MAȘINI
(masiniFA)

Disciplina : Ingineria Sistemelor

Realizat de studenții :

Cioban Fabian-Remus

Pojar Andrei-Gabriel

An : III

Grupa : 30233

Profesor coordonator : Burzo Adrian

CUPRINS

- 1.Introducere
- 2.Specificație Proiect
- 3.Diagrama Use Case
- 4.Implementarea propriu-zisă
- 5.Diagrame, Design Pattern
- 6.Limbaje de programare, frameworks, baze de date
- 7.Concluzii
- 8.Bibliografie
- 9.ReadMe

1.Introducere

Proiectul constă într-o aplicație web de licitație online pentru mașini second-hand numită masiniFA. Aplicația reprezintă o soluție modernă pentru tranzacțiile rapide și eficiente, ajutând vânzătorii să își vândă mașina instantaneu către compania noastră și să primească banii pe loc. Ulterior, mașina, care acum e proprietatea noastră va fi scoasă la licitație online. Scopul acestui proiect este să ofere o platformă sigură și ușor de utilizat, care să faciliteze procesul de licitație și să aducă beneficii atât cumpărătorilor cât și vânzătorilor.

2.Specificație Proiect

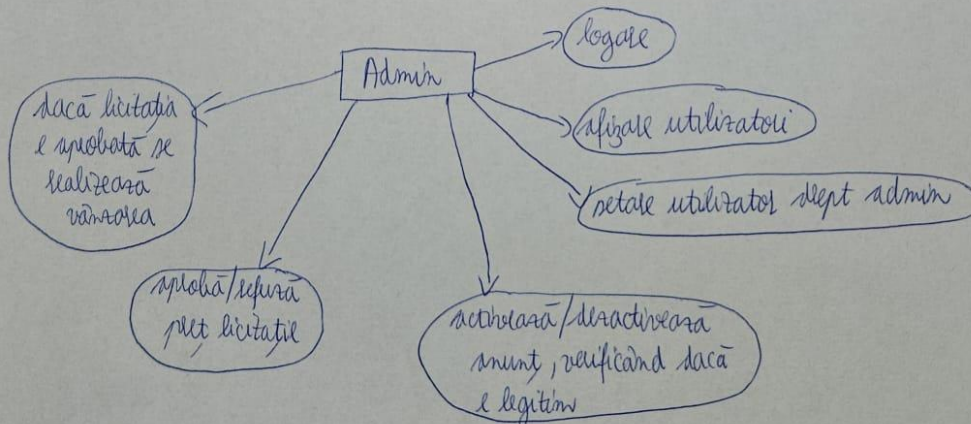
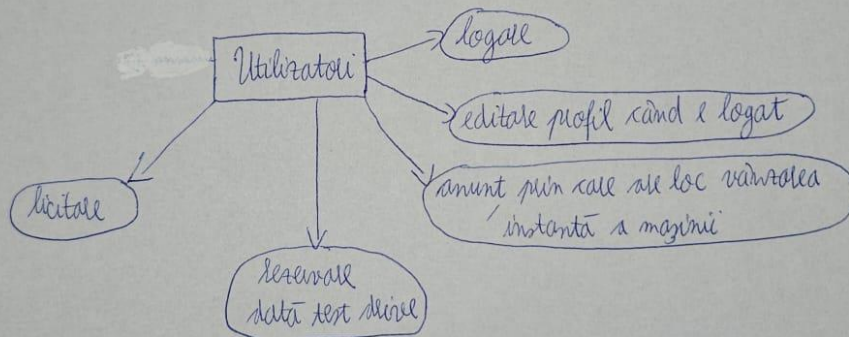
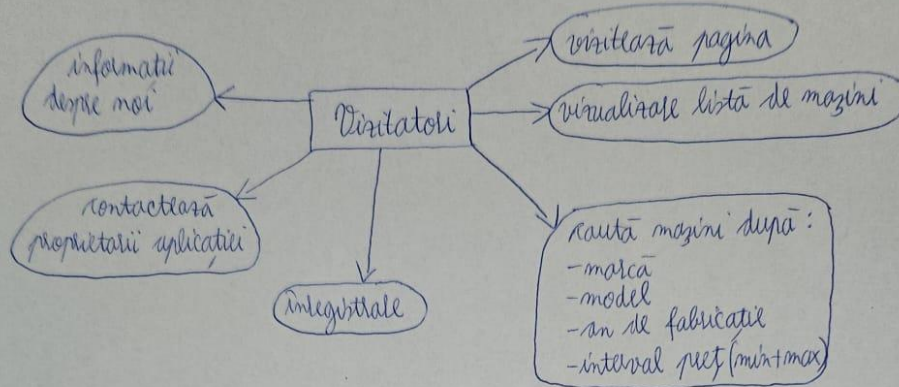
Platforma masiniFA a fost dezvoltată cu scopul de a oferi o soluție completă pentru piața auto-digitală. Principalele caracteristici ale proiectului includ:

- Tranzacții instantanee – vânzătorii pot beneficia de procesul rapid de vânzare primind banii pe loc în schimbul mașinii.
- Licitatii online – după achiziționarea mașinii, aceasta este pusă la licitație, permițând cumpărătorilor să participe la procesul de licitație
- Instrumente de decizie – cumpărătorii au acces la date detaliate și resurse digitale, precum caracteristicile mașinii, pentru a putea lua decizii de cumpărare

3.Diagrama Use Case

Diagrama Use Case evidențiază interacțiunile dintre utilizatori și sistem, ilustrând scenariile cheie.

DIAGRAMA DE USE-CASE



4. Implementarea propriu-zisă

Implementarea platformei masiniFA a implicat utilizarea unui set divers de tehnologii și framework-uri pentru a asigura o funcționalitate robustă și o experiență utilizator de calitate. Iată detalii specifice legate de implementarea propriu-zisă a diferitelor aspecte ale proiectului:

Backend Java Spring Boot:

Utilizarea Java Spring Boot a furnizat un cadru solid pentru dezvoltarea backend-ului. S-au utilizat servicii Spring pentru a organiza și separa funcționalitățile logice ale aplicației. JavaServer Pages (JSP) pentru Interfața Web Dinamică:

JSP a fost folosit pentru a crea pagini web dinamice, permitând afișarea datelor în mod dinamic și interacțiunea fluidă cu utilizatorii. Implementarea JSP a inclus integrarea cu backend-ul prin utilizarea expresiilor Java pentru a afișa și manipula datele în timp real.

CSS și Bootstrap pentru Stilizare:

Stilizarea paginilor web s-a realizat prin intermediul CSS, asigurând un aspect modern și atractiv. Bootstrap a fost utilizat pentru a facilita design-ul responsiv și pentru a garanta o experiență coezivă pe diferite dispozitive.

JavaScript pentru Interactivitate:

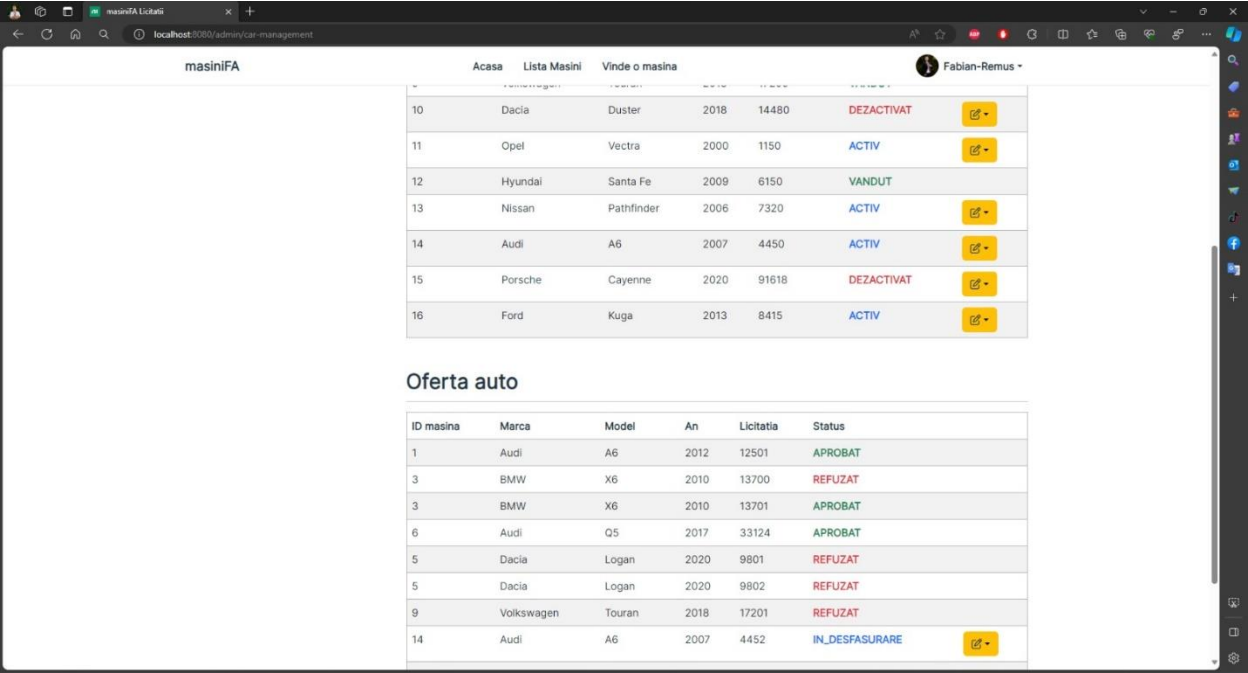
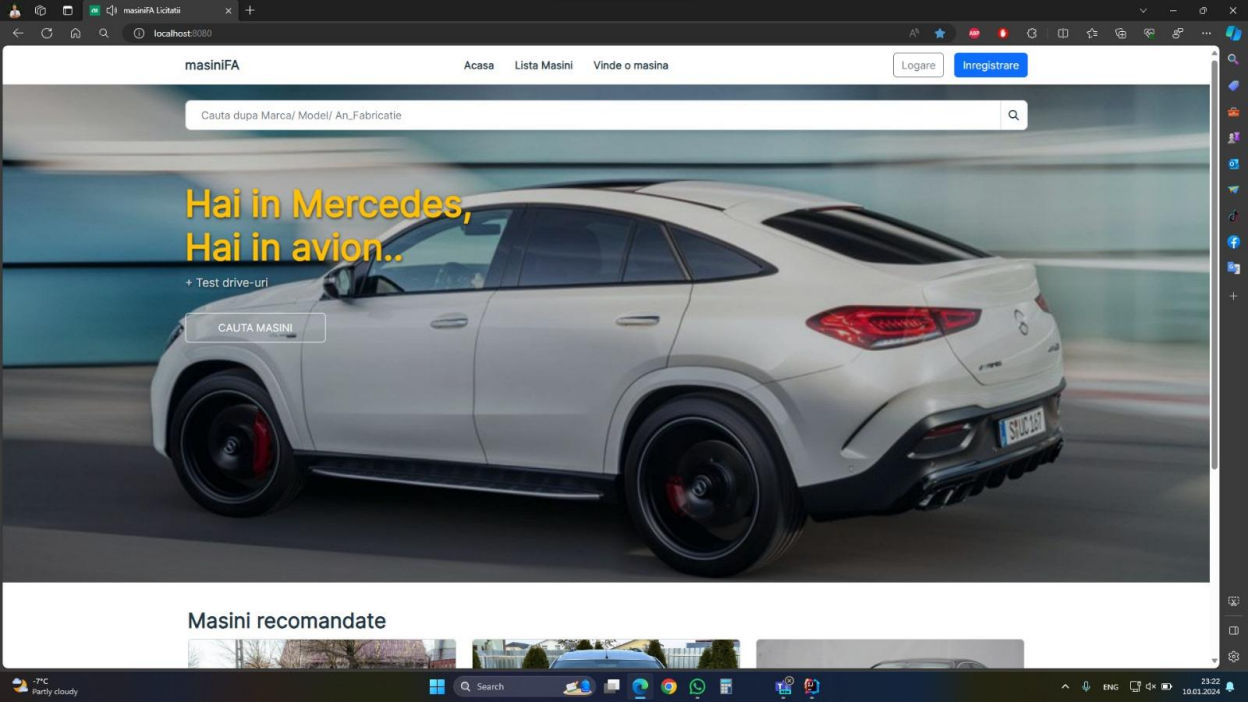
JavaScript a fost implementat pentru a adăuga interactivitate și funcționalitate pe utilizator pe partea de test drive.

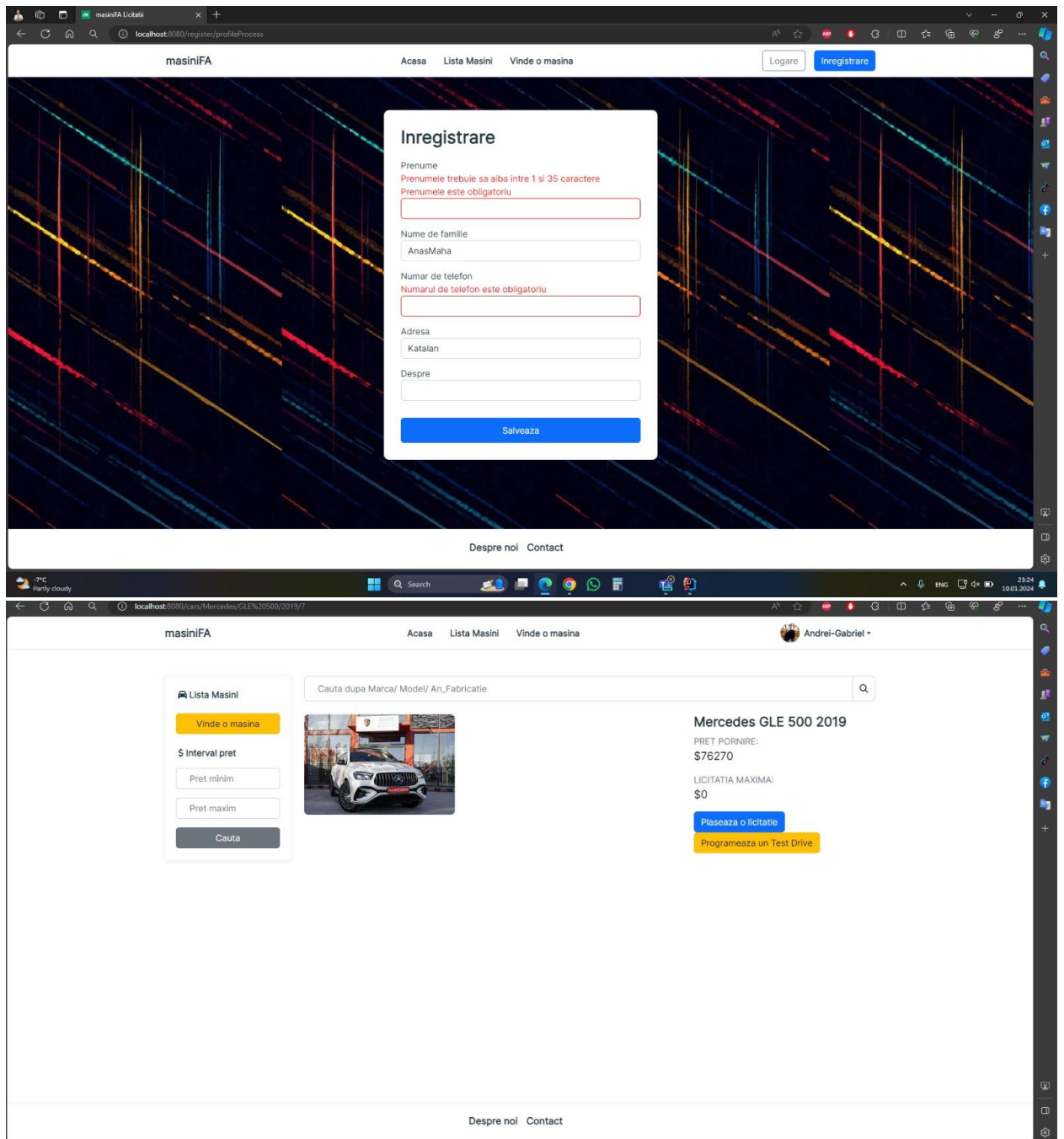
Gestionarea Bazei de Date MySQL:

MySQL a fost ales ca sistem de gestionare a bazelor de date pentru a stoca eficient informațiile legate de mașini, utilizatori și tranzacții. S-au realizat interogări pentru a extrage date în cadrul licitațiilor și tranzacțiilor.

Această abordare tehnologică și implementarea detaliată au condus la crearea unei platforme solide și eficiente, asigurând o experiență utilizator de calitate și funcționalități avansate pentru tranzacții auto rapide și licitații online transparente.

Interfața:





Acest cod reprezintă o pagină web scrisă în JavaServer Pages (JSP). Pagina include componente precum antet (header), bara de navigație (navbar), și subsol (footer). Conținutul principal al paginii este împărțit în două secțiuni: "Utilizatori" și "Admin". Fiecare secțiune afișează informații sub forma unor tabele despre utilizatori sau administratori, iar dropdown-urile oferă opțiuni de editare și transformare în administrator pentru fiecare utilizator.

Un script JavaScript monitorizează poziția de derulare a paginii și, după o întârziere de 10 secunde, reîncarcă pagina și readuce utilizatorul la poziția anterioară de derulare. În ansamblu, pagina pare să fie destinată administrării utilizatorilor și administratorilor într-un mediu de administrare.

```
MasiniFaApplication.java x dashboard.jsp
16         location.reload();
17         window.scrollTo(0, scrollPos);
18     }, 10000);
19 }
20 </script>
21 <!-- Navbar -->
22 <%@ include file="../../components/navbar.jsp" %>
23
24 <!-- Main -->
25 <main>
26     <div class="container pt-5">
27         <div class="d-flex">
28             <!-- Sidebar -->
29             <aside class="sidebar-admin pe-md-3">
30                 <ul>
31                     <li class="active-page">
32                         <a href="request.getContextPath() %>/admin"><i class="fa-solid fa-gauge-high"></i> Persoane</a>
33                     </li>
34                     <li>
35                         <a href="request.getContextPath() %>/admin/car-management"><i class="fa-solid fa-car"></i>
36                             Masini</a>
37                     </li>
38                 </ul>
39             </aside>
40
41             <!-- Content -->
42             <div class="content-wrapper">
43                 <!-- List User -->
44                 <h2 class="fw-bold mb-3">Utilizatori</h2>
45                 <div class="table-responsive-md">
46                     <table class="table table-striped">
47                         <!-- Head -->
48                         <thead>
49                             <tr>
50                                 <th>ID</th>
51                                 <th>Prenume</th>
52                                 <th>Nume de familie</th>
53                                 <th>Numar de telefon</th>
54                                 <th>Adresa</th>
55                                 <th></th>
56                             </tr>
57                         </thead>
58                     </table>
59                 </div>
60             </div>
61         </div>
62     </div>
63 </main>
64 </body>
65 </html>
```


5.Diagrame, Design Pattern

Am realizat diagrama de stări și diagrama de activități.

Diagrama de activități:

Activity Diagram

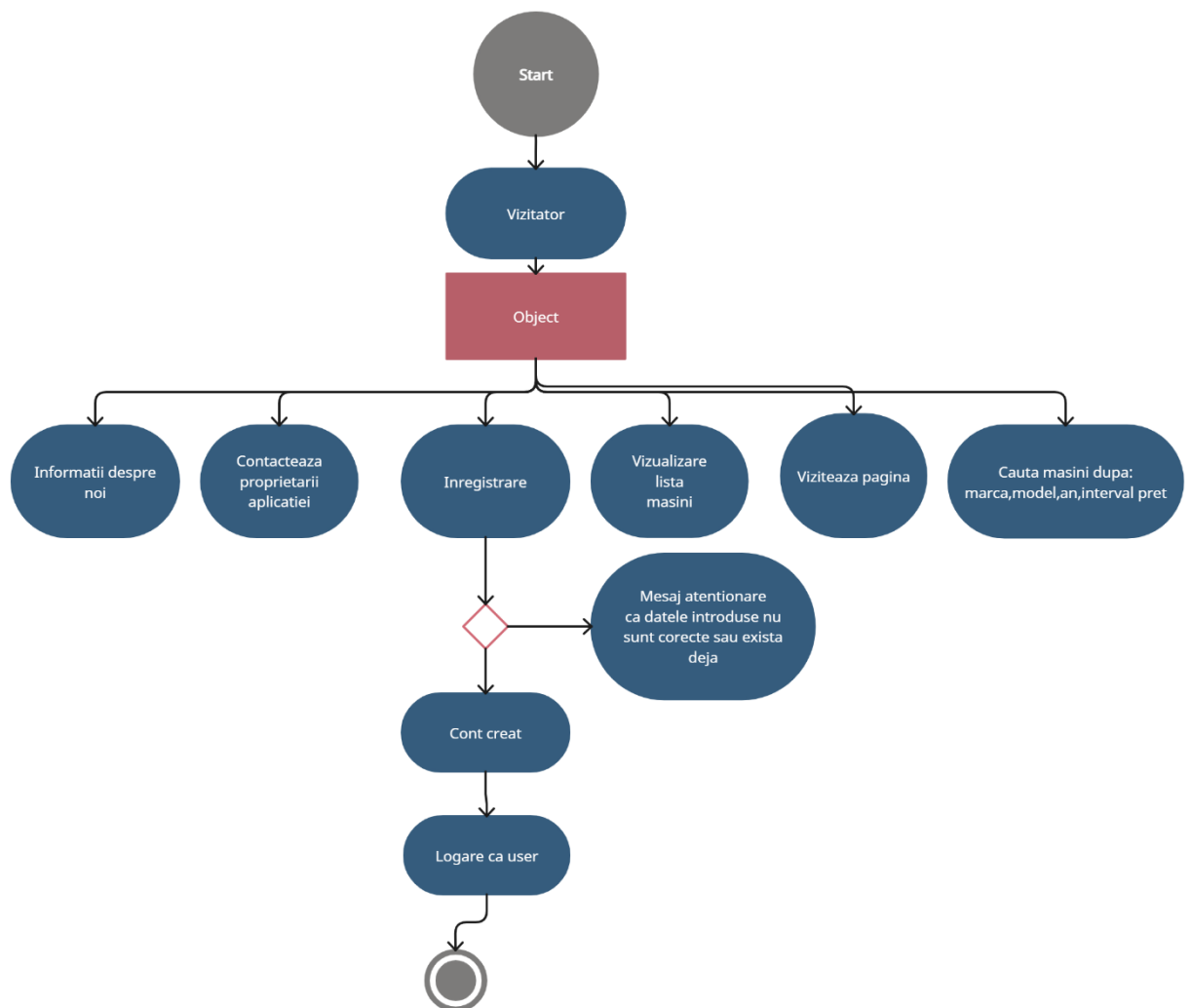
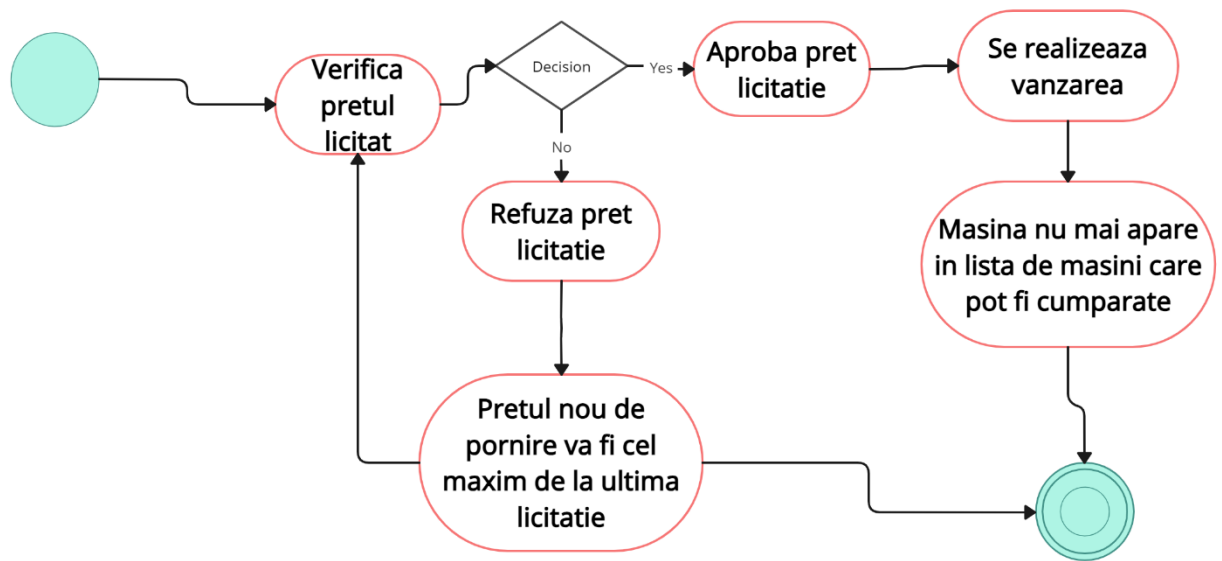


Diagrama de stări:

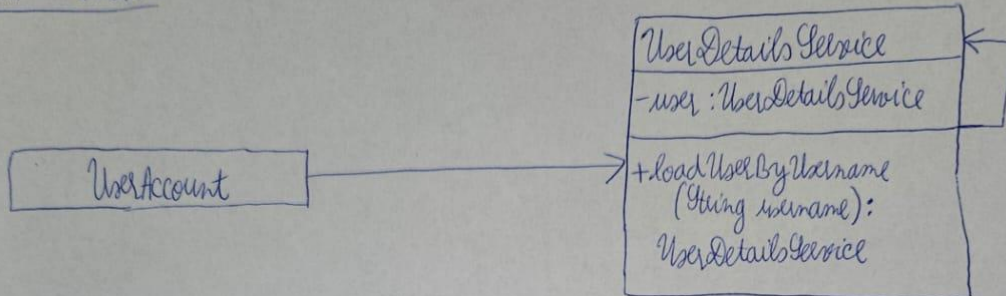
State Diagram



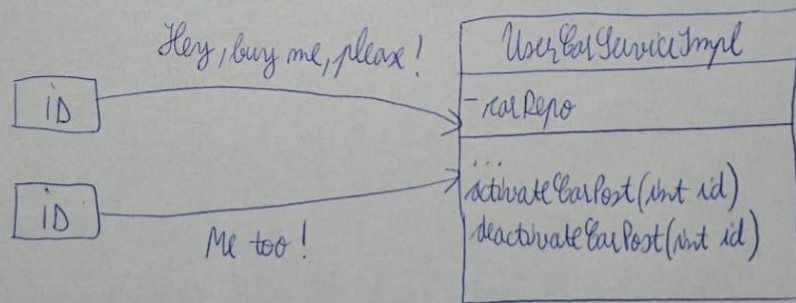
Design Pattern:

DESIGN PATTERN

SINGLETON



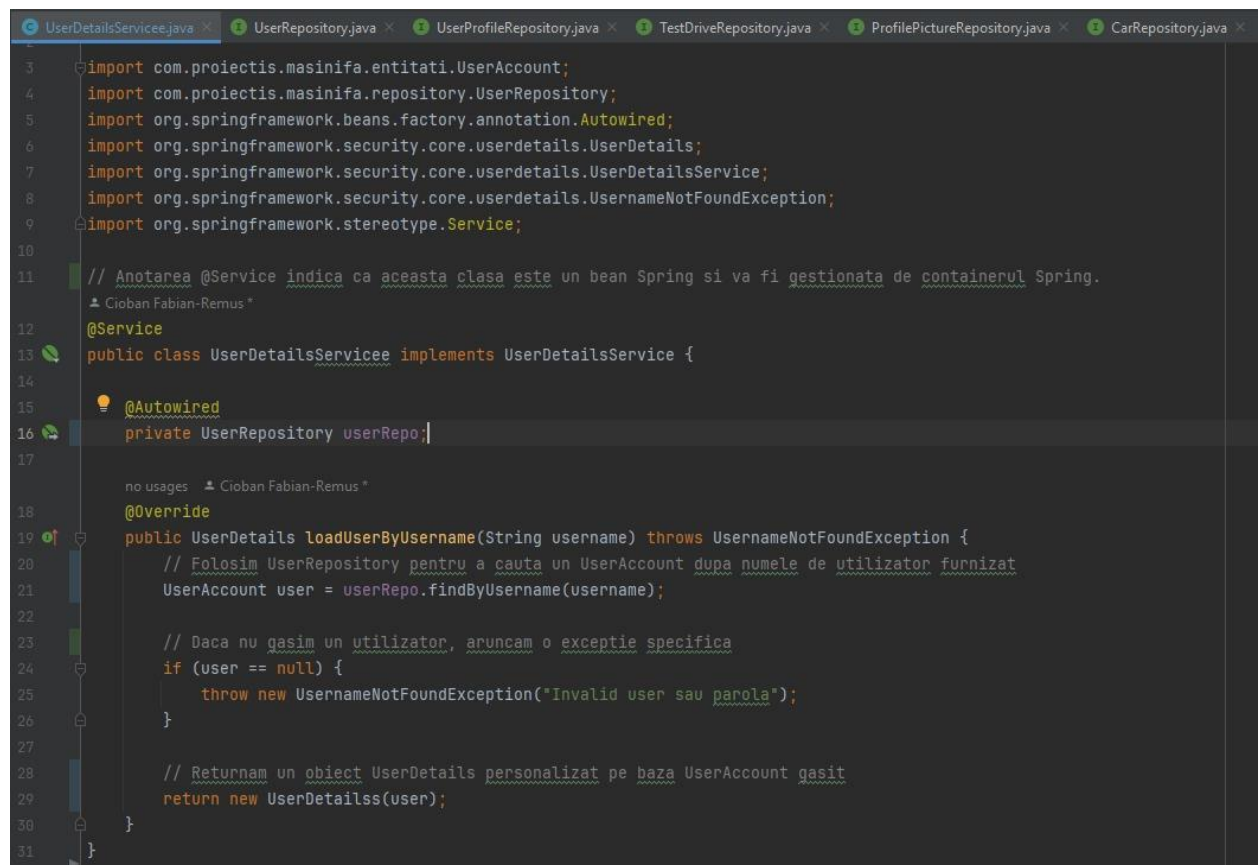
OBSERVER



Design Pattern->Singleton:

Singleton este un model de proiectare creativ care vă permite să vă asigurați că o clasă are o singură instanță, oferind în același timp un punct de acces global la această instanță.

Am folosit acest design pattern în clasa `UserDetailsService`. Problema pe care o rezolvă design pattern-ul Singleton în acest context este evitarea creării multiple a instanțelor clasei `UserDetailsService`. În cadrul unui serviciu de autentificare și autorizare, cum este în acest caz, având o singură instanță a `UserDetailsService`, se evită conflicte și incoerențe care ar putea apărea dacă ar exista mai multe instanțe ale aceleiași clase.



```
1  UserDetailsService.java x  UserRepository.java x  UserProfileRepository.java x  TestDriveRepository.java x  ProfilePictureRepository.java x  CarRepository.java x
2
3  import com.proiectis.masinifa.entitati.UserAccount;
4  import com.proiectis.masinifa.repository.UserRepository;
5  import org.springframework.beans.factory.annotation.Autowired;
6  import org.springframework.security.core.userdetails.UserDetails;
7  import org.springframework.security.core.userdetails.UserDetailsService;
8  import org.springframework.security.core.userdetails.UsernameNotFoundException;
9  import org.springframework.stereotype.Service;
10
11  // Anotarea @Service indica ca aceasta clasa este un bean Spring si va fi gestionata de containerul Spring.
12  // Cioban Fabian-Remus *
13  @Service
14  public class UserDetailsService implements UserDetailsService {
15
16      @Autowired
17      private UserRepository userRepo;
18
19      // no usages  Cioban Fabian-Remus *
20      @Override
21      public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
22          // Folosim UserRepository pentru a cauta un UserAccount dupa numele de utilizator furnizat
23          UserAccount user = userRepo.findByUsername(username);
24
25          // Daca nu gasim un utilizator, aruncam o exceptie specifica
26          if (user == null) {
27              throw new UsernameNotFoundException("Invalid user sau parola");
28          }
29
30          // Returnam un obiect UserDetails personalizat pe baza UserAccount gasit
31          return new UserDetails(user);
32      }
33  }
```

Design Pattern->Observer:

Observatorul este un model de proiectare comportamentală care vă permite să definiți un mecanism de abonare pentru a notifica mai multe obiecte cu privire la orice eveniment care se întâmplă cu obiectul pe care îl observă.

În contextul metodelor `activateCarPost` și `deactivateCarPost`, se poate spune că acestea ar putea beneficia de utilizarea design pattern-ului Observer în situații în care alte părți ale sistemului sau interfețelor de utilizator trebuie să fie notificate atunci când starea unei mașini se schimbă (în acest caz, atunci când se activează sau dezactivează o mașină).

```

    }

    2 usages  Cioban Fabian-Remus
    @Override
    public void activateCarPost(int id) {
        Car editedCar = carRepo.findById(id).get();

        // Actualizează statusul mașinii la "ACTIV"
        editedCar.setStatus("ACTIV");

        carRepo.save(editedCar);
    }

    2 usages  Cioban Fabian-Remus
    @Override
    public void deactivateCarPost(int id) {
        Car editedCar = carRepo.findById(id).get();

        // Actualizează statusul mașinii la "DEZACTIVAT"
        editedCar.setStatus("DEZACTIVAT");

        carRepo.save(editedCar);
    }

    no usages  Cioban Fabian-Remus

```

6. Limbaje de programare, frameworks, baze de date

Limbaje de programare: Java, JSP(Java Server Pages), CSS, JavaScript

Frameworks: Java Spring Boot pentru backend și Bootstrap pentru frontend

Baze de date: MySQL

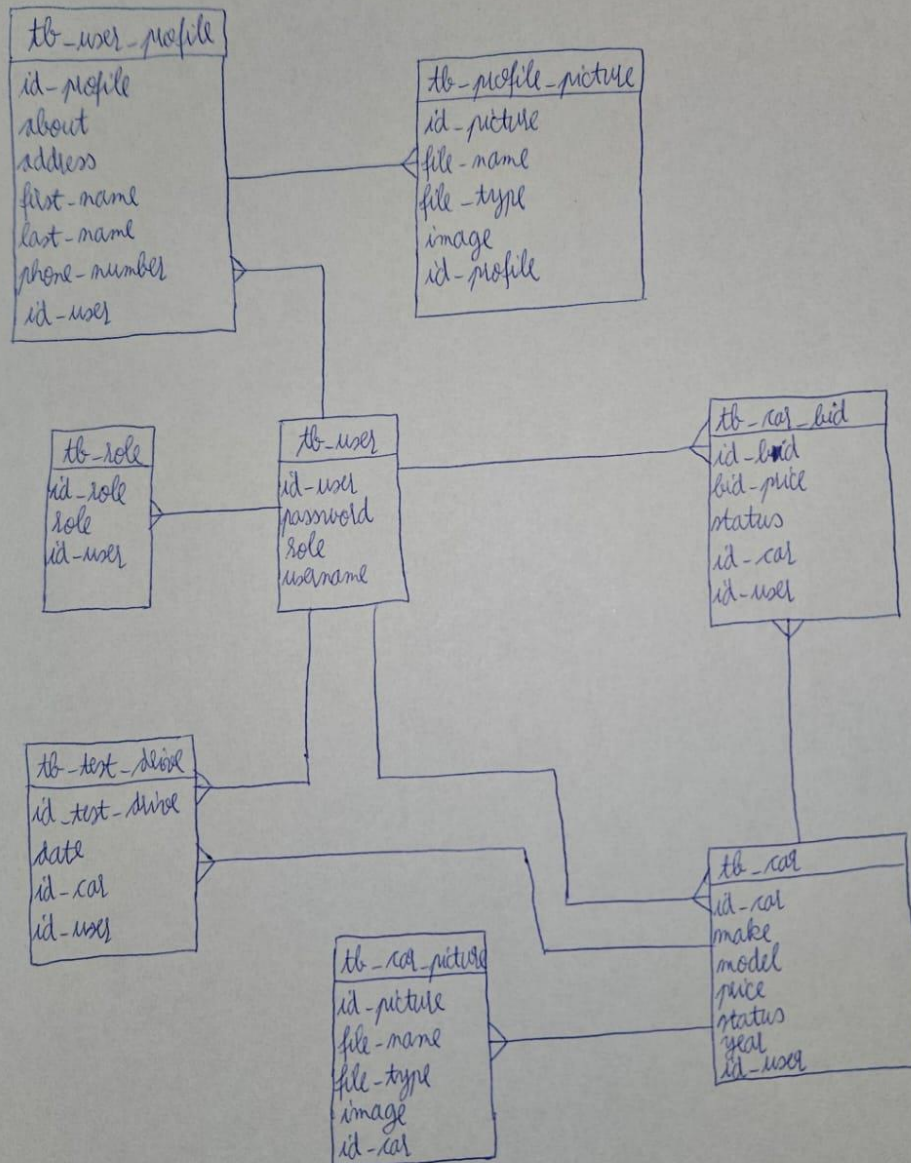
Am folosit Java pentru dezvoltarea backend-ului și logicilor, JSP pentru construirea paginilor web și interacțiunile cu ele, CSS pentru înfrumusețare și JavaScript pentru cererea de test drive.

Java Spring Boot a fost utilizat pentru dezvoltarea rapidă a aplicației web, facilitând gestionarea dependențelor și configurarea simplă a serverului. Bootstrap a fost folosit pentru a asigura un aspect modern interfeței.

MySQL a fost selectat ca sistem de gestionare a bazelor de date, oferind o soluție fiabilă și performantă pentru stocarea datelor.

Diagrama de baze de date:

DIAGRAMA DE BASE DE DATE



7. Concluzii

Acest proiect, masiniFA a reușit să creeze o platformă inovatoare care aduce beneficii semnificative atât vânzătorilor, cât și cumpărătorilor în industria auto. Mecanismul nostru de vânzare instantanee a adus un avantaj mare vânzătorilor, permițându-le să își vândă mașina și să primească banii pe loc. Implementarea licitațiilor online a creat un mediu în care cumpărătorii au acces la informații detaliate despre mașini și pot lua decizia de a licita pentru aceasta, iar în cazul în care compania noastră este de acord cu prețul acordat, acesta va intra în posesia mașinii.

Interfața dezvoltată a asigurat o experiență și o navigare pe platformă plăcută utilizatorilor. Sistemul nostru de gestionare a tranzacțiilor, susținut de o bază de date MySQL a contribuit la procesarea eficientă a datelor legate de mașini și licitații.

8. Bibliografie

- <https://spring.io/guides/gs/spring-boot/>
- <https://www3.ntu.edu.sg/home/ehchua/programming/java/JSPByExample.html>
- <https://www.freecodecamp.org/news/the-best-bootstrap-examples/>
- <https://www.vogella.com/tutorials/JavaPersistenceAPI/article.html>

9. ReadMe

Cerințe de Sistem:

- Java Development Kit (JDK)
- MySQL Workbench
- Apache Maven (opțional, dacă nu se utilizează containerul Maven încorporat)

Configurare:

- Clonare repo
- Creare bază de date

Rulare aplicație:

- Deschizi proiectul cu un IDE
- Actualizează fișierul application.properties din directorul src/main/resources cu credențialele bazei de date
- Run application
- Deschizi browserul și navighează la adresa <http://localhost:8080/>

Te bucuri de funcționalitatea aplicației