



**Academia de Studii Economice din București**  
**Facultatea de Cibernetică, Statistică și Informatică Economică**  
**Specializarea: Informatică Economică**

## **Aplicație Web pentru gestiunea unui service auto**

---

### **LUCRARE DE LICENȚĂ**

Coordonator științific:  
**Prof. Univ. Dr. Diaconița Vlad**

Absolvent:  
**Cupii Cezin**

**București**  
**2021**

# Cuprins

<b>Introducere .....</b>	<b>1</b>
<b>1. Descrierea problemei economice .....</b>	<b>3</b>
1.1 <b>Prezentarea domeniului abordat.....</b>	<b>3</b>
1.2 <b>Prezentarea activității care va fi informatizată .....</b>	<b>4</b>
1.3 <b>Analiza sistemelor existente pe piață.....</b>	<b>6</b>
<b>2. Analiza sistemului informatic.....</b>	<b>9</b>
2.1 <b>Descrierea generală a sistemului informatic .....</b>	<b>9</b>
2.2 <b>Specificarea cerințelor sistemului informatic.....</b>	<b>10</b>
2.2.1 <b>Diagrame ale cazurilor de utilizare .....</b>	<b>10</b>
2.2.2 <b>Diagrama cazului de utilizare programare service .....</b>	<b>11</b>
2.2.3 <b>Diagrama cazului de utilizare programare service .....</b>	<b>12</b>
2.3 <b>Analiza sistemului existent .....</b>	<b>13</b>
2.3.1 <b>Diagrame de activitate.....</b>	<b>13</b>
2.3.2 <b>Diagrame de clase .....</b>	<b>15</b>
2.3.3 <b>Diagrame de stare.....</b>	<b>15</b>
2.3.4 <b>Diagrame de interacțiune .....</b>	<b>16</b>
2.3.5 <b>Diagrame de procese și colaborare în BPMN .....</b>	<b>18</b>
<b>3. Proiectarea sistemului informatic.....</b>	<b>20</b>
3.1 <b>Diagrama de clase detaliată.....</b>	<b>20</b>
3.2 <b>Diagrama bazei de date .....</b>	<b>20</b>
3.3 <b>Proiectarea interfețelor utilizatorului.....</b>	<b>21</b>
3.4 <b>Harta de navigare printre interfețe .....</b>	<b>23</b>
3.5 <b>Diagrama de componente.....</b>	<b>23</b>
3.6 <b>Diagrama de desfășurare.....</b>	<b>24</b>
<b>4. Proiectarea sistemului informatic.....</b>	<b>25</b>
4.1 <b>Tehnologii informatice utilizate.....</b>	<b>25</b>
4.2 <b>Implementarea aplicației.....</b>	<b>29</b>
4.2.1 <b>Implementarea modului de <i>back-end</i> .....</b>	<b>29</b>
4.2.2 <b>Implementarea modului de <i>front-end</i> .....</b>	<b>34</b>
4.3 <b>Prezentarea aplicației .....</b>	<b>38</b>
<b>Concluzii.....</b>	<b>46</b>
<b>Bibliografie .....</b>	<b>48</b>
<b>Anexe.....</b>	<b>50</b>

## Introducere

În ultimii zece ani, cu toții am observat că tot ce ține de tehnologie a avut o evoluție exponențială, atât la nivelul utilității cât și la nivel financiar, dar mulți dintre noi ne întrebăm cum am ajuns de la unelte primitive cum ar fi sulița și săgețile la mașinării care fac treaba în locul nostru.

Din punct de vedere istoric și statistic, populația umană a avut cea mai abruptă creștere începând cu anul 1781, odată cu prima revoluție industrială, fiind reprezentată de inventarea motorului cu abur de către James Watt. Cel mai important obiectiv realizat de această invenție este acela că a ajutat populația să treacă peste bariera puterii fizice, atât umană cât și animală, reușind astfel să genereze cantități mari de energie în orice moment. [1]

Cea de-a doua revoluție industrială a fost reprezentată de darea în folosință a curentului electric din anul 1844 și de apariția telefonului în anul 1876, urmând ca în anul 1893 să fie prezentată prima demonstrație la scară mare a curentului electric, aprinzând peste 100.000 felinare.

Totuși, aceste tehnologii au avut însă o viteză de dezvoltare foarte lentă comparativ cu cea a internetului de astăzi. Primele două tehnologii au apărut acum peste 100 de ani dar totuși, nu se compară cu evoluția internetului, care are o vârstă de doar 30 de ani. Acest lucru are la baza două legi pe care se bazează, **legea lui Moore** și **legea lui Bell**. [1]

În 1965, înainte de a fonda compania Intel, una din cele mai mari companii din zilele noastre producătoare de microprocesoare, **Gordon Moore** a făcut o predicție în ceea ce privește viteza de procesare. Bazat pe un studiu al costurilor circuitelor raportat la puterea produsă de acestea realizat între anii 1962 și 1965, americanul a afirmat că puterea de procesare se dublează la aproximativ fiecare doi ani. [1]

**Gordon Bell** a afirmat, în anul 1972 cum fiecare tip de sistem de calcul se formează, evoluează și, opțional dispar, toate aceste lucruri întâmplându-se într-o perioadă de 10 ani, ele devenind la fiecare ciclu de 10 ori mai mici, mai rapide și mai ieftine.

Cel mai bun exemplu pentru toată această evoluție este chiar în mâinile noastre în cea mai mare parte a timpului. *Smartphone-urile* din ziua de astăzi sunt de sute de mii de ori mai puternice decât computerul care a i-a ghidat pe Neil Armstrong, Buzz Aldrin și Michael Collins pe lună în anul 1969, în cadrul misiunii Apollo 11. [1]

Anul 2020 a fost unul foarte greu pentru noi, din toate punctele de vedere. Am fost loviți de o pandemie neașteptată care ne-a schimbat complet viețile și ne-a făcut să ne adaptăm la un cu totul alt stil de viață. Vreau să subliniez un cuvânt din propoziția anterioară, și anume **adaptare**. Întrucât a trebuit să stăm în cea mai mare parte a timpului în casă, toate acțiunile pe care le făceam fizic, de la mersul la cumpărături, la service, la școală și facultate, până la sport, toate au fost nevoite să migreze într-un alt mediu, cel online. Astfel, mediul online a avut o creștere imensă, un exemplu foarte bun este cel de pe piața de *Ecommerce*, care a avut în USA o creștere de 44%, de la 598 de miliarde de dolari în 2019, la 861 de miliarde în 2020. [2]

Scopul lucrării mele de licență este de a veni cu o mână de ajutor către un sector folosit de aproape jumătate din populația totală a țării, și anume sectorul auto, partea de mentenanță a acestuia, cu peste 10 milioane de intrări pe an.

Pentru a putea realiza acest lucru, voi folosi tehnologii cât mai simple pentru utilizatorul de rând, dar în același timp cât mai de actualitate, disponibile pe cât mai multe tipuri de dispozitive (laptop, telefon, tableta). Un aspect care va avea un impact semnificativ pentru utilizator este partea vizuală, care dacă este ușoară și intuitivă, îl va face să se întoarcă și să reutilizeze aplicația.

# 1. Descrierea problemei economice

## 1.1 Prezentarea domeniului abordat

Necesitatea întreținerii și reparării unei tehnologii este o cauză directă a invenției tehnologiei în cauză și, o scurtă incursiune în istoria autoturismului este, deci, esențială pentru a înțelege funcționarea a ceea ce astăzi reprezintă industria de service auto.

În 1879, Carl Benz inventează primul motor staționar cu un cilindru în doi timpi, pe care îl dezvoltă timp de încă 6 ani, culminând cu invenția unui vehicul care transportă două persoane, echipat cu un motorului compact cu un cilindru în patru timpi. Astfel, primul autoturism, patentat ca ”vehicul alimentat de motor pe gaz”, ia naștere în 1885. Apoi, în 1908, Henry Ford inventează Model T, care devine primul autoturism produs în masă. Este important de menționat că în afara inginerilor care au participat la construcția mașinii, foarte puțini oameni cunoșteau mecanismul de funcționare al acesteia în amănunt, fapt care a cauzat ca deținătorii de autoturisme să recurgă la ateliere de reparații de biciclete pentru întreținerea mașinilor cu motor. Astfel, s-a creat nevoia de ateliere de reparații care angajează mecanici specializați în autoturisme, și, odată cu aceasta, ia naștere industria de service auto. [3]

În prezent, se află în România 8 milioane de deținători de permise de conducere și aproximativ 9 milioane de mașini înmatriculate în țara noastră, conform datelor DRPCIV. Astfel, publicul țintă este unul destul de mare, iar problema pe care vreau să o expun este una destul de des întâlnită. [4]

Fiecare deținător de permis auto a vizitat un service auto măcar o dată, fie pentru o problemă apărută pe moment, fie pentru o simplă inspecție tehnică periodică. O problemă des întâlnită în acest context este cea a programării în cel mai scurt timp, cu precădere în cazuri urgente, cum ar fi tractări, defecțiuni ale motorului sau daune datorate accidentelor. Este important de menționat că numărul accidentelor, atât cele grave, cât și cele ușoare, este în continuă creștere, ceea ce evidențiază nevoia serviciilor auto ușor accesibile.

Conform ACEA, vârsta medie a parcului auto din România este de 16.4 ani, cu 4 ani mai mult decât media Uniunii Europene și dublul valorilor întâlnite în anumite țări vestice (Austria: 8.2 ani, Irlanda: 8.4 ani). [5] Remarcăm că România, alături de Lituania și Estonia, deține recordul pentru cea mai înaintată vârstă a parcului auto, iar majoritatea mașinilor aflate în funcțiune pe străzile din România au o vârstă mai mare de 10 ani. În acest context, devine evidentă problema accesibilității la un service auto de calitate, care poate prelungi considerabil durata de viață a unui autovehicul. [6]

Aceste tendințe se păstrează și în materie de autovehicule comerciale, care înregistrează vârste medii de peste 16 ani la toate categoriile. În același timp, întreținerea regulată este o prioritate pentru societățile comerciale care dețin acest tip de vehicule, intens utilizate într-o țară în care două treimi din mărfuri sunt transportate rutier. De aceea, este important ca orice

posesor de autovehicul să dispună de o platformă intuitivă, ușor de folosit și care maximizează beneficiile serviciilor auto disponibile în vecinătatea sa.

Nici perspectiva pieței de muncă nu trebuie neglijată. În materie de angajări directe în domeniul auto, România este pe primul loc, alături de Slovacia. [7] Sierra Quadrant raporta în 2020, pe baza datelor Registrului de Comerț și Ministerului Finanțelor, că România se află în topul european al țărilor cu cele mai multe companii care se ocupă cu activități de ”întreținere și reparare a autovehiculelor” (COD CAEN 4520), aflate în număr de 15.000 în țara noastră, acesta înregistrând o creștere de aproape 50% față de 2010. Domeniul de service auto este, fără îndoială, lucrativ, majoritatea companiilor înregistrând profituri pozitive, iar tot mai multe firme și ateliere independente își amplifică prezența online pentru a atrage clientela. Totuși, fiind o piață fragmentată, firmele mari concurează cu ateliere independente care fac compromisuri de calitate în scopul atingerii unui preț final cât mai mic, ceea ce indică importanța unei prezențe online accentuate a furnizorilor renumiți de servicii auto. [8]

Aceste condiții indică scara la care România acționează pe piața auto, atât în domeniul angajărilor, cât și al consumului, și, implicit, scara la care soluția propusă de mine în această lucrare poate influența experiența tuturor părților implicate în întreținerea unui autovehicul.

## 1.2 Prezentarea activității care va fi informatizată

Având în vedere multitudinea de opțiuni prezentă în domeniul abordat, posesorului de autovehicul îi revine responsabilitatea de a se informa și a alege un service auto care îi satisface nevoile, proces care poate deveni copleșitor, mai ales în orașele cu o ofertă variată. Pentru a înțelege acest proces, cât și importanța automatizării și digitalizării sale, vom sublinia nevoile deținătorului modern de autoturism, așa cum sunt evidențiate de Pereira et. al. (2007): raportul calitate-preț, disponibilitatea de a rezolva problemele întâmpinate de client și condiția generală a atelierului unde este reparată mașina. Acest studiu, în pofida limitării scopului său (populația de studiu este reprezentată de posesori de autoturisme din Brazilia), reflectă procesul decizional prin care trece și utilizatorul din România, iar concluziile sale pot fi generalizate în prezenta lucrare pe baza acestui argument.

### **Raportul calitate-preț**

Este lesne de înțeles motivația acestui factor decisiv în alegerea unui service auto.

Conform efectului calitate-preț prezentat de Nagle și Holden, cumpărătorii au o sensibilitate mai redusă la preț, cu cât prețurile mai mari indică o calitate mai înaltă. Contra pozitivă acestui efect indică tendința cumpărătorului de a fi influențat puternic de preț în comparația de produse și servicii atunci când prețurile acestora nu pot fi folosite ca referință pentru calitate, acesta fiind cazul pentru toate bunurile și serviciile ”normale”, i.e., excluzând chilipiruri și bunuri de lux.

Analiza datelor din Pereira et. al. indică faptul că posesorii de autoturism, atât cei care frecventează ateliere care aparțin de un brand cunoscut cât și ateliere independente, consideră raportul calitate-preț un factor decizional de mare importanță.

## **Disponibilitate**

Acest factor adresează, în fapt, două nevoi: serviciile oferite de firmă și istoricul autovehiculului.

În primul rând, intuiția indică accentul pus de client pe accesul la servicii de calitate. În mod evident, acestea pot fi oferite cu două condiții: interes sporit din partea mecanicului de a rezolva problemele întâmpinate de client și echipamentul pe care acesta îl are la dispoziție pentru a efectua sarcinile corespunzătoare. Interesul mecanicului se remarcă în studiul lui Pereira et. al. prin factorul decizional al disponibilității de a rezolva problemele clientului care se remarcă a fi de importanță reală pentru toți clienții, însă doar atelierele independente au un scor înalt în această categorie. Acest fapt indică relațiile bune pe care atelierele independente le au cu clienții lor, și, în schimb, nevoia ca service-urile de franșiză să întărească aceste relații.

În al doilea rând, istoricul autovehiculului este o preocupare inerentă a deținătorului, atât pentru utilizatorii de autoturisme, cât și pentru societăți comerciale. Acest istoric se circumscrie categoriei de management al contului de client, care include și stocarea facturilor de la vizitele în service. În România cu precădere, este esențial ca istoricul autovehiculului să fie ușor accesibil posesorului, având în vedere că aceste este o necesitate în tranzacțiile de vânzare-cumpărare, iar piața de autovehicule la mână a doua este de trei ori mai mare decât cea de autovehicule noi în țara noastră, aceasta excluzând dezmembrări și vânzări pe piese. [9]

## **Condiția atelierului**

Este inevitabil ca serviciile mulțumitoare de reparații, care asigură revenirea clientului, să poată fi îndeplinite doar cu ajutorul unor echipamente de calitate înaltă. Cu toate acestea, clientul rareori dispune de timpul sau cunoștințele tehnice necesare pentru a evalua aceste echipamente, iar condiția generală a service-ului poate fi utilizat ca un indicator bun de potențial al calității serviciilor oferite. În acest fel, putem spune că posesorul de autoturism ia în considerare imaginea de ansamblu a locației pentru a decide dacă va opta sau nu pentru locația în cauză. Când comparația între locații este efectuată în mediul online, este clar că fotografiile clare ale locației, echipamentului și angajaților pot spori șansele unui service de a fi selectat.

O analiză a nevoilor furnizorului de servicii este, de asemenea, utilă pentru a ilustra complexitatea procesului în totalitatea sa. Astfel, vom sublinia importanța unui proces simplu de gestiune a programărilor, precum și efectele economice oferite de centralizarea documentelor clienților și a inventarului.

## **Gestiune**

Organizarea eficientă și timpurie a programărilor la service permite managerului și mecanicilor să dedice mai mult timp activităților care contează, în speță reparații și revizii. Atunci când calendarul programărilor este gestionat eficient, este mai ușor pentru managerul de service să se asigure că acesta dispune de piesele de schimb necesare reparațiilor viitoare. Astfel, satisfacția clienților este sporită de timpul scurt de așteptare și promptitudinea mecanicilor, ceea ce aduce un plus de valoare atât furnizorului cât și beneficiarului de serviciu.

## **Centralizare**

Solicitățile de istoric sunt, în mod clar, numeroase și frecvente în România, unde autoturismele la mâna a doua reprezintă 82% din volumul tranzacțiilor de pe piață. Istoricul este în posesia service-urilor auto vizitate de posesorul automobilului și conține informații esențiale deciziei de a achiziționa un automobil, precum kilometrajul, specificații tehnice incluzând piese schimbate și potențiale daune. De aceea, importanța accesibilității acestui tip de documente nu poate fi supraestimată, iar stocarea lor într-o bază de date digitală conferă avantaje ambelor părți implicate: posesorul automobilului poate verifica starea acestuia cu ușurință și poate obține documentația necesară vânzării într-un timp scurt, iar furnizorul de servicii poate lua decizii mai compatibile cu automobilul în necesitate de reparații, fiindu-i ușor accesibil raportul detaliat al daunelor și specificațiilor acestuia.

### **1.3 Analiza sistemelor existente pe piață**

În România, există în prezent numeroase aplicații care au ca obiect optimizarea experienței programării la un service auto, pentru diverse servicii. Datorită acestui fapt, vom analiza comparativ platformele cu cel mai intens trafic. Acestea se adresează unor segmente de piață și utilizatori diferiți- de la posesorul de autoturism la managerul de service auto-, dar au propuneri de valoare similare: împuternicirea utilizatorului modern în calitate de consumator de tehnologie și factor decizional.

### **Aplicații destinate utilizatorilor unui anumit brand de autoturism**

Acest tip de aplicație, compatibile cu dispozitive Android și iOS, sunt dedicate celor care dețin un cont pe platforma online a producătorului de mașini și au ca scop înlesnirea procesului de mentenanță a autoturismului. În primul rând, aplicațiile oferă un serviciu de localizare a celor mai apropiate service-uri și reprezentanțe ale companie, precum și programare a vizitei la service, care este, în majoritatea cazurilor, sincronizată cu calendarul Google. În subsidiar, caracteristicile fiecărui autoturism înregistrat la un cont sunt redată și explicate într-o interfață intuitivă. Un exemplu elocvent este Mercedes me Service, a cărei



interfață este prezentată în Figura 1. În afara caracteristicilor menționate anterior, utilizatorii aplicației pot beneficia de:

- reduceri și oferte personalizate nevoilor autoturismului;
- informații privind detalii și specificații tehnice care se află în manualul autoturismului;
- conținut educativ care adresează conectivitatea între smartphone și autoturism.

Aplicații cu scop și funcționalitate similare au fost dezvoltate și de BMW (My BMW, BMW Connected), Land Rover (Land Rover InControl Remote), Audi (myAudi), Renault (MY Renault, R&Go).

În ciuda funcționalității extinse, acest tip de aplicații se concentrează în principal pe branding și revizii periodice. În mod clar, mentenanța este un factor cheie al longevității autoturismului, iar utilizatorii interesați de acest tip de aplicație nu sunt străini

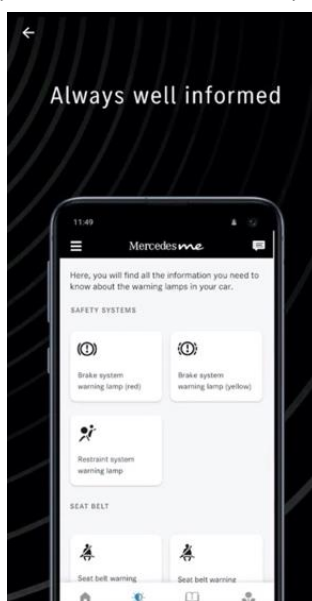


Figura 1. Aplicația Mercedes Me

**Sursa:** <https://www.mercedes-benz.ro/passengercars/mercedes-benz-cars/mercedes-me/mercedes-me-connect-basic-services.module.html>

## NETCAR

Spre deosebire de aplicațiile anterior menționate care au în prim plan posesorul de autovehicul, acest website adresează nevoile furnizorului de servicii pentru gestionare de inventar și contabilitate. Fiind construit în jurul managerului de service auto, website-ul oferă două opțiuni pentru gestionarea resurselor de care acesta dispune: *manager service auto* și *manager service utilaje*. Accesibilitatea generală este un punct forte al acestui website, fiind compatibil cu orice browser, cu versiuni pentru telefon și tabletă. Stocarea datelor în cloud asigură o copie de rezervă în cazul oricăror defecțiuni tehnice.

Funcționalitatea website-ului include:

- generarea documentelor necesare fiecărei vizite a unui client (facturi, devize), care pot fi trimise automat la firma de contabilitate afiliată cu service-ul;
- calcul automat al datei următoarei vizite a clientului în service pentru revizii periodice, precum și notificarea prin SMS în data anterioară vizitei;
- memorarea codurilor de piese și crearea unui șablon pentru adăugarea lor la o lucrare programată;
- interfață dedicată gestiunii ofertei de preț pentru o vizită a unui client.

The screenshot displays the NETCAR web application interface. The top navigation bar includes links for OPERE, LUCRARI, FACTURI, VEHICULE, CLIENTI, PRESCOPERITI, CONFIGURARE, and DECONECTARE. The main form is titled "DATE SERVICE" and contains various input fields for company and vehicle information. Below the form is a table with user data.

Numar	Nume Utilizator	Parola	Telefon Agent	Email Agent	Website
Elit	perfectservice	*****			
AutoTotal	perfectserviceca	*****			
AutoNet	perfectservice2	*****			
Aungburg	10015473	*****			

Figura 2. Aplicația NETCAR

Sursa: <https://managerserviceauto.ro/>

## 2. Analiza sistemului informatic

### 2.1 Descrierea generală a sistemului informatic

Încă de când eram copil am avut o pasiune pentru mașini, ele fiind, jucăriile mele preferate, probabil ca majoritatea băieților de la acea vârstă. În schimb, odată cu vârsta și cu dobândirea carnetului de conducere au început să apară problemele, în general cele administrative, legate de întreținerea autovehiculului personal.

În anul II de facultate, am primit mașina părinților mei în București cu care am împărtășit o serie de peripeții într-un oraș relativ nou pentru mine, care m-am deplasat în mare parte în subteran, cu metroul. Prima întâmplare a fost când o curea s-a rupt și nu am mai putut mișca mașina din loc, așa că a trebuit să găsesc un serviciu de tractare. Cum suntem în era digitală, primul lucru pe care l-am făcut a fost să caut pe Google. Majoritatea serviciilor pe care le-am găsit au fost fie prea departe de locul unde eram, fie prețul era prea mare sau chiar necomunicat.

O altă întâmplare cu același deznodământ a fost în momentul în care a trebuit să îmi fac o programare la un service pentru un simplu schimb de ulei sau când mi-a expirat asigurarea și ITP-ul fără ca eu să știu.

În urma acestor peripeții, m-am gândit că trebuie să fac ceva ca cei care ajung în situația mea, destul de mulți la număr, să nu mai piardă la fel de mult timp ca mine și să nu aibă aceleași probleme.

Aplicația eficientizează atât procesul de management al uneia sau a mai multor mașini, de înscriere a acestora pentru o anumită intervenție, cât și procesul de management al unui service, unde se pot vedea mult mai organizat lista mașinilor cât și statusul acestora. Fiecare utilizator va găsi în cadrul aplicației opțiunea de a adăuga una sau mai multe mașini în contul personal, vizualizarea service-urilor grafic cu ajutorul unor hărți interactive, programarea la un service din lista celor disponibile pentru o anumită intervenție (Diagnosticare, ITP, schimb de ulei), vizualizarea serviciilor de tractare prin intermediul unei hărți interactive.

Din perspectiva administratorului de service, el va avea acces la funcționalități cum ar fi vizualizarea tuturor mașinilor din cadrul service-ului respectiv, vizualizarea statusului mașinilor din cadrul service-ului respectiv, vizualizarea profitului din ultimele  $n$  zile, mașinile care sunt în așteptare pentru a fi preluate, o pagină de statistici în care managerul își poate vizualiza vânzările pentru a putea modifica strategia în vederea maximizării profitului și minimizării pierderilor.

## 2.2 Specificarea cerințelor sistemului informatic

Conform celor menționate anterior, aplicația denumită CarCare își propune să ofere utilizatorilor un mijloc eficient de gestionare a mașinilor personale sau a service-urilor pe care le administrează.

### 2.2.1 Diagrame ale cazurilor de utilizare

Cerințele funcționale avute în vedere în cadrul dezvoltării aplicației vor fi identificate și modelate prin intermediul diagramelor cazurilor de utilizare.

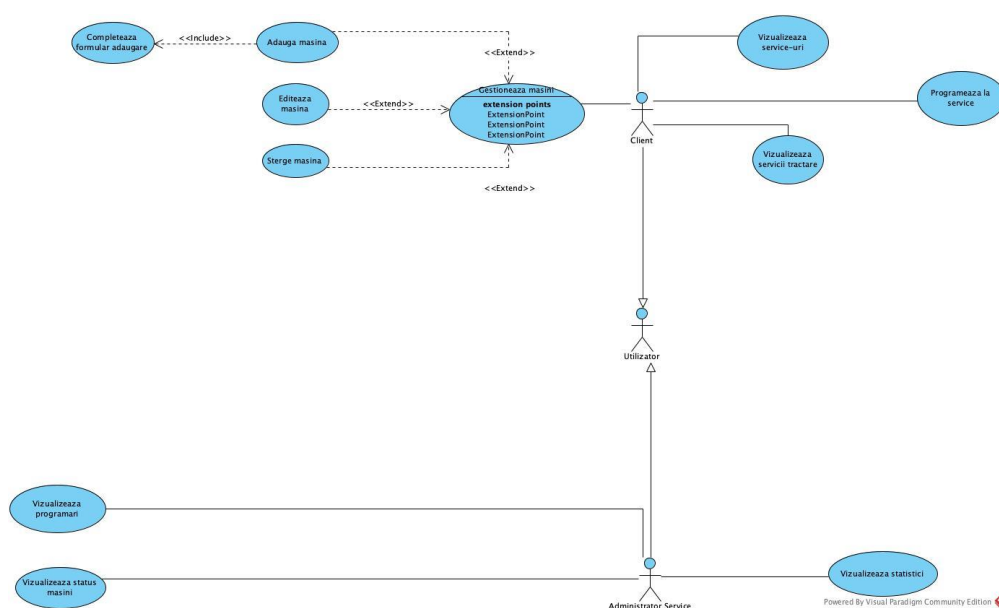


Figura 3. Diagrama generală a cazurilor de utilizare

Având în vedere Figura 3, sistemul informatic prezintă două tipuri de utilizatori: clienți și administratori service.

După logarea în aplicație, utilizatorul este clasificat în una din cele două categorii, având anumite drepturi și acces la anumite funcționalități din cadrul ei, astfel distingându-se mai multe cazuri de utilizare pentru fiecare tip.

Clientul va putea gestiona mașinile proprii în cadrul aplicației, acest lucru constând în adăugarea, editarea și ștergerea unei mașini. El poate vizualiza lista de service-uri autorizate prezentă, își poate programa un autovehicul pentru o anumită intervenție în cadrul unui service sau poate vizualiza și serviciile de tractare și locația lor.

Cea mai importantă funcționalitate de care va beneficia un administrator de service este aceea de administrare de mașini care constă în vizualizarea programărilor și a statusurilor mașinilor, cât și vizualizarea statisticilor pentru a putea maximiza profitul.

## 2.2.2 Diagrama cazului de utilizare programare service

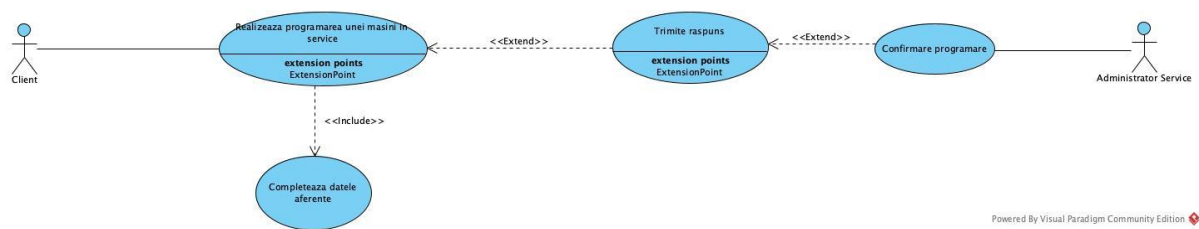


Figura 4. Diagrama cazului de utilizare programare service

Element al cazului de utilizare	Descriere
Cod	CU02
Stare	Schiță
Scop	Programarea unei mașini in service
Nume	Programare service
Actor principal	Client
Descriere	Clientul realizează o cerere pentru programarea mașinii in service
Precondiții	Clientul trebuie sa aibă aplicația instalata, conexiune la internet, sa aibă cont activ si cel puțin o mașină adăugată
Postcondiții	Programarea a fost realizata cu succes si clientul primește o confirmare a acesteia
Declanșator	Clientul realizează programarea
Flux de bază	<ol style="list-style-type: none"> <li>1. Clientul se autentifica in aplicație</li> <li>2. Clientul selectează opțiunea de programare a unei mașini</li> <li>3. Clientul completează formularul de adăugare</li> <li>4. Se verifica datele</li> <li>5. Administratorul de service confirma, daca totul este in regula, programarea</li> </ol>
Fluxuri alternative	-
Relații	-
Frecvența utilizării	Frecvent
Reguli ale afacerii	Se poate realiza programarea daca toate condițiile sunt îndeplinite

Tabel 1. Descrie textuală a cazului de utilizare programare service

## 2.2.3 Diagrama cazului de utilizare programare service

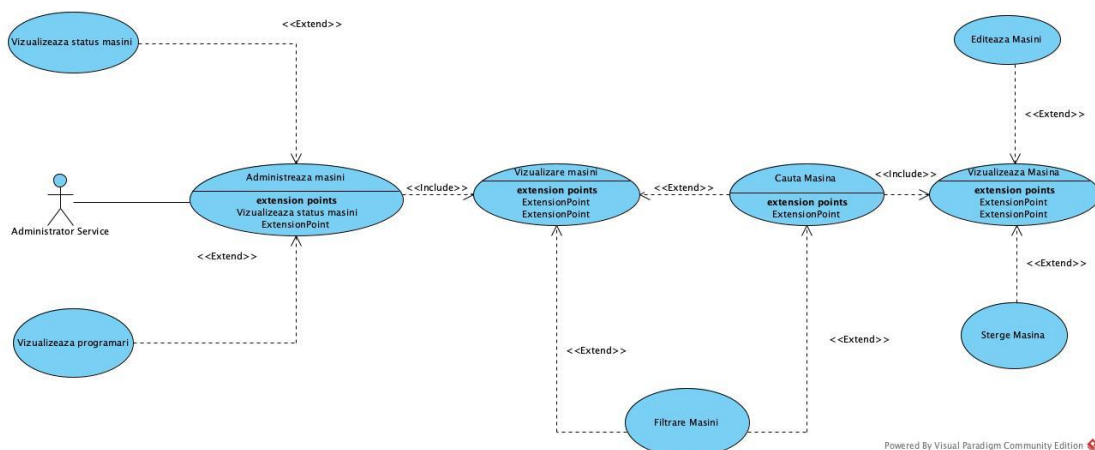


Figura 5. Diagrama cazului de utilizare administrare mașini

Element al cazului de utilizare	Descriere
Cod	CU03
Stare	Schiță
Scop	Administrarea mașinilor din service
Nume	Administrare Mașini
Actor principal	Administrator Service
Descriere	Administratorul poate gestiona toate informațiile legate de mașinile din cadrul service-ului sau
Precondiții	Administratorul are aplicația instalată, are acces la internet, este logat și are cel puțin o mașină în cadrul service-ului
Postcondiții	Administratorul a modificat tot ce își dorea la mașinile din cadrul service-ului
Declanșator	Administratorul accesează pagina cu mașinile prezente în service
Flux de bază	<ol style="list-style-type: none"> <li>1. Administratorul este autentificat în aplicație</li> <li>2. Administratorul accesează pagina mașinilor</li> <li>3. Administratorul poate vizualiza lista mașinilor</li> <li>4. Administratorul poate cauta o anumită mașină după anumite câmpuri</li> <li>5. Administratorul poate filtra mașinile după anumite criterii</li> <li>6. Administratorul poate edita o anumită mașină sau o poate șterge</li> </ol>
Fluxuri alternative	-
Relații	-
Frecvența utilizării	Foarte frecvent

Reguli ale afacerii	Se poate realiza administrarea daca exista cel puțin o mașină in service
---------------------	--

Tabel 2. Descrierea textuală a cazului de utilizare administrare mașini

## 2.3 Analiza sistemului existent

### 2.3.1 Diagrame de activitate

- Diagrama de activitate „Adăugare mașină”

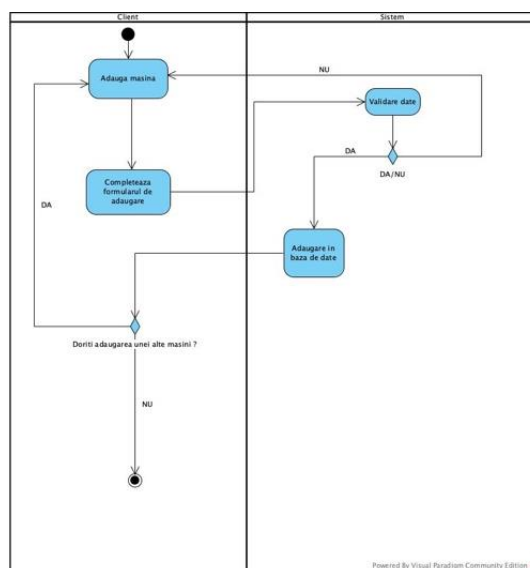


Figura 6. Diagrama de activitate "Adăugare mașină"

În figura 6 este reprezentată adăugarea unei mașini de către un client. Acesta va accesa opțiunea de adăugare mașină din meniul aplicației, urmând să fie redirecționat către completarea formularului cu date relevante despre mașină (Marca, an, capacitatea motorului, număr de înmatriculare). După completarea acesteia, sistemul va verifică dacă datele introduse sunt corecte. Dacă datele sunt incorecte, utilizatorul este întors la introducerea lor corectă cu ajutorul validărilor. În cazul contrar, sistemul introduce mașina in baza de date și este întrebat dacă dorește să introducă o altă mașină, in cazul afirmativ, tot procesul este reluat iar in cazul negativ, procesul se încheie.

○ Diagrama de activitate „Programare service”

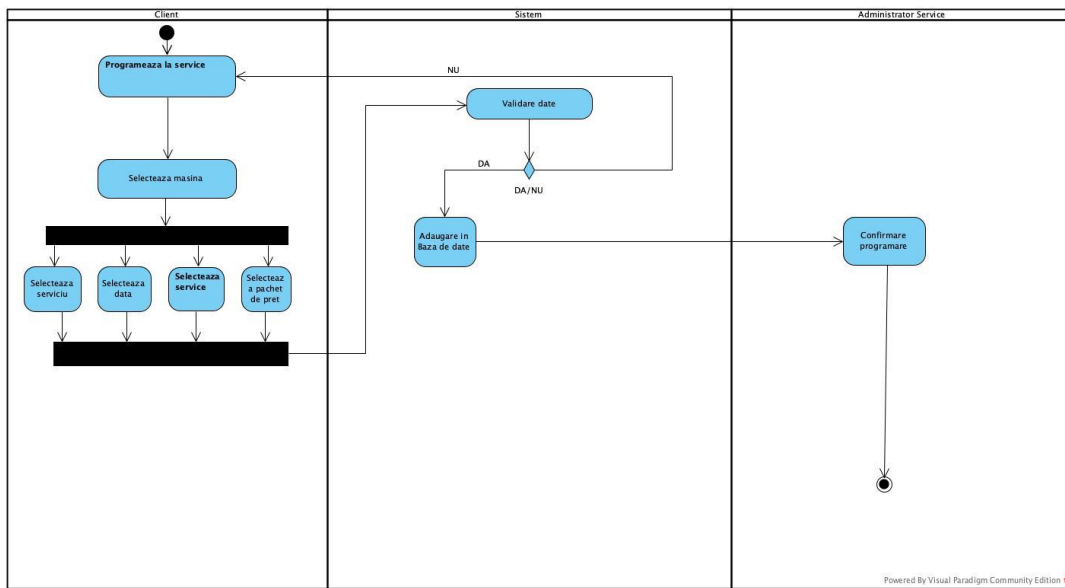


Figura 7. Diagrama de activitate "Programare service"

În figura 7 se regăsește diagrama de activitate pentru programarea unei mașini în service. Utilizatorul intră în această parte a aplicației unde este întâmpinat de opțiunea de a selecta mașina dorită, urmând să introducă o serie de date necesare programării (selectarea serviciului, a datei, a service-ului și a pachetului de preț). După ce toate aceste date sunt introduse, acestea sunt trimise în sistem pentru validarea lor. Dacă ele sunt valide, programarea este adăugată în baza de date, administratorul confirmă programarea iar activitatea este încheiată. În cazul în care datele nu trec de validări, utilizatorul este întors în primul pas unde trebuie să introducă datele corecte.

○ Diagrama de activitate „Vizualizare servicii tractare”

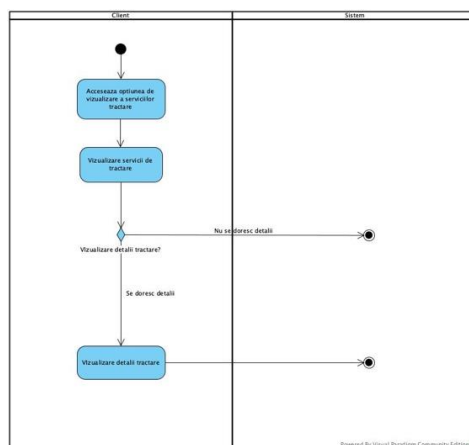


Figura 8. Diagrama de activitate "Vizualizare servicii tractare"



Figura 8 prezintă activitatea de vizualizare a serviciilor de tractare, acțiune realizată de un client din cadrul aplicației în momentul în care el accesează opțiunea de vizualizare a serviciilor de tractare, în urma căreia este afișată lista serviciilor de tractare în cadrul unei hărți interactive. În nodul decizional se hotărăște dacă utilizatorul vrea să vadă detalii legate de un anumit serviciu de tractare sau nu. Ambele variante duc la încheierea activității.

### 2.3.2 Diagrame de clase

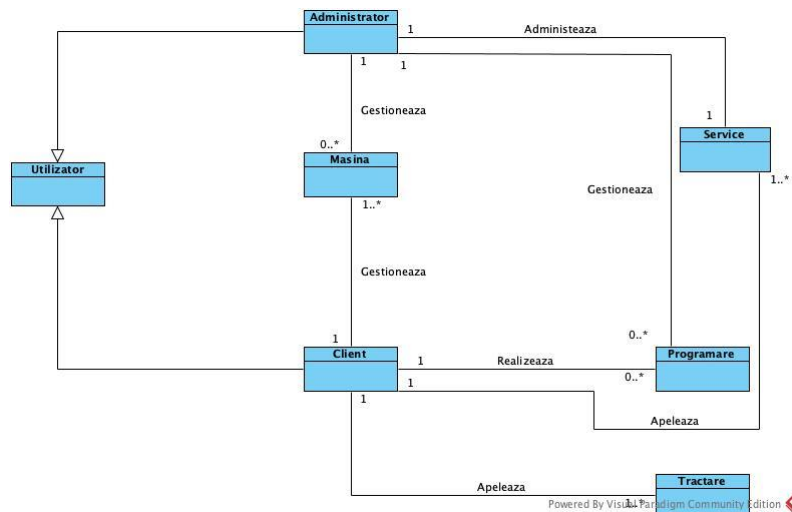


Figura 9. Diagrama de clase

În figura de mai sus se regăsește diagrama de clase în cadrul căreia sunt prezentate cele două tipuri de utilizatori : administrator , care poate gestiona mașinile care se afla în service, poate gestiona și programările viitoare, ocupându-se astfel de clienți, și clientul, care își poate gestiona mașinile personale (adăugare, editare, ștergere), poate realiza programări, apela la servicii de tractare sau service.

### 2.3.3 Diagrame de stare

- Diagrama de stare a unui utilizator

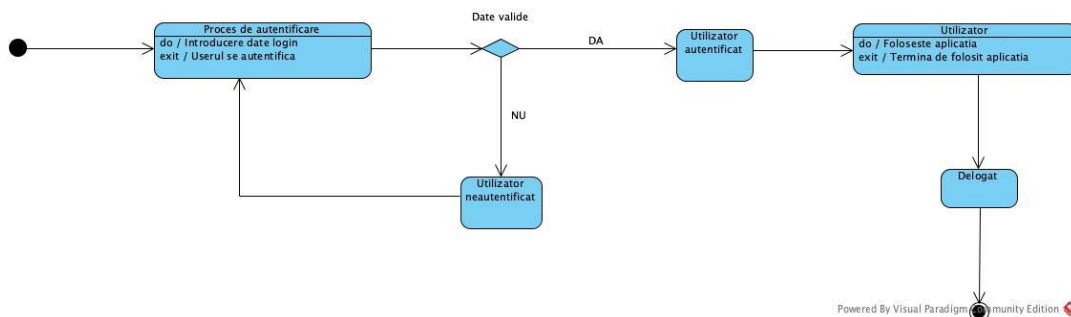


Figura 10. Diagrama de stare a unui utilizator

În diagrama de mai sus sunt prezentate stările prin care trece orice utilizator de când deschide aplicația până când termină de lucrat pe ea. Acesta trece prin procesul de autentificare iar dacă datele sunt corecte, el va fi trimis în aplicație și o poate folosi. Când acesta termină ce are de făcut, se deconectează.

- Diagrama de stare a unei programări

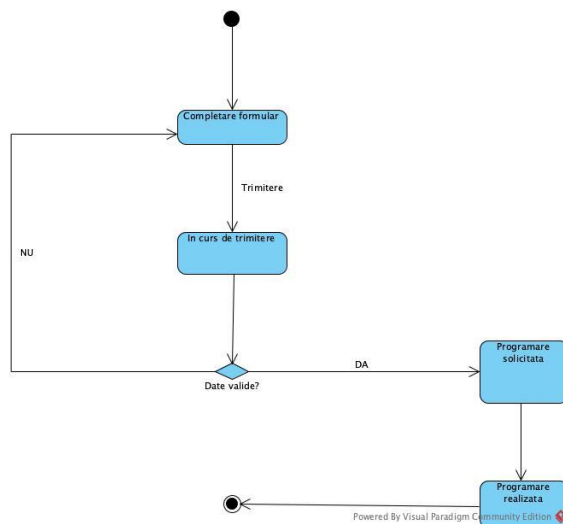


Figura 11. Diagrama de stare a unei programări

În figura 11 este prezentată diagrama de stare a unei programări înainte să fie creată. Această este realizată de un client care dorește să își repare ceva la unul din autovehiculele personale. După completarea formularului de adăugare, datele sunt verificate de sistem, dacă acestea sunt valide, programarea se realizează și se adaugă în baza de date. În caz contrar, utilizatorul trebuie să refacă cererea.

### 2.3.4 Diagrame de interacțiune

- Diagrama de interacțiune pentru logare

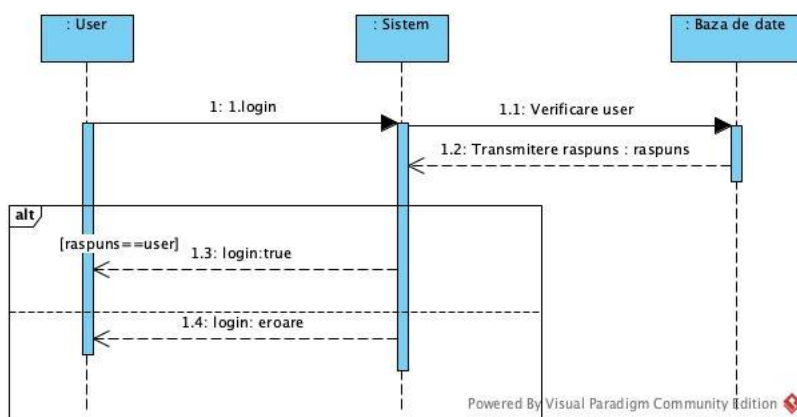


Figura 12. Diagrama de interacțiune pentru logare

Diagrama de interacțiune pentru logare în cadrul aplicației este prezentată în figura 11. Cei 3 parteneri care interacționează sunt userul, sistemul și baza de date. În momentul în care userul dorește să se logheze, el trimite un mesaj sincron către sistem, care îi verifică existența în baza de date și trimite un răspuns. Dacă acesta este pozitiv, userul va fi logat în aplicație. Dacă este negativ, va fi trimis un mesaj de eroare și userul va trebui să reintroducă datele.

- Diagrama de interacțiune pentru programarea unei mașini

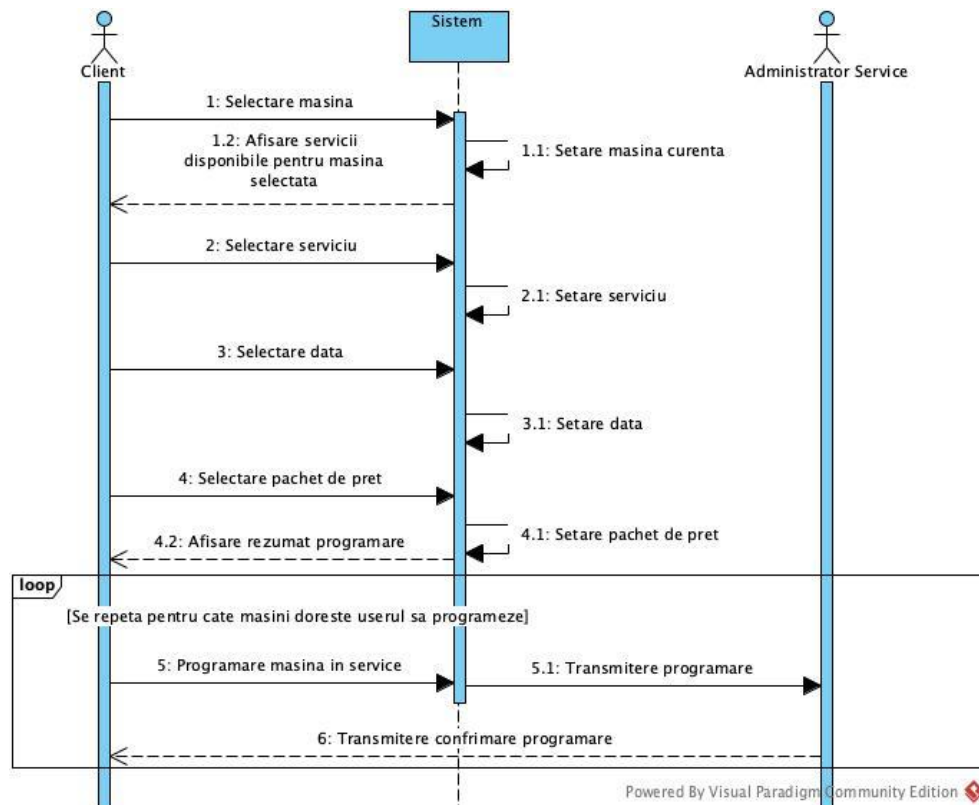


Figura 13. Diagrama de interacțiune pentru programarea unei mașini

În diagrama de mai sus este prezentată interacțiunea dintre client, sistem și administratorul de service în momentul în care se realizează o programare de mașină. Userul va selecta mașina pe care dorește să o programeze, primind un răspuns de la sistem cu lista de servicii disponibile pentru mașina selectată. Următorul pas este selectarea serviciului, a datei și a pachetului de preț. După ce fiecare tip de dată este introdus, sistemul le setează și la sfârșit trimite un rezumat al programării. *Loop-ul* are scopul de a arăta că procesul se poate repeta pentru câte mașini dorește userul.

- Diagrama de interacțiune pentru adăugarea unei mașini

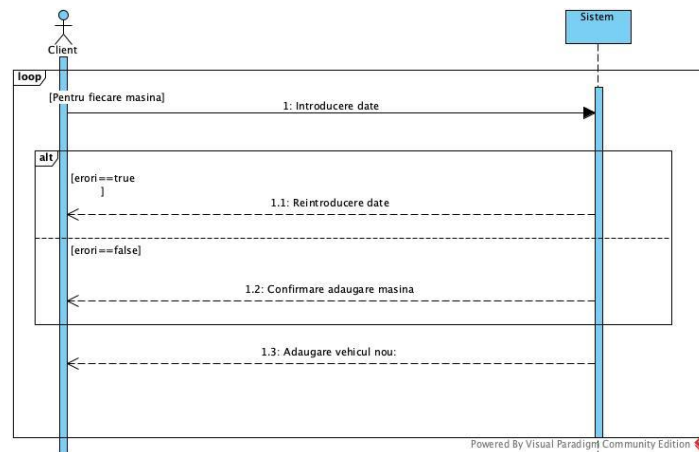


Figura 14. Diagrama de interacțiune pentru adăugarea unei mașini

În figura de mai sus este reprezentată interacțiunea dintre client și sistem în momentul adăugării unei mașini. Fragmentul *loop* arată că pentru fiecare mașină, studentul va introduce date, iar în fragmentul *alt* se verifică dacă datele introduse de către utilizator sunt corecte.

### 2.3.5 Diagrame de procese și colaborare în BPMN

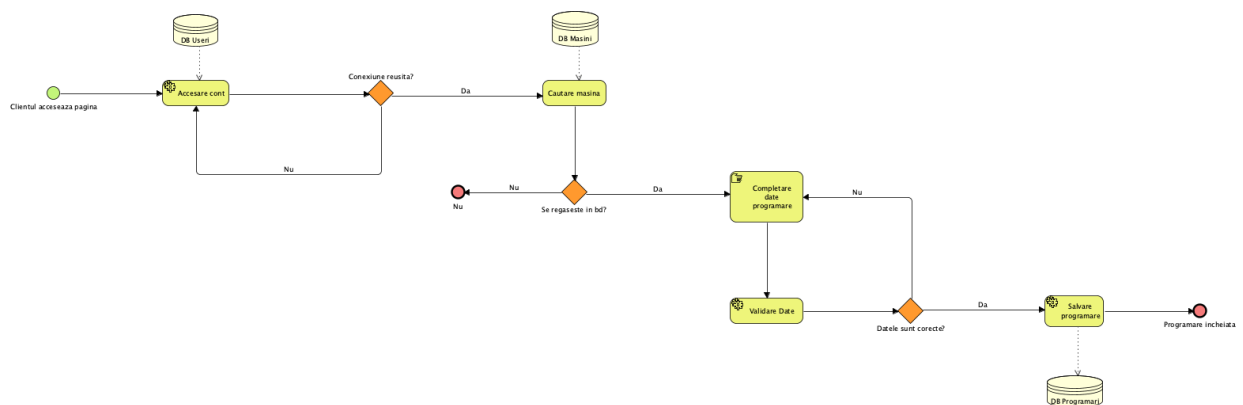


Figura 15. Diagrama de proces (programare mașină)

În figura de mai sus este reprezentată diagrama de proces pentru programarea unei mașini. Pentru ca acest lucru să fie posibil, clientul trebuie mai întâi să se logheze, iar dacă credențialele lui sunt valide este redirecționat către pagina principală. Din pagina principală el intră pe pagina de programare mașină și completează formularul. Datele sunt validate, iar dacă acestea sunt corecte, programarea este introdusă în sistem. În cazul negativ, clientul primește o notificare și este dus la reintroducerea datelor.

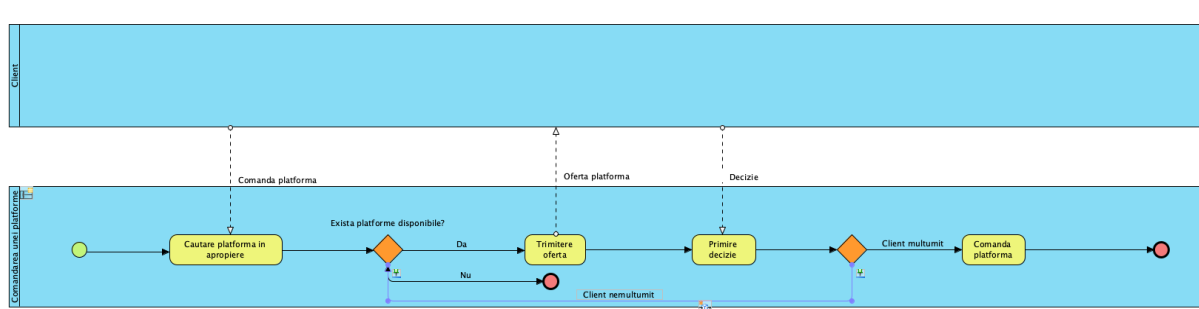


Figura 16. Diagrama de colaborare (comandare platforma)

Când clientul comandă o platformă, el își introduce adresa, iar în sistem este căutată o platformă în apropierea locației clientului. Dacă se găsește o platformă disponibilă, clientul primește oferta cu toate detaliile necesare, urmând să ia o decizie pe care o transmite înapoi către sistem. Dacă clientul transmite un răspuns pozitiv, platforma este confirmată și plasa către aceasta iar activitatea se încheie.

### 3. Proiectarea sistemului informatic

#### 3.1 Diagrama de clase detaliată

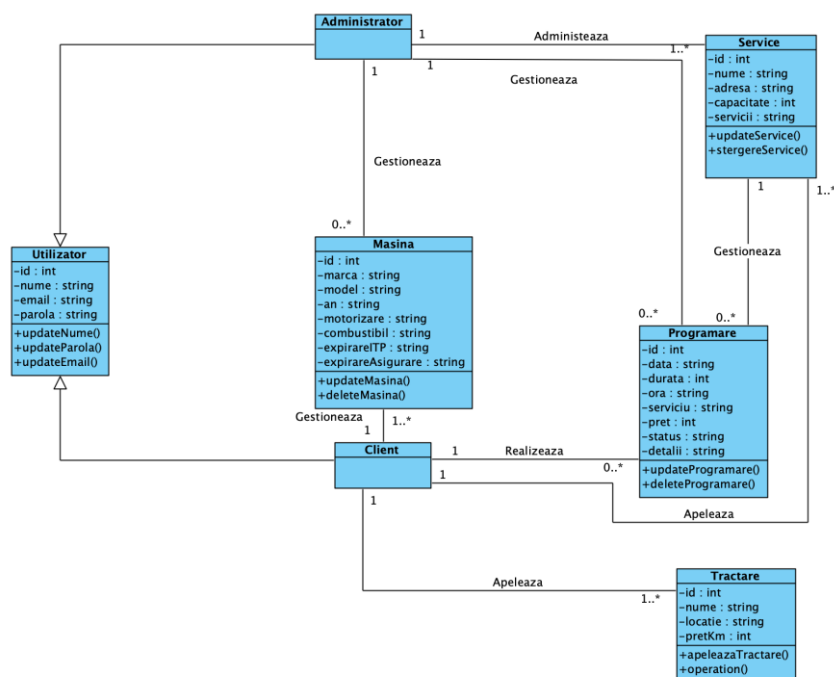


Figura 17. Diagrama claselor detaliată

Aplicația va avea două tipuri de utilizatori: client și administrator. Asupra utilizatorului avem metode pentru modificarea credențialelor (nume, email, parola). Administratorul se va ocupa cu gestiunea programărilor din service-uri, acestea putând fi actualizate sau șterse în funcție de preferințele acestuia. De asemenea, el poate modifica atât datele din interiorul unuiu sau a mai multor service-uri, cât și datele specifice mașinilor.

Clientul își va putea realiza programări în cadrul unuiu dintre service-urilor prezente în sistem, va avea posibilitatea să apeleze la serviciile de tractare existente și își va putea gestiona mașinile personale adăugate în sistem.

#### 3.2 Diagrama bazei de date

Baza de date a aplicației este formată dintr-un număr de 5 tabele, toate fiind legate între ele prin cel puțin o relație.

Tabela Utilizatori este formată din 4 câmpuri (id – cheie primară, nume, email, parola) aflându-se în relație de 1 la n cu tabela Mașină, referențiată prin id-ul utilizatorului.

Tabela Mașini stochează informațiile necesare integrării acestuia în aplicație (id, marca, model, an, motorizare, combustibil, expirareITP, expirareAsigurare). Aceasta se află în relație

1 la n cu tabela Tractări, prin câmpul idMașină și într-o relație de 1 la n cu tabela Programări prin același câmp de referință.

În tabela Programări se înregistrează datele despre programările efectuate de către clienți în cadrul aplicației, având următoarele câmpuri: id, data, durata, ora, serviciu, preț, status și detalii, aflându-se în relație cu tabelele Mașini și Service-uri.

Tabela Service-uri conține informațiile fiecărui service prezent în aplicație. Are în componența sa 6 câmpuri (id, nume, adresa, capacitate, servicii) și o relație de 1 la n cu tabela Programări prin idService.

În cadrul tablei Tractări se regăsesc datele solicitărilor de tractare efectuate de către clienți, având în componență 4 câmpuri (id, nume, locație, prețKm) și se află în relație cu tabela Mașini.

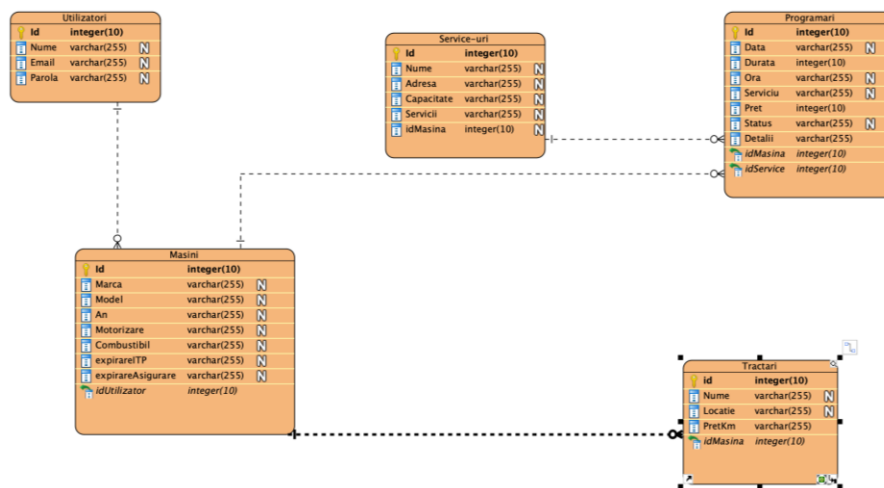


Figura 18. Diagrama bazei de date

### 3.3 Proiectarea interfețelor utilizatorului

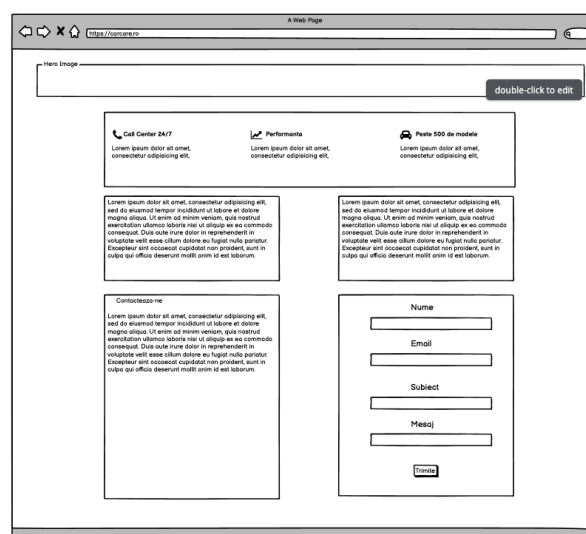


Figura 19. IU3 – Pagina principală

În momentul accesării aplicației utilizatorul vizualizează pagina principală, unde i se va oferi o privire de ansamblu asupra funcționalităților și serviciilor oferite prin intermediul unei interfețe intuitive. În prima parte a paginii, sunt prezentate motivele pentru care vizitatorii ar alege serviciile oferite de noi în detrimentul competiției existente pe piață.

În continuare, utilizatorul are un acces foarte rapid la datele de contact ale administratorului, având opțiunea de a trimite un email către echipa tehnică din cadrul acestei pagini.

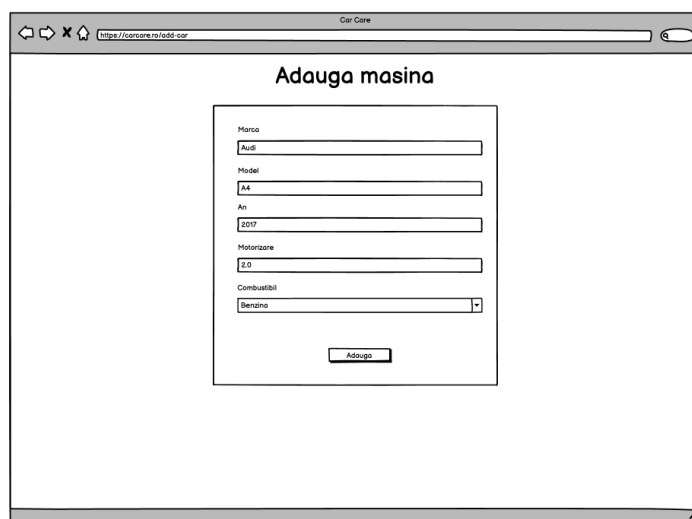


Figura 20. IU4 – Adaugă mașină

Așa cum am menționat anterior, clientul va avea posibilitatea să își adauge una sau mai multe mașini în vederea unor programări ulterioare la unul dintre service-urile prezente în aplicație, a unor solicitări de tractare sau doar pentru a avea o evidență a datei de expirare a anumitor acte (asigurare, inspecție tehnică periodică), acest lucru fiind posibil prin intermediul formularului din cadrul paginii de mai sus.

După completarea formularului, mașina este adăugată în baza de date în contul utilizatorului curent, oferindu-i-se opțiunea de a mai adăuga un alt autovehicul.

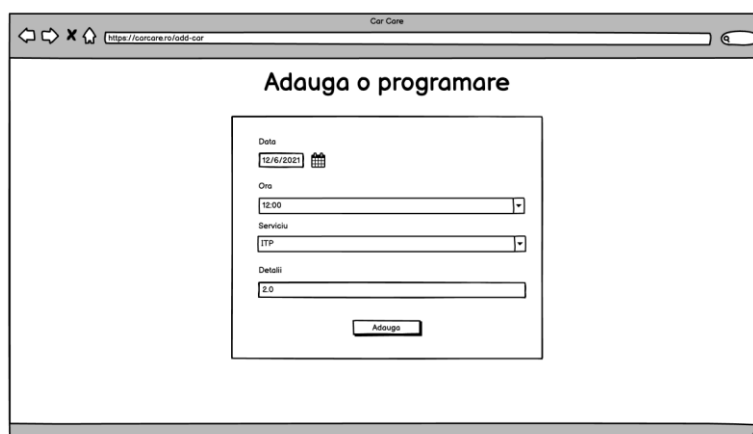


Figura 21. IU5 – Adaugă programare



Atunci când clientul are nevoie de unul din serviciile oferite de către service-urile prezente în cadrul aplicației, el va face acest lucru în cadrul formularului prezentat în figura de mai sus. El va hotărî detaliile necesare realizării unei programări, aceasta fiind înregistrată ulterior în baza de date.

### 3.4 Harta de navigare printre interfețe

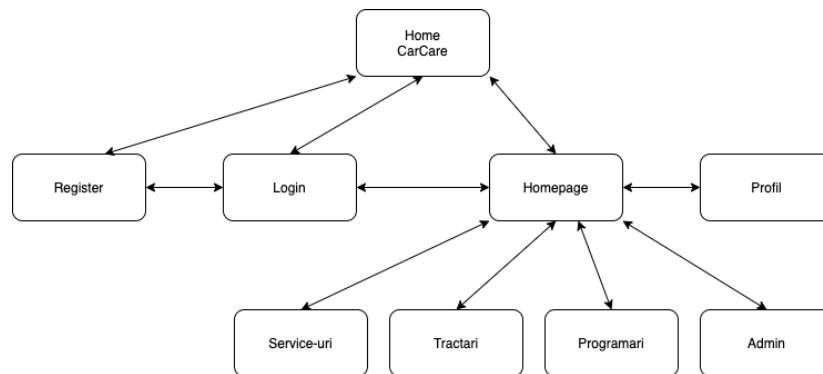


Figura 22. Harta de navigare printre interfețe

În figura de mai sus, sunt evidențiate rutele paginilor aplicației și modul în care utilizatorii pot naviga printre acestea. După finalizarea procesului de autentificare, utilizatorul este redirecționat către pagina principală, din care va avea acces către funcționalitățile prezente în sistem.

### 3.5 Diagrama de componente

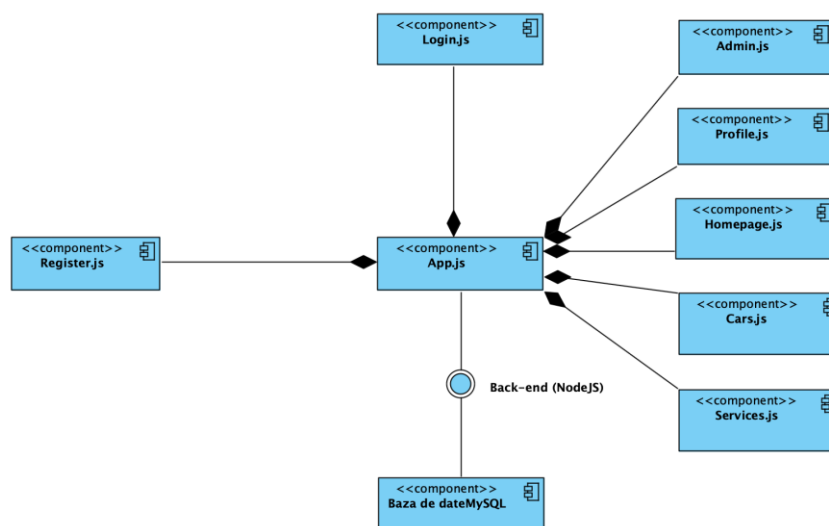


Figura 23. Diagrama de componente

În diagrama de mai sus sunt prezentate componentele principale necesare în dezvoltarea sistemului informatic, componenta de baza fiind App.js, conținând toate celelalte componente: Login.js și Register.js necesare în cadrul procesului de autentificare, Admin.js pentru accesarea informațiilor de tip administrativ, Homepage.js ce conține informațiile utile pentru prezentarea aplicației, Profile.js având în interiorul ei datele atât despre utilizatorul logat cât și o privire de ansamblu asupra activității acestuia în aplicație, Services.js ce prezintă o listă a service-urilor disponibile iar Towing.js prezintă, analog, lista serviciilor de tractare disponibile.

Accesul la baza de date *MySQL* se face prin *back-end-ul* realizat în *NodeJS*, aici urmând a fi stocate toate datele necesare rulării aplicației cu succes.

### 3.6 Diagrama de desfășurare

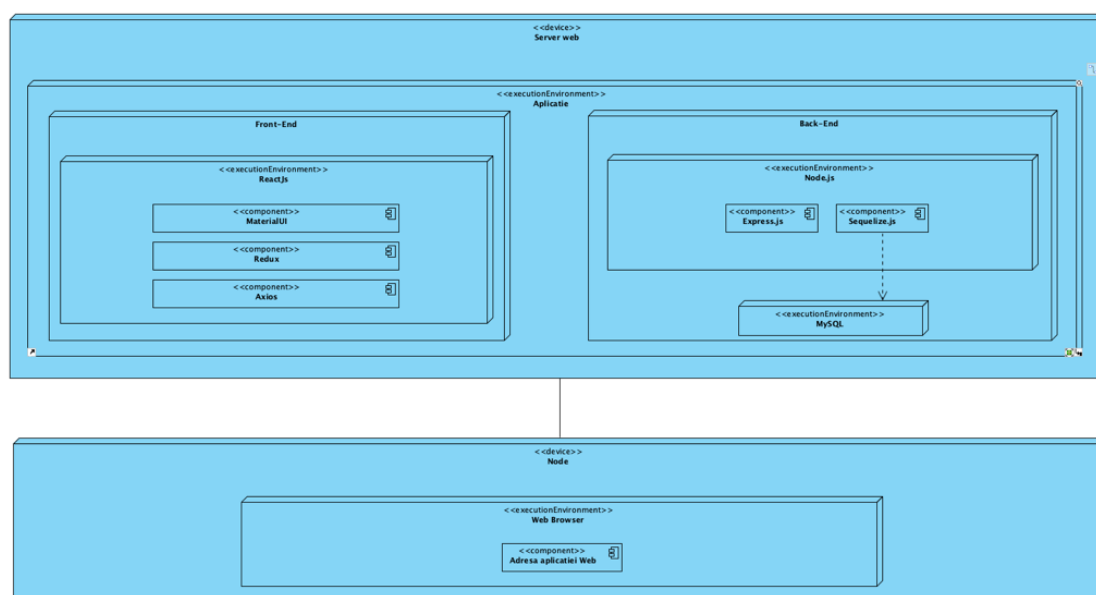


Figura 24. Diagrama de desfășurare

Aplicația rulează pe un server web și are două mari secțiuni, **front-end** și **back-end**. Partea de *front-end* a fost realizată cu ajutorul mediului de execuție *ReactJS*, o librărie de *JavaScript* în interiorul căreia se află 3 mari componente, *MaterialUI*, *Redux* și *Axios*. *MaterialUI* este un *framework* de componente ce ne oferă o soluție rapidă, eficientă de a crea o interfață ce se poate mula pe toate tipurile de ecrane. *Redux* este o bibliotecă ce ne va ajuta la gestionarea stărilor aplicației iar *Axios* va fi puntea de legătură dintre *front-end* și *back-end*, realizând operații asincrone între cele două secțiuni.

Pe partea de *back-end* am folosit mediul de execuție *NodeJs*, un mediu de rulare open-source care ne oferă posibilitatea rulării codului *JavaScript* în afara *browserului* web. *ExpressJS* este un *framework* ce ne va ajuta la construirea *API-urilor* iar *Sequelize.js* este o librărie ce este bazată pe promisiuni, oferind un acces rapid către baza de date relațională *MySQL*.

## 4. Proiectarea sistemului informatic

### 4.1 Tehnologii informatice utilizate

Pentru a crea o aplicație în ziua de azi, opțiunile sunt într-un număr foarte mare, din punct de vedere al tehnologiilor disponibile pe internet. În momentul în care am hotărât ce voi folosi pentru aplicația mea, am luat în considerare mai mulți termeni: care este media de vârstă a utilizatorilor, din ce medii provin și mai ales, care este venitul lor lunar mediu. În urma vizualizării tuturor variabilelor, am luat hotărârea de a crea o aplicație web ce poate fi folosită pe orice dispozitiv ce are acces la internet.

Programarea web are la baza doi mari piloni care nu ar putea exista unul fără celălalt, *front-end* și *back-end* sau *client-side* și *server-side*.

Pentru front-end, am folosit următoarele tehnologii:

- *JavaScript*
- *ReactJS*
- *Redux*
- *Material UI*
- *Maps JavaScript API*
- *Geocoding API*

### JavaScript

*JavaScript* a apărut în anul 1995 ca o soluție pentru a putea face posibilă adăugarea de programe în *browserul* cel mai folosit din acea perioadă, și anume *Netscape Navigator*. De atunci a fost adoptată de toate *browsers*le web, reușind să facă posibilă prezența aplicațiilor web în care utilizatorul poate interacționa direct cu pagina, fără să fie nevoie de un *reload* la fiecare acțiune făcută de acesta. [10]

Un lucru foarte important este faptul că *JavaScript* nu are nici o legătură cu limbajul de programare Java. Similaritatea numelor a fost mai mult o tactică de marketing deoarece în momentul introducerii limbajului, *Java* avea o popularitate ascendentă și a fost luată decizia de a avea aceasta similitudine lexicală. [10]

Conform *Tiobe* index, *JavaScript* a fost, încă din 2001, în top 10 cele mai folosite limbaje de programare, fluctuând la fiecare 5 ani iar anul acesta este pe locul 7. [11] Probabil întrebarea este de ce este atât de popular? Răspunsul este simplu, în primul rând, poate fi folosit fie direct în *browserul* web, în detrimentul altor limbaje de programare unde suntem nevoiți să le descărcăm pe mașina noastră locală pentru a putea accesa toate fiecare caracteristică a acestuia, fie în orice *Integrated development environment(IDE)* (*WebStorm*) sau editor de cod (*Visual Studio Code*) prin crearea unui fișier cu extensia **.js** și rularea acestuia într-un *browser*, deoarece este un limbaj interpretat, neavând nevoie de compilare. [12]

În al doilea rând, un alt pilon pe care se bazează *JavaScript* este programarea pe baza evenimentelor, existând deja incorporate funcții ca „*onClick*” sau „*onChange*” care așteaptă ca utilizatorul să facă o anumită acțiune pentru a rula un anumit segment de cod.

În ultimul rând, odată cu apariția *Node.js*, un *framework* care rulează cod de *JavaScript* în afara *browserului*, acest lucru crescându-i dramatic popularitatea, deoarece putea fi folosit un singur limbaj de programare pentru o întreagă aplicație web. Ca și exemplu, se poate comunica cu bazele de date, se pot apela metode *HTTP* și genera conținut dinamic, totul cu o singură sintaxă.

## ReactJS

Pentru a putea face o aplicație web modernă, cât mai ușor scalabilă și utilizabilă pe cât mai multe dispozitive, am hotărât să folosesc un *framework*, cele mai cunoscute fiind *ReactJS*, *AngularJS* și *VueJS*. *ReactJS* este o librărie de *JavaScript* dezvoltată de cei de la *Facebook* cu prima versiune lansată în 2013. Principalul avantaj al acestui *framework* este atât organizarea sa internă, întrucât aceeași parte din cod este responsabilă pentru partea sa vizuală cât și pentru comportament cât și comunitatea, având peste 6.700 de urmăritori pe *GitHub*. [13] *AngularJS* a fost fondat de către *Google* în anul 2010, având în spate *TypeScript*, un limbaj de programare dezvoltat de către *Microsoft*. Componentele din cadrul acestui *framework* sunt recunoscute sub numele de directive, ele fiind cele mai apropiate dintre cele 3 de programarea orientată obiect. [14] *VueJS* a fost fondat de către *Evan You* și a apărut în anul 2014, avantajele lui fiind îmbinarea tuturor celor 3 părți ale unei pagini (*HTML*, *CSS*, *JavaScript*) denumite *template*, *style* și *script* într-un singur fișier, având astfel un control mult mai mare asupra *scalabilității* fiecărei componente în parte. Apărând cel mai târziu, el a îmbinat tot ce au bun celelalte două și a avut o creștere foarte mare în popularitate în ultimii ani. [15]

Am ales *ReactJS* pentru că, din punctul meu de vedere este cel mai accesibil, cel mai ușor de înțeles și cu cele mai multe probleme deja rezolvate pe forumul său dedicat.

## Redux

*Redux* este o librărie pentru a ajuta la managementul stărilor în cadrul aplicațiilor dezvoltate în *React*, *React Native*, *Angular* sau orice alta librărie, fiind un container predictibil de stare. Comparativ cu alte librării similare, *Redux* folosește un singur *store*, acest aspect fiind des numit ca „*single source of truth*” fiind singurul loc unde stările pot fi acceptate. Motivul pentru care *Redux* este cel mai folosit din categoria sa este că acesta face schimbările de stare predictibile printr-un set bine definit de reguli. [16]

Prima și probabil cea mai importantă regulă este că stările trebuie să fie imutabile, rezultând astfel performanțe mai bune ale aplicației, dezvoltare și *debugging* mult mai ușor. Cea de-a doua regulă este că schimbările trebuie să aibă loc doar cu ajutorul acțiunilor, iar cea

de-a treia este folosirea reducerilor, funcții pure care specifică cum acțiunile modifică starea aplicației. [16]

## Material UI

*Material UI* este unul din cele mai eficiente și puternice instrumente pentru construirea aplicațiilor *web*, punctul său forte este adăugarea animațiilor și design-urilor combinate cu funcționalități tehnice. Este bazat pe *Material Design*, un limbaj de design realizat de *Google* în anul 2014, fiind bazat pe un sistem de tip grilă și pe componente *responsive*, care au un aspect plăcut pe orice tip de dispozitiv. [17]

Poate fi integrat pe toate cele trei mari *framework-uri* de *JavaScript*, *ReactJS*, *AngularJS* și *VueJS* pentru a aduce un iz profesional și plăcut aplicației prin viteza sa de execuție, dimensiunile reduse ale acestuia (sub *1KB*), accesul la o temă predefinită sau crearea propriei teme pentru a putea avea o consistență solidă în cadrul codului și prin lista generoasă de componente predefinite. [17]

## Maps JavaScript API

În cadrul *Google* există o serie de funcționalități care ne sunt oferite pentru a îi integra serviciile în cadrul aplicației noastre. Unul din acestea este *Maps JavaScript API*, care ne ajută să introducem în sistemul nostru hărțile *Google*, cu conținutul și imaginile noastre.

*API-ul* vine cu patru tipuri de bază de hartă modificabile prin stil, evenimente și multe alte librării. [18]

## Geocoding API

Un alt *API* din suita de la *Google* care ne este de folos este cel de *Geocoding*, care, alături de *Places API* ne ajuta să convertim adrese fizice, cum ar fi Strada Căderea Bastiliei 2 în coordonate geografice, cum ar fi latitudine 44.447736 și longitudine 26.09542, pentru a putea introduce adrese grafic direct pe o hartă. [19]

Pentru *back-end* am hotărât să folosesc următoarele:

- *Node.js*
- *MySQL*
- *Sequelize*
- *Passport.js*

## Node.JS

La conferința europeană *JSConf* din anul 2009, Ryan Dahl a prezentat un proiect personal, o platforma care combină motorul dezvoltat de echipa *The Chromium Project* din cadrul *Google*, *V8* cu un *API I/O*. Cu ajutorul simplității oferite de către *JavaScript*, proiectul

denumit *Node.JS* transformă problema scrierii aplicațiilor de *back-end* în *browser* într-o sarcină ușoară. [20]

*Node.JS* a avut un feedback pozitiv încă din primele versiuni din mai multe motive. Primul din aceste motive este limbajul de programare pe care este bazat, ci anume *JavaScript*, cel mai folosit limbaj de programare *web* de pe glob, iar majoritatea programatorilor de aplicații știu să îl folosească sau au scris cel puțin o dată în *browser* folosind acest limbaj.

Un alt motiv este simplitatea sa, deoarece funcționalitățile de bază ale acestuia sunt ținute la minim, fiind mereu loc de extensie al acestuia. El vine alături de *npm* (*node package manager*), cel mai mare manager de pachete, unde se pot găsi peste 1.3 milioane de module, așa că, pentru construirea unei aplicații mai complexe se pot folosi cu ușurință unul din acestea.

O aplicație *Node.JS* rulează pe un singur proces, furnizând o serie de primitive asincrone care nu lasă codul de *JavaScript* să se blocheze, astfel încât în momentul unei operații, cum ar fi citirea din baza de date sau o operație *HTTP*, aceasta este lăsată să ruleze, iar restul aplicației își continuă fluxul normal și se va întoarce la operația anterioară abia atunci când știe că aceasta s-a efectuat și a primit un răspuns. [21]

## MySQL

O bază de date relațională este un instrument esențial în foarte multe domenii, de la contexte educaționale, *research*, business până la aplicații *web*. La început, bazele de date erau considerate un „lux” pentru destul de multe companii, întrucât aveau niște cerințe la nivel hardware foarte mari pentru a avea o performanță mediocră. [22]

Situația s-a schimbat în ultimii 10 ani, atât pe partea hardware cât și pe cea software, deoarece costurile au scăzut de câteva ori față de începuturi, iar performanța a crescut semnificativ. Alte lucruri care au devenit mult mai accesibile sunt aplicațiile care se ocupa de baze de date. Unul din acestea este *MySQL*, un sistem *SQL* (*Structured Query Language*) relațional de management al bazelor de date, originar din Scandinavia, cu prima versiune realizată în anul 1979. El include un *server SQL*, programe administrative și o interfață de *admin*. [22]

Principalele motive pentru care am hotărât că *MySQL* este potrivit pentru aplicația mea sunt următoarele:

- **Viteza:** Dezvoltatorii sistemului consideră ca *MySQL* este cel mai rapid sistem de baze de date prezent pe piață
- **Portabilitatea:** *MySQL* rulează pe o varietate de sisteme de operare, atât pe *UNIX* cât și pe *Windows*, rulând cu succes de la mașini de mici dimensiuni (calculatoare personale) până la servere de ultimă generație
- **Dimensiunea redusă:** Are o dimensiune foarte mică comparativă cu alte sisteme

- **Capabilitate:** Rulează mai multe *thread-uri*, pentru ca multiplii clienți se pot *loga* simultan, folosind bazele de date generate în același timp. Se poate interacționa cu baza de date prin o serie de interfețe cum ar fi *command-line*, *browsers* sau interfețe vizuale.

## Sequelize

Pentru a putea avea un management eficient al datelor, am luat decizia de a folosi MySQL alături de cel mai popular *ORM (Object-Relational Mapping)* bazat pe *Node.JS*, ci anume *Sequelize*, avantajul sau principal este faptul că se bazează pe „promisiuni” și oferă un acces asincron și rapid către baze de date cum ar fi *Postgres*, *MySQL*, *MariaDB*, *SQLite* și *Microsoft SQL Server*.

## Passport.JS

Securitatea datelor utilizatorului este un subiect foarte important care ar trebui abordat de fiecare programator în 2021, de aceea am hotărât să am în vedere acest aspect în dezvoltarea aplicației mele.

*Passport* este un *middleware* folosit pentru autentificarea în cadrul aplicațiilor *Node*. Are o singură funcționalitate, aceea de a se ocupa de cererile de autentificare are fiecărui utilizator în momentul logării acestuia. Autentificarea în cadrul aplicațiilor web moderne poate avea mai multe forme. În mod tradițional, utilizatorii introduc un *username* și o parolă, dar acest lucru nu este extrem de sigur, de aceea *Passport* introduce credențiale bazate pe *token-uri* și criptare a parolelor în baza de date ca, în cazul compromiterii acestora sau a unui administrator rău intenționat, nimeni în afara de persoana care a creat contul nu va avea acces la parola acestuia. [23]

## 4.2 Implementarea aplicației

### 4.2.1 Implementarea modului de *back-end*

Modulul de *back-end* a fost realizat în mediul de rulare *Node.js* prezentat anterior, iar primii pași pentru crearea serverului sunt inițializarea mediului prin comanda *npm init* și inițializarea dependențelor necesare acestuia și legarea de router. Prin definirea unui port și prin apelarea funcției *app.listen* care primește ca parametri portul și, opțional, o funcție ce ne va confirma că serverul rulează.

```

app.use(bodyParser.json());
app.use(cors(corsOptions));
app.use(passport.initialize());
app.use(passport.session());
app.use(function (req : Request<ParamsDictionary, any, any, ParsedQs, Record<string, any>>, res : Response<any, Record<string, any>>, next : NextFunction ) {
  res.header( field: "Access-Control-Allow-Origin", value: "http://localhost:3000");
  res.header(
    field: "Access-Control-Allow-Headers",
    value: "Origin, X-Requested-With, Content-Type, Accept"
  );
  res.header( field: "Access-Control-Allow-Methods", value: "GET,DELETE,PUT,POST");
  res.header( field: "Access-Control-Allow-Credentials", value: "true");
  next();
});
app.use("/", router);

app.listen(port, callback: () => console.log("App is running on port " + port));

```

Figura 25. Setările inițiale ale serverului

În figura anterioară se poate observa că sunt folosite o serie de setări, prima dintre ele fiind *bodyParser*, un *middleware* care primește întreg corpul cererilor și le transpune într-un fișier de tip *JSON* în cadrul *req.body*. Pentru a putea accesa din pagină rezultatele din partea de *back-end*, este nevoie să ne folosim de *CORS* (*Cross-Origin Resource Sharing*), dând aplicației permisiunea de a trimite și primi răspunsuri de la pagina noastră web.

Baza de date este una relațională, dezvoltată în *MySQL* cu ajutorul *Sequelize*, un *ORM* ce oferă un acces asincron și rapid către date, fiind bazat pe promisiuni. Pentru a crea tabelele în mediul nostru de lucru, ele trebuie definite ca și modele după o anumită structură.

```

const Appointment = sequelize.define('appointment', {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
    allowNull: false
  },
  date: DataTypes.STRING,
  duration: DataTypes.INTEGER,
  hour: DataTypes.STRING,
  option: DataTypes.STRING,
  price: DataTypes.INTEGER,
  details: DataTypes.STRING,
  status: DataTypes.INTEGER,
  carId: DataTypes.INTEGER,
  userId: DataTypes.INTEGER,
  serviceId: DataTypes.INTEGER
});

```

Figura 26. Definirea modelului Appointment

În figura atașată anterior, este prezentată definirea modelului *Appointment*, unde este specificat fiecare coloană a tabeli, alături de tipul de dată aferent, acesta fiind o proprietate a obiectului *DataTypes* din cadrul *Sequelize*, fiind foarte asemănătoare ca și nume cu cele din *SQL*, cu câteva excepții (*String=Varchar*). Pentru a putea integra o cheie primară trebuie să setăm câmpul *primaryKey* la valoarea *true*, fiind recomandată și activarea câmpurilor *autoIncrement* și *allowNull*.



Legătura dintre tabele se realizează prin asocierea acestora între ele cu ajutorul funcțiilor *belongsTo*, *hasOne*, *hasMany*, acestea primind ca parametri tabelă către care se va face referința și un obiect ce este reprezentat de opțiunile dorite. Un exemplu de legătura este prezentat mai jos, unde tabela mașină este referențiată către utilizator, are mai multe programări și tractări.

```
Car.associate = models => {
  Car.belongsTo(models.user, options: {
    as: 'user',
    foreignKey: "userId"
  });
  Car.hasMany(models.appointment, options: {
    onDelete: "cascade"
  });
  Car.hasMany(models.towing, options: {
    onDelete: "cascade"
  });
}
```

Figura 27. Definirea legăturilor din cadrul tabelii Car

Rutarea este reprezentată de comportamentul *endpoint-urilor* aplicației în momentul primirii sau trimiterii unei cereri de la sau către client. În contextul nostru, avem nevoie de cel puțin unul din fiecare dintre verbele *HTTP* (*Post*, *Put*, *Get*, *Delete*). O rută foarte importantă este cea care ne resetează baza de date, deoarece dacă realizăm orice modificare a coloanelor tabelilor, vom da de o eroare dacă continuăm să lucrăm pe structura veche. De notat este faptul că aceasta rută nu trebuie să fie accesibilă în momentul în care aplicația iese din stadiul de dezvoltare deoarece oricine poate o poate apela și ne va face să ne pierdem toate datele din baza noastră de date.

Mai jos este prezentată structura rutelor ce afectează utilizatorul, ele fiind apelate prin router urmat de tipul de *endpoint* pe care dorim să îl apelăm, având ca parametri *URL-ul* și funcția care va fi apelată în momentul în care se ajunge pe acesta.

```
router.post( path: '/user/add', userController.addUser);
router.post( path: '/login', checkAuth, passport.authenticate( strategy: "local", options: {
  successRedirect: "/success",
  failureRedirect: "/fail"
})))
router.delete( path: '/user/:id', userController.deleteOneUser);
router.get( path: '/user/:id', userController.getOneUser);
router.get( path: '/user/', userController.getAllUsers);
router.put( path: '/user/:id', userController.updateUser);
```

Figura 28. Definirea rutelor utilizatorului

Aceste funcții sunt importate dintr-un controller specifică modelului care urmează să fie influențat. În definiție, un controller este o funcție de tip *callback* care corespunde unui anumit router pentru a trata cererile din partea utilizatorului, făcând parte din *design pattern-ul MVC* (*Model View Controller*), fiind folosit pentru a păstra codul cât mai ușor de citit și concis.

```

updateUser: async(req,res) => {
  let userId=req.params['id'];
  const user = await UserDB.findOne({ where: {id:userId}});
  let hashedPass;
  if (req.body.password !== null && req.body.password !== undefined) {
    let password = req.body.password;
    hashedPass = await bcrypt.hash(password, salt: 10);
  } else {
    hashedPass = user.password;
  }
  user.update({
    firstName:req.body.firstName,
    lastName:req.body.lastName,
    email:req.body.email,
    password: hashedPass
  })
  .then(() => {
    res.status(200).send({message:"User successfully updated"})
  })
  .catch(() => {
    res.status(500).send({message:"Server error"})
  })
},

```

Figura 29. Definirea funcției updateUser

În cadrul controller-ului de mai sus este prezentată o cerere de tip *PUT*, ce primește ca parametru un obiect care reprezintă câmpurile ce vor fi modificate în cadrul bazei de date. Întrucât confidențialitatea este extrem de importantă în ziua de azi, în momentul în care parola este modificată, aceasta nu este trimisă în forma ei brută, ci este criptată cu ajutorul unui modul auxiliar, *bcrypt* care folosește algoritmul de criptare *SHA-256*, el primind ca parametri parola ce trebuie criptată și o valoare, *cost*, care reprezintă numărul de apeluri ale criptării, în cazul nostru se apelează de 10 ori.

Pentru a se putea face referință la userul corect, este folosită funcția *findOne* din cadrul *Sequelize*, care găsește prima apariție a unei intrări din cadrul tabelii în funcție de parametri care sunt oferiți în proprietatea *where*.

```

getAllCarsOfUser: async(req, res) => {
  try{
    const currentUser = await req.user;
    let cars = await CarModel.findAll( options: {
      where : {
        userId: currentUser.id
      }
    });
    res.status(200).send(cars);
  } catch(err){
    res.status(500).send({
      message: "Error selecting all cars of user " + currentUser
    })
  }
},

```

Figura 30. Definirea funcției getAllCarsOfUser

În figura 30 se află o cerere de tip *GET* care va trimite către utilizator un obiect ce conține toate mașinile utilizatorului din sesiunea activă. În constanta *currentUser* se regăsesc datele utilizatorului logat, acestea venind dintr-o sesiune din cadrul *passport* care le serializează

în momentul logării și îl deserializează în momentul chemării acestuia. Vectorul de obiecte este returnat cu ajutorul funcției *findAll* apelată cu clauza *where* și id-ul curent.

Pentru a adăuga o nouă programare, vom folosi funcția *addAppointment* prezentată în figura de mai jos. Obiectul primit ca parametru conține aproape toate câmpurile necesare și pentru a atribui programarea utilizatorului curent, îl vom prelua din cadrul *passport*. Dacă toate câmpurile trec de validări, le vom introduce în sistem prin funcția *create*, apelată cu obiectul creat de noi.

```
addAppointment: async(req,res) => {
  const currentUser = await req.user;
  const appointment = {
    date: req.body.date,
    duration: req.body.duration,
    hour: req.body.hour,
    option: req.body.option,
    price: req.body.price,
    details: req.body.details,
    status: req.body.status,
    carId: req.body.carId,
    userId: currentUser.id,
    serviceId: req.body.serviceId
  }

  let err=false;

  if(validate()) {
    err=true;
  }

  if(!err){
    try{
      try{
        await AppointmentDB.create(appointment);
        res.status(200).send("Appointment added.")
      } catch(err){
        res.status(500).send(err)
      }
    } catch(err){
      res.status(500).send(err)
    }
  }
},
```

Figura 31. Definirea funcției *addAppointment*

Ultimul tip de cerere prezentă în aplicație este cea de ștergere. În figura 31 este prezentată funcția care face acest lucru. Ștergerea din baza de date este realizată cu ajutorul funcției *destroy* ce este apelată cu clauza *where* și id-ul fiind primit în parametri.

```
deleteOneTowing: async(req, res) => {
  try{
    let towingId = req.params['id'];
    const car = await TowingDB.destroy({
      where: {
        id: towingId
      }
    })
    res.status(200).send({
      message: "Towing " + towingId + " deleted."
    });
  }catch(error){
    res.status(500).send({
      message: "Error deleting towing!"
    })
  }
},
```

Figura 32. Definirea funcției *deleteOneTowing*

#### 4.2.2 Implementarea modului de *front-end*

Modulul de *front-end* a fost realizat în *ReactJS*, o librărie de *JavaScript* realizată de *Facebook*, primii pași pentru instanțierea ei sunt prin folosirea comenzii predefinite *npx create-react-app*. Așa cum este și în definiție, *React* se bazează în mare parte pe componente, iar acestea sunt de două tipuri: *class-based* și *function-based*. Diferența dintre cele două este, în primul rând, sintaxa, întrucât prima categorie este o simplă clasă ce extinde *React* iar ce-a de-a doua este o funcție ce returnează conținut de tip *JSX*.

În al doilea rând, în cadrul *function-based* nu exista termenul de stare al aplicației, și este introdus printr-o funcționalitate introdusă în versiunea 16.8.0, și anume *React Hooks*, iar în cadrul *class-based* exista atât termenul de stare, cât și metode specifice acestuia.

Am hotărât să folosesc al doilea tip este pentru că este folosit mult mai mult în industrie, codul este mult mai ușor de citit și sunt evitate foarte multe probleme.

Pentru a putea păstra persistența unui utilizator logat pe toată perioadă activării sale în cadrul aplicației, am folosit sistem de state management, anume *Redux*. Toate datele necesare utilizatorului vor trebui stocate într-un *store* ce va conține unul sau mai multe reduceri. Un *reducer* este reprezentat de o suită de acțiuni. În cadrul figurii de mai jos este prezentat un *userReducer* ce conține acțiunile de *login* și *logout*, în care sunt salvate respectiv șterse datele.

```
export const userSlice = createSlice( options: {
  name: "user",
  initialState: {
    user: null,
  },
  reducers: {
    login: (state : Draft<State> , action : PayloadAction<any> ) => {
      state.user = action.payload;
    },
    logout: (state : Draft<State> ) => {
      state.user = null;
    },
  },
});
```

Figura 33. Definirea funcțiilor de tip *reducer*

Un alt lucru foarte important în cadrul aplicației este rutarea paginilor. Această funcționalitate va fi implementată cu ajutorul *react-router-dom*, un pachet ce introduce o serie de componente care, dacă respectă anumite condiții, redirecționează utilizatorul către pagina cu pricina. În figura de mai jos este prezentat un *Switch*, o componenta care va randa prima rută din lista al cărui nume se potrivește cu calea dată de utilizator. În interiorul ei se regăsesc mai multe *ProtectedRoutes*, o rută personalizată care verifică, cu ajutorul datelor din *store* dacă utilizatorul este logat sau nu. În caz afirmativ, este trimis către pagina pe care a accesat-o iar în caz contrar este redirecționat către pagina care îi permite să se autentifice.

```

<Switch>
  <Route exact path="/login" render={props => <Login {...props}/>}/>
  <ProtectedRoute exact path="/" component={Homepage}/>
  <ProtectedRoute exact path="/profile" component={Profile}/>
  <ProtectedRoute exact path="/appointment" component={Appointment}/>
  <ProtectedRoute exact path="/services" component={Profile}/>
  <ProtectedRoute exact path="/tow" component={Tow}/>
</Switch>

```

Figura 34. Definirea rutării și a rutelor protejate

În pagina principală este prezent un formular de feedback care, în momentul trimerii, îi va trimite un email administratorului aplicației cu tot ce a introdus utilizatorul. Această funcționalitate a fost realizată cu ajutorul *emailjs*, un pachet care intermedia transmiterea mesajului între receptor și emițător. În figura 32 este prezentată funcția care trimite mailul, ea fiind apelată în momentul în care formularul este completat și trimis. Are 4 parametri obligatorii : serviceID, templateID, event, userID.

ServiceID-ul este cheia aferentă fiecărui serviciu de mail activat în cadrul contului, templateID este cheia către *template-ul* folosit ca răspuns, event conține toate datele din interiorul formularului iar userID-ul este constanta prin care este verificat administratorul.

```

const SendEmail = (e) => {
  e.preventDefault();
  emailjs.sendForm( serviceID: 'service_iso0znn', templateID: 'template_wucj0lm', e.target, userID: 'user_7fqXS2qIazaRWIIBg8La')
    .then((result : EmailJSResponseStatus) => {
      console.log(result.text);
    }, (error) => {
      console.log(error.text);
    });
  e.target.reset();
}

```

Figura 35. Funcția care trimite emailul prin emailjs

Pentru a avea un design cât mai curat și fluent, am folosit librăria de componente *Material-UI*. Sistemul de tip *Grid* este una din cele mai folosite funcționalități, întrucât cu ajutorul ei se pot crea pagini *responsive* foarte rapid, primind ca parametri *xs,sm,md,lg*, un set de puncte în care pagina se va schimba, ea fiind împărțită în 12 părți. Spre exemplu, dacă dorim ca o componentă să ocupe jumătate de pagina pe toate ecranele mai mari de 500px, vom folosi proprietatea *xs={6}*, iar restul jumătății o putem împărți după bunul plac. Un alt avantaj din *Material-UI* este tema, un obiect *customizabil* unde putem seta propriile culori, fonturi dimensiuni de text și multe altele. În cadrul componentelor putem defini o multitudine de opțiuni ale acestora, de la tipul componentei (*outlined, header*) până la valori și la comportament în cazul unei schimbări de text sau a unui click, așa cum se poate observa în Anexa 1.

Întrucât aplicația mea are legătură cu activități realizate fizic într-un loc anume, pentru a ușura experiența utilizatorului, am folosit un serviciu *Google*, și anume *Google Maps*. Pentru

o integrare mai ușoară din punct de vedere al dezvoltării am folosit un pachet, *google-maps-react*, care ne oferă un set de componente (*Map*, *Marker*, *InfoWindow*) ajutătoare.

```
<Map
  className={"google-maps-container"}
  google={this.props.google}
  initialCenter={{
    lat: this.state.mapCenter.lat,
    lng: this.state.mapCenter.lng,
  }}
  style={{
    width: 400,
    height: 400,
    position: "relative",
    borderRadius: "15px",
  }}
>
  {this.props.services.map((mark, index) => {
    const address = mark.name + "\n" + mark.address;
    return (
      <Marker
        key={index}
        onClick={this.onMarkerClick}
        position={mark.coordinates}
        name={mark.name}
        address={mark.address}
      />
    );
  })}
  <InfoWindow
    marker={this.state.activeMarker}
    visible={this.state.showingInfoWindow}
  >
    <div>
      <h4>{this.state.selectedPlace.name}</h4>
      <h4>{this.state.selectedPlace.address}</h4>
    </div>
  </InfoWindow>
</Map>
```

Figura 36. Implementarea Google Maps API

În figura de mai sus este reprezentată implementarea unei hărți, ce are ca punct central două coordonate setate pe centrul Bucureștiului, și un set de puncte pe hartă, preluate din lista de service-uri prezente în aplicație. În momentul apăsării unuia dintre ele, este afișat un text cu adresa completă și numele service-ului respectiv.

Din același motiv ca acela din paragraful anterior, am decis să folosesc *Google Places*, ca în momentul în care un utilizator trebuie să își cheme o tractare, să poată primi, în timp ce tastează, sugestii legate de străzile prezente în București. Am folosit un alt pachet, pentru ușurință, *react-places-autocomplete* în combinație cu componentele oferite de către Material-UI. În implementarea prezentată în figură, se observă faptul că `<PlacesAutocomplete>` este o componentă de tip *wrapper*, în interiorul căreia am introdus un câmp de introducere a textului. Dacă există sugestii pentru textul care este scris, vor fi afișate în pagină sub forma unor butoane.

```

<PlacesAutocomplete
  value={location}
  onChange={setLocation}
  onSelect={handleSelect}
>
  {({
    getInputProps,
    suggestions,
    getSuggestionItemProps,
    loading,
  }) => (
    <div>
      <TextField
        fullWidth
        variant="outlined"
        label={Address}
        inputProps={{
          ...getInputProps(),
        }}
      />
      <div>
        {loading ? <MenuItem>loading</MenuItem> : null}
        {suggestions.map((suggestion) => {
          const style = {
            backgroundColor: suggestion.active
              ? "#f26438"
              : "#fff",
            color: suggestion.active
              ? "whitesmoke"
              : "black",
            borderRadius: '3px',
            marginTop: '2px',
          };
          return (
            <MenuItem
              {...getSuggestionItemProps(suggestion, { style })}
              key={suggestion.description}
              value={suggestion.description}
            >
              {suggestion.description}
            </MenuItem>
          );
        })}
      </div>
    </div>
  )}

```

Figura 37. Implementarea Google Places

În cadrul paginii destinată doar administratorilor se regăsesc 5 componente, reprezentând fiecare tabelă din baza de date și una generală de tip *dashboard*, ce conține informații statistice despre aplicației. În cea din urmă prezentată sunt prezente, sub forma unor cartonașe, date legate de profit, vânzări totale și costuri, în comparație cu ultima lună, care pot ajuta persoanele care sunt la conducere să își schimbe strategia astfel încât să își maximizeze profitul.

În ce-a de-a doua parte, am implementat 3 reprezentări grafice ce analizează utilizatorii înscriși, câștigurile și numărul de cereri pe fiecare service în parte. Pentru implementare, am folosit *recharts*, un modul din cadrul *node package manager*, ce vine cu grafice interactive care pot fi integrate cu un set de date corespunzător.

```

<ResponsiveContainer width="100%" aspect={4}>
  <LineChart
    width={500}
    height={300}
    data={data}
    margin={{
      top: 20,
      right: 30,
      left: 20,
      bottom: 10,
    }}
  >
    <CartesianGrid strokeDasharray="3 3" />
    <XAxis dataKey="name" height={60} tick={CustomizedAxisTick} />
    <YAxis />
    <Tooltip />
    <Legend />
    <Line
      type="monotone"
      dataKey="pv"
      stroke="#8a8a22"
      name={"2021"}
      label={CustomizedLabel}
    />
    <Line type="monotone" dataKey="uv" stroke="#f26438" name={"2020"} />
  </LineChart>
</ResponsiveContainer>

```

Figura 38. Implementarea line-chart

Din figură se poate observa că, în afară de axe, legendă și un element de tip *tooltip*, sunt prezente două linii, prima din ele fiind pentru anul curent, iar a doua pentru anul trecut.

Al doilea grafic, cel despre analiza câștigurilor pentru fiecare service, este de tip *pie chart* și are în implementare două astfel de grafice, unul deasupra celuilalt, fiecare reprezentând unul din serviciile oferite: programări în service sau tractări.

Ultimul grafic este de tip radar, reprezentând numărul de cereri pentru fiecare service. În el se află două obiecte de tip Radar, unul pentru fiecare tip de serviciu.

```
<RadarChart cx="50%" cy="50%" outerRadius="80%" data={data03}>
  <PolarGrid />
  <PolarAngleAxis dataKey="subject" />
  <PolarRadiusAxis angle={30} domain={[0, 150]} />
  <Radar
    name="Programari"
    dataKey="A"
    stroke="#0a0a22"
    fill="#0a0a22"
    fillOpacity={0.6}
  />
  <Radar
    name="Tractari"
    dataKey="B"
    stroke="#f26438"
    fill="#f26438"
    fillOpacity={0.6}
  />
  <Legend />
  <Tooltip />
</RadarChart>
```

Figura 39. Implementare radar-chart

### 4.3 Prezentarea aplicației

În momentul în care un utilizator intră pentru prima dată în aplicație, acesta este redirecționat către pagina principală, acolo unde poate vedea, în mare, toate funcționalitățile pe care sistemul informatic le oferă. Informațiile regăsite pe această pagină sunt cruciale pentru a putea convinge utilizatorul să își creeze un cont și să intre în ecosistem. Sunt prezentate cele mai importante calități, printre care se enumeră și suportul de 24 de ore pe zi, 7 zile pe săptămână, calitatea produselor și a serviciilor executate de către profesioniști cu experiență în domeniu și diversitatea mărcilor, fiind posibilă efectuarea reparațiilor pe orice model de mașină.

În subsolul paginii se află, alături de o mică istorie a aplicației, un formular de contact, care în momentul completării, trimite un email către administratorii sistemului pentru a putea avea constant păreri, problemele, sugestiile sau informațiile utilizatorilor.





Figura 40. Pagina principală

În momentul logării unui utilizator, acesta primește acces la un meniu de tip *drawer*, prin care poate intra pe una din cele 5 pagini : profil, pagina de start, programări, tractări și service-uri.

În cadrul paginii de programări, pe imaginea de fundal se află două butoane care declanșează două componente de tip modală, una pentru adăugarea unei programări în contul curent iar cealaltă pentru adăugarea unei mașini pentru care se poate apela la serviciile unui service sau a unei tractări.



Figura 41. Pagina de programări

Atunci când se apasă butonul „Fă o programare”, modală aferentă acesteia se deschide, în interiorul ei aflându-se un formular cu 7 câmpuri. Se va selecta data la care utilizatorul dorește să vina în reprezentanță, ora, mașina din cont la care se dorește să se facă anumite operații, se va selecta un service din lista celor disponibile, durata maximă pe care clientul este

dispus să o aștepte, serviciul dorit de acesta și, opțional, alte detalii. În momentul trimerii acestui formular, este verificată corectitudinea datelor, iar dacă acest răspuns este pozitiv, programarea este adăugată în baza de date, fiind vizibilă în contul utilizatorului și în interfața de *admin* iar în caz negativ, îi sunt afișate pe ecran erorile apărute, așteptând o alta instanță corectă.

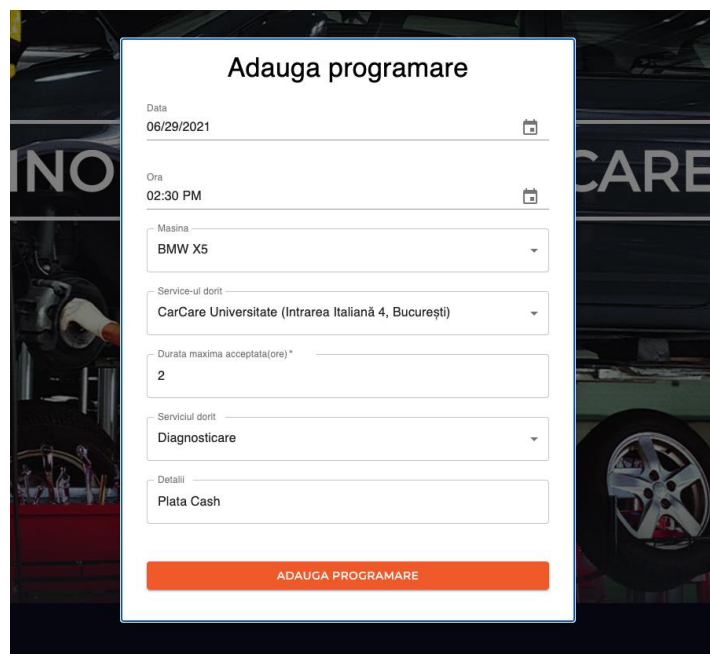


Figura 42. Formular de adăugare programare

Similar cu funcționalitatea prezentată anterior se poate realiza și adăugarea unei mașini, unde vor fi adăugate câmpuri cum ar fi brandul mașinii, marca acesteia, anul fabricației, capacitatea motorului sau tipul de combustibil. Atunci când se trimite formularul, după trecerea validărilor, este adăugat în sistem și este disponibil pentru editare sau ștergere atât din pagina utilizatorului cât și din interfața administratorului.

În momentul accesării paginii de tractări se regăsește o interfață asemănătoare cu cea anterioară, cu o imagine de tip banner pe care se află titlul paginii și un buton denumit sugestiv „cheamă tractare” care va apela funcția care va deschide modala corespondentă adăugării în baza de date a unei cereri de tractare.

Formularul din interior are doar 4 câmpuri, o listă cu mașinile utilizatorului logat, o listă cu serviciile de tractare prezente în cadrul sistemului informatic, un câmp pentru adresă și unul pentru detalii.

În momentul începerii scrierii adresei, utilizatorului îi sunt prezentate, cu ajutorul unei funcționalități din cadrul *Google, Places*, locații a cărui nume are o similitudine mare cu textul introdus.

**Cheama o tractare**

Masina  
Ford Kuga

Tractarea dorita  
CarCare Universitate (Intrarea Italiană 4, București)

Adresa \*  
Strada Cadere

Strada Căderea Bastiliei, Bucharest, Romania  
Intrarea Căderea Bastiliei, Bucharest, Romania  
Hotel Unique, Strada Căderea Bastiliei, Bucharest, Romania  
Facultatea de Management, Academia de Studii Economice, Strada  
Nails Factory, Strada Căderea Bastiliei, Bucharest, Romania

Detalii \*  
Pana roata dreapta

CHEAMA TRACTARE

Figura 43. Formular chemare tractare

Pentru a putea lua o decizie cât mai bună în alegerea locului unde dorește să își ducă mașina, utilizatorul are la dispoziție pagina de prezentare a service-urilor. Aici se regăsește fiecare service prezent în sistem, sub o formă ușor de folosit, unde informațiile se afla la îndemână, pentru ca totul să fie cât mai eficient din punct de vedere al timpului.

Fiecare componentă are aceeași structură, ele fiind prezentate în culorile principale ale temei aplicației, oglindite pentru a putea da un aspect mai plăcut. Primul element observat este harta Google care are ca și punct central adresa service-ului respectiv, aceasta apărând într-un element de tip *tooltip* la click. Următoarele elemente sunt numele unității, adresa acesteia, o descriere succintă care îndeamnă utilizatorii să aleagă acea unitate și o lista neordonată cu serviciile care sunt prestate acolo.

Mai jos sunt 3 elemente, tipul unității, capacitatea acesteia și un buton care trimite către formularul de adăugare al unei programări, cu service-ul de pe care s-a apelat deja pre populat.

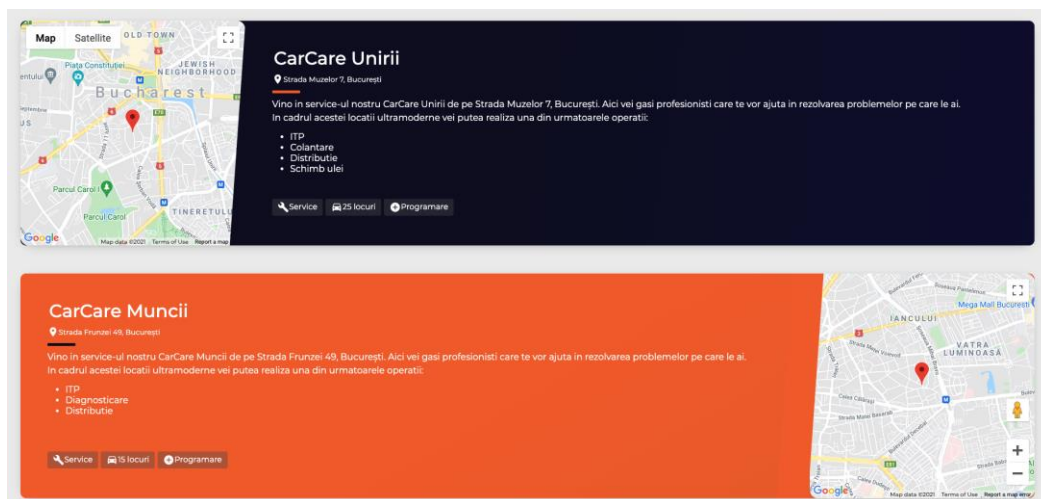


Figura 44. Pagina service-uri

Unul din scopurile principale ale aplicației este acela de management cât mai eficient al acțiunilor utilizatorului, iar pentru a putea îndeplini acest obiectiv a fost creată pagina de profil. Această pagină este împărțită pe verticală în două secțiuni. Prima din ele, cu conținut static, prezintă datele utilizatorului prezent (nume, prenume, email) și un buton ce permite editarea datelor acestuia, în mare parte folosit pentru cazul în care își schimbă adresa de email.

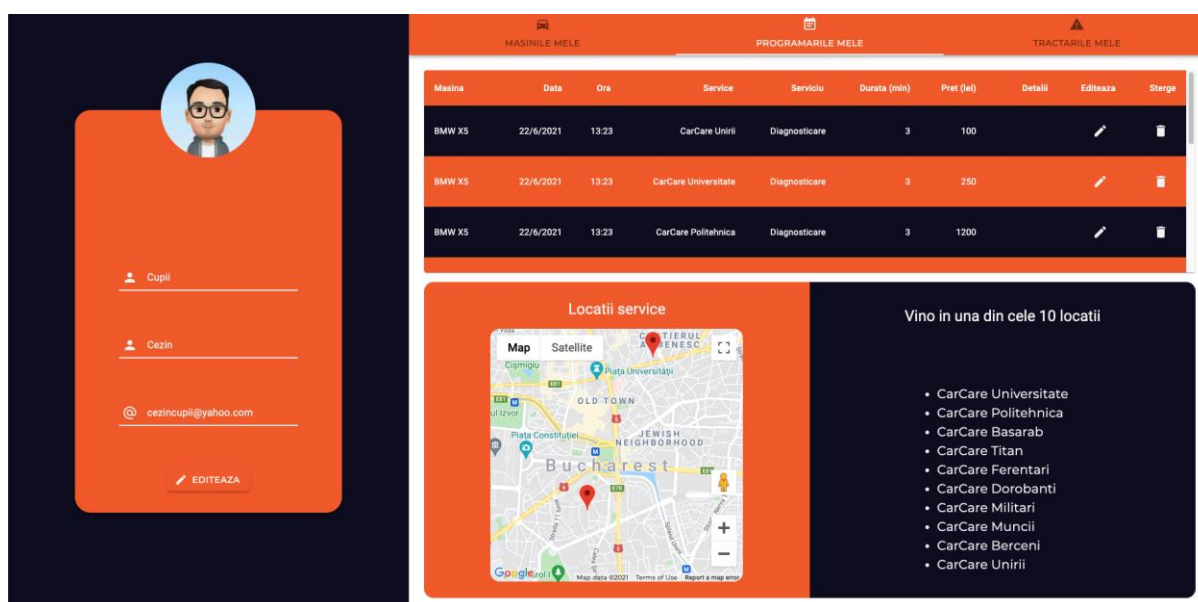


Figura 45. Pagina profil

Cea de-a doua secțiune este alcătuită din trei file (Mașinile mele, Programările mele, Tractările mele), fiecare dintre ele având în interior date relevante. În cadrul mașinilor, este prezentat un tabel ce conține, pe fiecare rând, câte un autovehicul adăugat de către utilizator și detaliile legate de acesta (brand, model, an fabricație, capacitate cilindrică, combustibil).

În cadrul programărilor sunt prezente coloanele unei programări iar în plus comparativ cu cele pe care le adaugă utilizatorul este prețul, adăugat de administrator în momentul în care acceptă programarea. În cazul tractărilor, pe lângă mașină, numele service-ului și locația, este adăugat și câmpul ce comunică prețul pe kilometru.

Fiecare rând din cele 3 tipuri de tabele are în componența sa două butoane, unul ce permite editarea intrării respective, ce va fi modificată atât din interfață cât și din baza de date a sistemului și, analog, unul ce va șterge acea apariție din cele două locuri prezentate în paragraful anterior.

În plus, pe fiecare filă se află o hartă interactivă din cadrul Google ce va face navigarea către acestea mult mai ușoară. În momentul apăsării unuia din cele 10 puncte de pe hartă este afișată adresa completă a acestuia.

Pentru a putea fi o aplicație completă, este nevoie de o **interfață** prin care administratorul poate vedea și administra toate intrările și ieșirile din service.

În cadrul acestei interfețe se regăsesc 6 pagini, una pentru fiecare tabelă prezentă în baza de date și una generală cu date statistice.

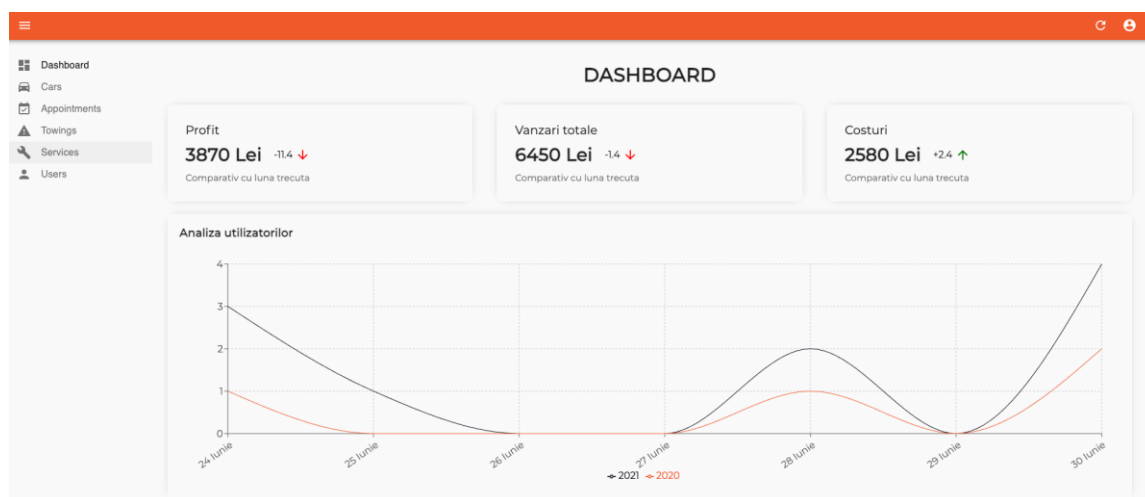


Figura 46. Dashboard admin

Pe prima pagină, cea de *dashboard*, se regăsesc o serie de date statistice. În partea de sus a paginii se regăsesc, pe 3 coloane, diferite sume de bani ce reprezintă profiturile, vânzările totale și costurile, urmate de un procent, fie el în creștere sau în scădere, comparat cu luna anterioară. Graficul din Figura 42 prezintă o analiză a utilizatorilor creați în ultimele 7 zile, comparativ cu aceeași perioadă a anului trecut, fiind un element foarte important pentru a se putea gestiona cât mai bine numărul de oameni înregistrați pe aplicație și, în caz de nevoie, de a se putea lua măsurile necesare.

În figura 43 se poate vedea atât graficul pentru analiza câștigurilor per service, ce arată suma totală câștigată pe fiecare locație în parte, atât pentru programări cât și pentru tractări, cât și graficul pentru numărul de cereri per service ce ne arată numărul total de programări



respectiv tractări din fiecare locație în parte. Aceste două reprezentări pot ajuta la luarea unor decizii importante cum ar fi extinderea capacității unui service sau chiar desființarea acestuia.

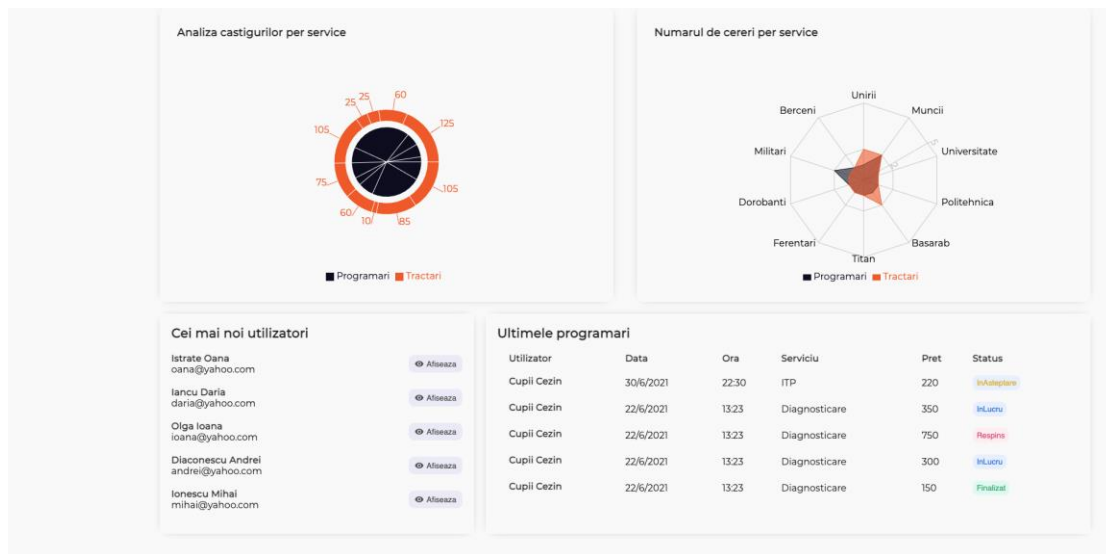


Figura 47. Dashboard admin 2

Ultimele 2 liste sunt prezente pentru a-i oferi administratorului o imagine de ansamblu asupra ultimilor utilizatori înscrși și a ultimelor programări făcute, pentru a putea vedea cât mai are disponibil din punct de vedere al capacității.

Pentru celelalte 5 pagini, structura lor este aceeași, pentru a putea păstra uniformitatea aplicației și pentru a putea avea un flux cât mai lin. Acestea sunt alcătuite din trei componente, o listă, un formular pentru adăugare și unul pentru editare.

Lista este reprezentată printr-un element de tip *DataGrid*, coloanele fiind identice cu cele din cadrul bazei de date, la care se îmbină și butoanele de editare și de ștergere a intrării respective.

ID ↑	Brand	Model	An fabricatie	Capacitate cilindrica	Combustibil	Id detinator		
<input type="checkbox"/>	9	Daewoo	Lanos	2020	1.2	GPL	1	EDIT DELETE
<input type="checkbox"/>	12	BMW	X5	2021	1.2	GPL	3	EDIT DELETE
<input type="checkbox"/>	14	Volkswagen	Golf	2012	1.5	Motorina	1	EDIT DELETE
<input type="checkbox"/>	15	BMW	X5	2018	1.2	Motorina	1	EDIT DELETE
<input type="checkbox"/>	16	BMW	X5	2018	1.2	Motorina	1	EDIT DELETE
<input type="checkbox"/>	17	Volkswagen	Polo	2021	1.2	GPL	3	EDIT DELETE
<input type="checkbox"/>	18	Volkswagen	Polo	2021	1.2	Motorina	1	EDIT DELETE
<input type="checkbox"/>	19	BMW	X5	1998	1.2	Motorina	1	EDIT DELETE
<input type="checkbox"/>	20	Audi	A4	2021	1.9	Benzina	2	EDIT DELETE
<input type="checkbox"/>	21	Ford	Kuga	2020	1.5	Benzina	3	EDIT DELETE

Rows per page: 10 1-10 of 20 1 2 NEXT

Figura 48. Lista mașini

O funcționalitate foarte importantă implementată este exportarea acestor date într-un fișier de tip CSV. Cazul de utilizare cel mai frecvent pentru acest tip de fișiere este

contabilitatea, întrucât se pot face operații, previziuni și se pot lua decizii mult mai ușor cu ajutorul datelor în acest format. Un exemplu al unui astfel de fișier este prezentat în figura 45.

cars (1)

id	brand	model	year	engine	fuel	userId	createdAt	updatedAt
9	Daewoo	Lanos	2020	1.2	GPL	1	2021-06-24T12:58:17.000Z	2021-06-25T12:20:17.000Z
12	BMW	X5	2021	1.2	GPL	3	2021-06-25T11:35:47.000Z	2021-06-25T12:20:53.000Z
14	Volkswagen	Golf	2012	1.5	Motorina	1	2021-06-25T11:47:42.000Z	2021-06-25T11:47:42.000Z
15	BMW	X5	2018	1.2	Motorina	1	2021-06-25T11:48:45.000Z	2021-06-25T11:48:45.000Z
16	BMW	X5	2018	1.2	Motorina	1	2021-06-25T11:57:54.000Z	2021-06-25T11:57:54.000Z
17	Volkswagen	Polo	2021	1.2	GPL	3	2021-06-25T12:07:39.000Z	2021-06-27T21:16:43.000Z
18	Volkswagen	Polo	2021	1.2	Motorina	1	2021-06-25T12:08:40.000Z	2021-06-25T12:08:40.000Z
19	BMW	X5	1998	1.2	Motorina	1	2021-06-25T12:10:09.000Z	2021-06-25T12:10:09.000Z
20	Audi	A4	2021	1.9	Benzina	2	2021-06-25T13:15:19.000Z	2021-06-25T13:15:19.000Z
21	Ford	Kuga	2020	1.5	Benzina	3	2021-06-29T19:59:28.000Z	2021-06-29T19:59:28.000Z

Figura 49. Fișier CSV

## Concluzii

În ziua de azi, din ce în ce mai mulți oameni aleg să folosească aplicațiile și platformele online pentru a-și rezolva diverse probleme, digitalizarea având un impact foarte mare în viețile noastre. Industria auto nu face nici ea excepție de la tendința menționată anterior, astfel, o aplicație pentru gestiunea service-urilor devenind indispensabilă pentru buna funcționare și un management cât mai bun al acestora.

Astfel, administratorii pot atrage mult mai ușor clienții și le pot oferi o gamă variată de servicii printr-o simplă interacțiune cu un sistem informatic, cu o interfață ușor de folosit, de pe orice tip de dispozitiv.

De aceea aplicația CarCare vine atât în ajutorul clienților cât și al administratorilor, oferindu-le o serie de funcționalități menite să îi ajute în rezolvarea problemelor de care se lovesc. Fiecare utilizator își poate adăuga în contul său una sau mai multe mașini având, ulterior, posibilitatea să le programeze la un service din lista celor disponibile. De asemenea, în cazul unui eveniment ce necesită o platformă de tractare, clientul o poate solicita direct din aplicație, specificând adresa la care se află. Prin simpla completare a unui formular ce are în componența sa elemente care ușurează interacțiunea cu sistemul (autocompletarea adresei, liste cu mărcile și modelele mașinilor pre populate) se realizează programarea sau cererea de tractare, clientul urmând, după caz, să se prezinte direct la service, sau să aștepte tractarea nemaifiind nevoie de nici o altă interacțiune cu personalul service-ului.

Din perspectiva administratorului de sistem, acesta va putea vizualiza imaginea de ansamblu a service-urilor sale printr-o serie de statistici, atât analize matematice cât și reprezentări grafice. În plus, va avea acces oricând la datele privind mașinile, programările, tractările, service-urile și utilizatorii, putând fi create, vizualizate, editate, șterse și exportate.

Având în vedere că, în zilele noastre, oamenii sunt din ce în ce mai ocupați și își doresc ca timpul rămas liber să fie alături de cei dragi, o aplicație de acest gen le salvează din timpul petrecut pentru rezolvarea unor probleme de acest tip, toate informațiile aflându-se la îndemână și au parte de o experiență plăcută, astfel clientul se va întoarce și în viitor pentru alte probleme sau va recomanda aplicația și altor oameni aflați în aceeași situație.

Pe viitor, pentru o și mai bună funcționare, ar putea fi adăugată o serie de opțiuni ce vor veni în completarea celor deja existente și care va face interacțiunea utilizatorului cu sistemul mult mai bună și mai eficientă. Prima din ele este integrarea calendarului Google, în momentul în care este adăugată o programare, utilizatorul curent este întrebat dacă dorește să o adauge și în calendarul Google. În caz afirmativ, acesta va fi redirecționat către o pagina de unde își va alege contul în care dorește să se facă acțiunea.

O altă propunere de dezvoltare este trimiterea notificărilor prin SMS, deoarece sunt cele mai vizibile și la îndemână pentru clienți. Din cadrul paginii de profil a fiecăruia dintre ei, va putea bifa sau debifa opțiunea care îl va anunța cu 24 de ore înainte de o programare cu toate detaliile necesare.



În cazul în care administratorul dorește să își extindă aria serviciilor, se poate dezvolta o nouă secțiune pentru oferte de asigurare, unde se pot obține polițe obligatorii, RCA și carte verde pentru a acoperi pagubele produse în afara teritoriului Uniunii Europene.

Omul va avea mereu nevoie de mobilitate iar autovehiculul este și va fi cea mai la îndemână și cea mai folosită modalitate de transport pe toate tipurile de distanțe, mentenanța acestuia fiind mereu necesară pentru buna funcționare al acestui ecosistem al transportului. Clientul are parte de toate aceste posibilități de asigurare a calității autovehiculului personal la un click distanță prin folosirea unei aplicații ce îl pune în contact direct cu service-urile ce se ocupă de aceste aspecte.

## Bibliografie

- [1] M. Verbenkov, „Bax Company,” [Interactiv]. Available: <https://baxcompany.com/insights/why-do-we-feel-that-technological-change-is-happening-too-quickly/>.
- [2] „DigitalCommerce360,” [Interactiv]. Available: <https://www.digitalcommerce360.com/article/us-ecommerce-sales/>.
- [3] „Daimler,” [Interactiv]. Available: <https://www.daimler.com/company/tradition/company-history/1885-1886.html>.
- [4] „DRPCIV,” [Interactiv]. Available: <https://www.drpciv.ro/news-details/statistica/60b74b367d66a112ac8ad28b>.
- [5] „Acea.auto,” 1 Februarie 2021. [Interactiv]. Available: <https://www.acea.auto/figure/average-age-of-eu-vehicle-fleet-by-country/>.
- [6] „Acea.auto,” 1 Ianuarie 2021. [Interactiv]. Available: <https://www.acea.auto/files/report-vehicles-in-use-europe-january-2021-1.pdf>.
- [7] „Acea.auto,” 1 Martie 2021. [Interactiv]. Available: <https://www.acea.auto/figure/share-of-direct-automotive-employment-in-the-eu-by-country/>.
- [8] „Sierra Quadrant,” 29 Octombrie 2020. [Interactiv]. Available: <https://www.sierraquadrant.ro/index.php/sierra-quadrant-este-nevoie-de-o-reglementare-mai-stricta-service-urilor-auto>.
- [9] R. Stan. [Interactiv]. Available: <https://lifenews.ro/2021/06/02/masinile-second-hand-reprezinta-in-acest-an-82-din-piata-auto-iata-cele-mai-cautate-marci/>.
- [10] M. Haverbeke, Eloquent JavaScript: A Modern Introduction to Programming, 2011.
- [11] „Tiobe,” [Interactiv]. Available: <https://www.tiobe.com/tiobe-index/>.
- [12] S. Khan, „General Assembly,” [Interactiv]. Available: <https://generalassemb.ly/blog/what-makes-javascript-so-popular/>.
- [13] Fullstack.io, 30 Days of React.
- [14] L. Ruebelke, AngularJS in action, New York: Manning Shelter Island, 2015.
- [15] C. Macrae, Vue.js: Up and Running.
- [16] M. K. Caspers, Rich Internet Applications w/HTML and Javascript, Oldenburg, 2017.
- [17] „GeeksforGeeks,” 20 Noiembrie 2020. [Interactiv]. Available: <https://www.geeksforgeeks.org/material-ui-introduction-and-installation-for-react/>.
- [18] „Google Maps Platform,” [Interactiv]. Available: <https://developers.google.com/maps/documentation/javascript/overview>.

- [19] „Google Maps Platform,” [Interactiv]. Available:  
<https://developers.google.com/maps/documentation/places/web-service/overview>.
- [20] P. Teixeira, Professional Node.js Building JavaScript-Based Scalable Software, 2013.
- [21] „NodeJS,” [Interactiv]. Available: <https://nodejs.dev/learn>.
- [22] P. DuBois, MySQL Developer's Library.
- [23] „PassportJS,” [Interactiv]. Available: <http://www.passportjs.org/docs/downloads/html/>.

## Anexe

### Listă de figuri

Figura 1. Aplicația Mercedes Me.....	7
Figura 2. Aplicația NETCAR .....	8
Figura 3. Diagrama generală a cazurilor de utilizare .....	10
Figura 4. Diagrama cazului de utilizare programare service .....	11
Figura 5. Diagrama cazului de utilizare administrare mașini .....	12
Figura 6. Diagrama de activitate "Adăugare mașină" .....	13
Figura 7. Diagrama de activitate "Programare service" .....	14
Figura 8. Diagrama de activitate "Vizualizare servicii tractare" .....	14
Figura 9. Diagrama de clase.....	15
Figura 10. Diagrama de stare a unui utilizator.....	15
Figura 11. Diagrama de stare a unei programări.....	16
Figura 12. Diagrama de interacțiune pentru logare .....	16
Figura 13. Diagrama de interacțiune pentru programarea unei mașini.....	17
Figura 14. Diagrama de interacțiune pentru adăugarea unei mașini.....	18
Figura 15. Diagrama de proces (programare mașină).....	18
Figura 16. Diagrama de colaborare (comandare platforma) .....	19
Figura 17. Diagrama claselor detaliată .....	20
Figura 18. Diagrama bazei de date .....	21
Figura 19. IU3 – Pagina principală .....	21
Figura 20. IU4 – Adaugă mașină .....	22
Figura 21. IU5 – Adaugă programare .....	22
Figura 22. Harta de navigare printre interfețe.....	23
Figura 23. Diagrama de componente .....	23
Figura 24. Diagrama de desfășurare .....	24
Figura 25. Setările inițiale ale serverului .....	30
Figura 26. Definirea modelului Appointment.....	30
Figura 27. Definirea legăturilor din cadrul tabelii Car .....	31
Figura 28. Definirea rutelor utilizatorului.....	31
Figura 29. Definirea funcției updateUser.....	32
Figura 30. Definirea funcției getAllCarsOfUser.....	32
Figura 31. Definirea funcției addAppointment.....	33
Figura 32. Definirea funcției deleteOneTowing .....	33
Figura 33. Definirea funcțiilor de tip reducer .....	34
Figura 34. Definirea rutării și a rutelor protejate .....	35
Figura 35. Funcția care trimite emailul prin emailjs.....	35
Figura 36. Implementarea Google Maps API .....	36
Figura 37. Implementarea Google Places .....	37
Figura 38. Implementarea line-chart.....	37
Figura 39. Implementare radar-chart .....	38
Figura 40. Pagina principală .....	39
Figura 41. Pagina de programări.....	39
Figura 42. Formular de adăugare programare.....	40
Figura 43. Formular chemare tractare.....	41
Figura 44. Pagina service-uri .....	42
Figura 45. Pagina profil .....	42
Figura 46. Dashboard admin.....	43

Figura 47. Dashboard admin 2.....	44
Figura 48. Lista mașini .....	44
Figura 49. Fișier CSV .....	45

#### Listă de tabele

Tabel 1. Descrie textuală a cazului de utilizare programare service.....	11
Tabel 2. Descrierea textuală a cazului de utilizare administrare mașini.....	13