MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

AUTOFOCUS - APLICATIE WEB PENTRU GESTIONAREA MAŞINILOR ȘI REPARAȚIILOR DINTR-UN LANȚ DE SERVICEURI AUTO

LUCRARE DE LICENŢĂ

Absolvent: **Alexandru Roman**

Coordonator dr. Ing. Gabriel Cristian Dragomir-Loga
științific:

2022

MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Alexandru Roman**

AUTOFOCUS - APLICATIE WEB PENTRU GESTIONAREA MAȘINILOR ȘI REPARAȚIILOR DINTR-UN LANȚ DE SERVICEURI AUTO

1. **Enunțul temei:** *Crearea unui site web pentru programarea și monitorizarea reparațiilor autovehiculelor de către client care conține și interfața de management pentru admin.*

2. **Conținutul lucrării:** *Introducere, Obiectivele proiectului, Studiu bibliografic, Analiza și fundamentare teoretică Proiectare de detaliu și implementare, Testare si validare, Manual de instalare si utilizare, Concluzii, Bibliografie, Anexă.*

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanți**: dr. Ing. Gabriel Cristian Dragomir-Loga

5. **Data emiterii temei:** 1 noiembrie 2021

6. **Data predării:** 3 iulie 2022

Absolvent:

Coordonator științific:

MINISTERUL EDUCAȚIEI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Declarație pe propria răspundere privind autenticitatea lucrării de licență

Subsemnatul(a) Roman Alexandru legitimat(ă) cu C.I seria AX nr. 688280 CNP 1990923011151, autorul lucrării APLICATIE WEB PENTRU GESTIONAREA MASINILOR SI REPARATIILOR DINTR-UN LANT DE SERVICEURI AUTO elaborată în vederea susținerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea CALCULATOARE din cadrul Universității Tehnice din Cluj-Napoca, sesiunea IULIE 2022 a anului universitar 2021 - 2022, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, pe baza cercetărilor mele și pe baza informațiilor obținute din surse care au fost citate, în textul lucrării și în bibliografie.

Declar că această lucrare nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în fața unei alte comisii de examen de licență.

In cazul constatării ulterioare a unor declarații false, voi suporta sancțiunile administrative, respectiv, *anularea examenului de licență*.

Data	Nume, Prenume
03.07.2022	Roman Alexandru

Semnătura

Cuprins

Capitolul 1. Introducere

1.1. Contextul proiectului

Evoluția este procesul prin care ne adaptăm la schimbările din mediul în care trăim. Este natura noastră ca oameni să descoperim și să evoluăm în orice mod posibil, iar acest lucru nu a făcut decât să îmbunătățească condițiile de viată a oamenilor și să ne beneficieze societatea. Una dintre cele mai importante descoperiri ale ultimului secol este știința calculatoarelor și desigur, internetul. Internetul, putem zice ca a apărut, în anii 80`, odată cu crearea arhitecturii precum Domain Name System și declararea TCP/IP ca fiind un standard de networking [1]. Acesta a revoluționat atât modul în care comunicăm și interacționam între noi, pentru totdeauna, indiferent de locația geografică, cât și modul în care ne documentăm si căutam informații și oportunități. Un website este un tip de pagini de tip World Wide Web, care conțin hyperlink-uri, una către cealaltă, și au scopul de a oferi posibilitatea distribuirii informațiilor. Web-ul, în ziua de astăzi, este un adevărat univers în sinea sa, în care poți găsi orice informație. Tehnologiile web au evoluat dea lungul anilor, și au început să ofere web designer-ilor capacitatea de a crea experiențe, animații și caracteristici noi, utilizatorilor care le accesează. Putem spune ca oferă o experientă îmbunătățită atât utilizatorilor site-ului cât și web developerului, prin utilizarea a noi componente și caracteristici care îi ușurează munca acestuia. Experiența web pe care o vedem în ziua de astăzi este adusă de către efortul unei comunități mari de programatori care asistă la crearea ultimelor tehnologii web. Unele dintre acestea, care merita menționate sunt: css 3, html 5, React JS, Angular, php, etc. Aceste tehnologii asigură de asemenea compatibilitatea dintre aplicația web și browser [2].

1.2. Motivația

Trăind într-o perioadă în care toate companiile optează pentru a-și dezvolta prezența digitală, aceasta devine din ce în ce mai importantă pentru dezvoltarea unei afaceri. Clienții, în căutarea diferitor produse sau servicii, apelează adesea la internet pentru a găsi cele mai bune opțiuni, motiv pentru care site-urile web măresc adesea cifra de afaceri a oricărui firme. Prezența online, este un factor care influențează numărul de clienți noi aduși către o companie sau firmă, prin publicarea informațiilor online, fiind de asemenea un mod de publicitate. Aplicația dezvoltată are niște funcționalități specifice, dar având o prezența online, clienții pot accesa cu ușurință informațiile distribuite, acest lucru făcând mai ușoară posibila alegere pentru un nou service auto. Un site web de asemenea este un semn care indică seriozitatea unui service, sau a oricărui tip de companii, deoarece scoate in evidență intenția și seriozitatea cu care operează aceasta.

Industria automobilistică, este o industrie care încă investește sume enorme de bani atât în producție, cât și in cercetare. Conform unui studiu din anul 2021 [3], se prezice creșterea CAGR(Compound Annual Growth Rate) cu 3% pentru următorii 5 ani, adică până în anul 2026. Mașinile devin tot mai complicate atât din punct de vedere mecanic, cât și din punct de vedere electronic, pentru mărirea performanței și confortului. Acest lucru se numește evoluție, și este un lucru ce aduce un impact pozitiv atât clienților, cât și mediului înconjurător. Cu toate acestea, complexitatea

mașinilor crește, și prin urmare, și complexitatea operațiilor efectuate de către service-uri. O mașină poate sta într-un service pentru o perioadă nelimitată de timp. Acesta este motivul pentru care aplicația web implementată vine alături de o pagină în care se poate supraveghea și monitoriza reparațiile făcute pe aceasta.

Motivul pentru care am ales implementarea unui site web in detrimentul unei aplicații de sine stătătoare, este acela de a face cât mai ușoară accesarea aplicației de pe orice dispozitiv cu acces la un browser web.

1.3. Conținutul lucrării

In acest subcapitol voi prezenta cele 8 capitole, alături de o sumară descriere a fiecăruia, referitor la ceea ce contin.

Capitolul 1. Introducere - această primă secțiune prezintă contextul proiectului, motivul care a dus la implementarea acestui proiect, și pentu care este util in lumea reală. De asemenea contine o scurtă descriere a capitolelor.

Capitolul 2. Obiectivele proiectului – este un capitol important pentru înțelegerea functionării aplicației, conținand și o scurtă prezentare a acesteia. Conține atât obiectivul proiectului cat și obiectivele secundare.

Capitolul 3. Studiul bibliografic – conține un rezumat al informatiilor care țin de partea de cercetare a lucrării. De asemenea conține comparația dintre diferite tipuri de framework-uri și arhitecturi, impreună cu motivul pentru care le-am ales pe fiecare. Prezintă și diferite aplicații cu utilizare în domeniu care pot oferi perspectiva asupra aplicațiilor existente pe piata.

Capitolul 4. Analiză și fundamentare teoretică – acest capitol, conține cerințele funcționale si non-funcționale, tehnologiile folosite, strategii de dezvoltare, cazuri de utilizare și tehnologiile folosite, acestea fiind explicate in amanunt și cu ajutorul diagramelor.

Capitolul 5. Proiectare de detaliu și implementare – este capitolul care explică modul in care a fost proiectată și gandită aplicația inmpreună cu modul in care au fost implementate. Contine arhitectura generală a aplciatiei, nivelul de date, nivelul logic si desigur, cel de prezentare.

Capitolul 6. Testare si validare – este capitolul în care sunt prezentate metodele de testare și validare a implementării proiectului.

Capitolul 7. Manual de instalare utilizare – conține în mod detaliat pașii care trebuie urmați pentru a utiliza aplicația, impreună cu toate functionalitățile acesteia, pentru fiecare tip de utilizator.

Capitolul 8. Concluzii – este prezentat raportul dintre rezultatele așteptate și cele obținute și sunt prezentate posibilele funcționalități ce pot fi implementate pentru a imbunatații aplicația.

Capitolul 2. Obiectivele Proiectului

În acest capitol, sunt prezentate specificația si obiectivul principal, urmate de 2 subcapitole în care se prezintă în mod sumar obiectivele specifice și cele secundare care au dus la formarea proiectului.

2.1. Specificația Proiectului

Scopul acestui proiect este crearea și implementarea unui site web, care are la baza un număr de 3 tipuri de utilizatori, destinat folosirii de către un lanț de service-uri auto pentru a-si ușura comunicarea cu clienții. Funcționalitatea aplicației este de a oferi suport atât pentru programarea unei mașini într-un service ales de către client, cât și urmărirea printr-o interfața user-friendly a progresului reparațiilor făcute mașinii. Aplicația oferă un chat pentru a face posibilă comunicarea problemelor sau întrebărilor întâmpinate în procesul de reparare sau mentenanță a mașinii. Aceasta este orientată pentru clienți și oferă clienților posibilitatea de a intra neautentificați pentru găsirea unui service, într-o locație convenabilă. Odată autentificat, un client beneficiază de restul funcționalităților site-ului.

2.2. Obiectivul principal

Obiectivul principal al acestui proiect este implementarea unui site web care să poată fi folosit pentru a ajuta procesul de management al clienților și lucrărilor acestora, și a eficientiza procesul de programare a fiecărui client în service-uri. Proiectul își propune să ajute procesul de comunicare dintre clienți și service-uri, și să ofere o descriere vizuală și informativă pentru clienți a reparațiilor autovehiculelor aduse în service-uri. Acesta urmărește a oferi o experiență plăcută tuturor utilizatorilor prin intermediul unei interfețe ușor de înțeles, ușor de folosit, care bifează toate opțiunile și ficționalitățile necesare pentru trecerea unui autovehicul printr-o sesiune de reparații în service.

2.3. Obiective specifice

Pe lângă obiectivul principal, prezentat la subcapitolul 2.1, in cadrul acestui proiect se urmăresc următoarele:

Înregistrarea utilizatorilor: clienților li se cere să se autentifice pe website pentru a li se împărții serviciile necesare fiecăruia într-un mod securizat. Utilizatorii neautentificați pot naviga pe site și intra pe paginile service-urilor.

Autentificare: Autentificarea are scopul de a permite fiecărui client să comunice și să intre pe site într-un mod sigur, request-urile făcute de către acesta folosind un jwt token, obținut la autentificare.

Posibilitatea de rezervare loc in service: aceasta se poate realiza, oferind detaliile privind mașina cu probleme și specificațiile acesteia, și diverse observații sau dorințe ale clientului.

Vizualizarea lucrărilor efectuate asupra mașinii: o pagină numita "Your cars" poate fi deschisă după rezervare, in modul client, unde poți vedea statusul vehiculului, lucrările făcute pe acesta și prețul total al lucrărilor, împreună cu diverse detalii.

Comunicarea cu personalul service-ului: un chat este disponibil pentru utilizator, iar acesta poate trimite sau primii mesaje legate de diverse reparații sau lucrări efectuate mașinii.

Configurarea service-urilor in mod admin: în modul admin, ai opțiunea de a configura detaliile service-urilor, și crearea conturilor de tip "Staff" pentru fiecare service, care poate efectua atribuțiile care controlează acțiunile service-urilor.

2.4. Obiective generale

Pentru a atinge toate obiectivele proiectului, acesta trebuie să atingă un anumit obiectiv de calitate. Printre cele mai importante obiective de calitate este **securitatea**. Aceasta asigură navigarea în mod sigur pe website, rutele de navigare fiind separate pentru tipurile de admin sau 'staff" față de cele utilizate de către clienți. Se utilizarea un token de tip jwt care asigură fiecărui sesiuni un timp limitat de utilizare și oferă o comunicare securizată între partea de front-end îs back-end. **Utilizabilitatea** este un aspect important când vina vorba de un website deoarece acesta trebuie să fie ușor de înteles pentru orice tip de clienți.

2.5. Cerințe funcționale

In această secțiune vor fi prezentate cerințele funcționale, pentru a întelege mai bine caracteristicile sitemului ce va fi implementat. Acestea sunt organizate pe tipul de utilizator, inclusiv pentru un utilizator neautentificat.

Utilizator neautentificat

Acțiuni posibile ale unui utilizator neautentificat	
CF 1.1	Utilizatorul poate să creeze un cont nou
CF 1.2	Utilizatorul poate să se autentifice
CF 1.3	Utilizatorul poate să vadă lista de ateliere de reparații

Tabel 2.1. Acțiuni posibile ale unui utilizator neautentificat

Utilizator autentificat

Acțiuni posibile ale unui utilizator autentificat	
CF 2.1	Utilizatorul poate să creeze un cont nou
CF 2.2	Utilizatorul poate să iasă din cont
CF 2.3	Utilizatorul poate să vadă lista de ateliere de reparații
CF 2.4	Utilizatorul poate să facă o rezervare
CF 2.5	Utilizatorul poate vedea propriile mașini și progresul reparației
CF 2.6	Utilizatorul poate sa comunice prin mesagerie cu un reprezentant al atelierului

Tabel 2.2. Acțiuni posibile ale unui utilizator autentificat

Utilizator de tip admin

Actiuni posibile ale unui utilizator de tip admin	
CF 2.1	Adminul poate să creeze un cont nou
CF 2.2	Adminul poate să iasă din cont
CF 2.3	Adminul poate creeze și editeze lista de service-uri
CF 2.4	Adminul să schimbe service-ul curent
CF 2.5	Adminul poate accesa dahboard-ul și schimba limita de rezervări
CF 2.6	Adminul poate crea conturi de tip Staff pentru un service anume
CF 2.7	Adminul poate schimba statusurile mașinilor și vedea ce doreste clientul
CF 2.8	Adminul poate adăuga reparații mașinilor
CF 2.9	Adminul poate comunica prin mesagerie cu clientul

Tabel 2.1. Acțiuni posibile ale unui utilizator de tip admin

Utilizator de tip staff

Acțiuni posibile ale unui utilizator de tip staff	
CF 2.1	Adminul poate să creeze un cont nou
CF 2.2	Adminul poate să iasă din cont
CF 2.3	Adminul poate accesa dahboard-ul și schimba limita de rezervări pentru atelierul la care este repartizat
CF 2.4	Adminul poate schimba statusurile mașinilor si vedea ce doreste clientul
CF 2.5	Adminul poate adăuga reparații mașinilor
CF 2.6	Adminul poate comunica prin mesagerie cu clinetul

Tabel 2.1. Acțiuni posibile ale unui utilizator de tip staff

2.6. Cerințe non-funcționale

În contrast cu cerințele funcționale, care ne arată ce ar trebui un sistem sa facă, se află cerințele non-funcționale, care sunt deseori înțelese ca și atribute de calitate. În realitate, ele reprezintă un criteriu pentru evaluarea a cum sistemul ar trebui să realizeze niște funcționalități. Diferența constă în faptul că un sistem trebuie să dețină anumite atribute de calitate pentru a întruni caracteristicile unei cerințe non-funcționale. Când multe cerințe non-funcționale sunt întâlnite într-un sistem, calitatea acestuia este ridicată, acestea având constrângeri atât asupra arhitecturii sistemului, cât și asupra logicii și tehnologiilor utilizate [4].

2.6.1. Securitate

Securitatea unui sistem este probabil cea mai importantă cerință, iar aceasta presupune protejarea sistemului de către atacatori rău intenționați, păstrând confidențialitatea datelor cu caracter sensibil. Primul pas către filtrarea datelor care ajung la utilizatori, se face prin autentificare. Prin aceasta, se va genera un JWT(Json Web Token), care este un standard folosit pentru a transmite mesaje cu caracter sensibil între un client și un server. Fiecare JWT vine cu set de permisiuni, iar acestea sunt create printr-un algoritm specific. Acesta vine cu un termen de expirare, iar după o anumită durată de timp este necesară din nou autentificarea. Toate rutele care deschid funcționalități restricționate userilor autentificați sunt restricționate.

2.6.2. Uzabilitate

Când ne referim la aplicații web, termenul de uzabilitate se referă la ușurința de folosire a acestuia, de către utilizator, fără a fii necesară alocarea de timp în plus pentru invățarea aplicației. Ambele designul si procesul de development au avut ca și țintă atingerea unui prag mare de accesibilitate. Procesul de design al aplicației a avut

în vedere anumite atribute: simplitate, familiaritate si consistență. Simplitatea și familiaritatea constă în aranjarea generală a paginii, într-un mod simplul, ușor de înțeles și care este folosit într-un mod în care este obișnuit utilizatorul. Aranjarea în pagină este făcută într-un mod în care toate elementele importante se află în locul unde te aștepți, iar culorile folosite sunt alese într-un mod plăcut ochiului și sunt foarte sugestive în concordanță cu elementele pe care sunt plasate.

2.6.3. Performanța

Performanța este un factor cheie când vine vorba de clienți. Când vine vorba de o aplicație web, performanța acesteia este reprezentată de timpul de răspuns oferit către utilizator în raport cu numărul de utilizatori conectați la sistem. Factorul de performantă este dictat de diferite decizii de implementare prezente și în aplicația implementată precum scrierea codului în așa fel încât, componentele din front-end nu se încarcă de mai multe ori. O altă decizie de implementare a aplicației o face parte realizarea call-urilor api de preluare a datelor, către back-end care sunt făcute doar când datele trebuie aduse, nu de mai multe ori.

2.6.4. Extensibilitate

Extensibilitatea este principiul de design al unui sistem care urmărește posiblilitatea codului de a fi extins în viitor și este masurată prin timpul de effort necesar pentru implementarea îmbunătățirilor. Datorită arhitecturii pe layere a componentelor aplicației, aceasta permite implementarea ușoară a noi funcționalități.

2.6.5. Scalabilitate

Scalabilitatea este calitatea sistemului de a permite sistemului să susțină un număr cât mai mare de utilizatori și un număr crescut de apeluri la baza de date. Există două tipuri de scalare și anume cea pe orizontală, care presupune adăugarea a mai multor noduri la sistem, pentru a lucra in paralel, și scalabilitatea pe verticală care presupune adăugarea de resurse la un nod, resurse putând însemna procesoare cpu, memorie, capacitate de stocare, etc. În aplicația ce urmează a fi dezvoltată, scalabilitatea va consta în filtrarea numărului de call-uri la baza de date.

2.6.6. Documentație

Aplicația dezvoltată va dispune de o documentație în format document word, în care va fii prezentată complet atât la nivel de implementare cât și la nivel de prezentare și va dispune de un manual de utilizare.

Capitolul 3. Studiu Bibliografic

3.1. Industria automobilistică

Industria automobilistică a început in anul 1860 când mai mulți producători au creat primul tip de mașina fără cai, statele unite conducând producția de mașini timp de multe decenii. Până in zilele noastre producția de mașini a început să crească iar numărul de mașini din lume a crescut exponențial numărul lor ajungând la aproximativ 1 miliard in 2010 (acest număr include mașini, autobuze si titluri de mici, medii și mari dimensiuni). Un studiu recent făcut in anul 2019 [5] estimează că sunt aproximativ 1.4 miliarde de autovehicule in lume. În figura 3.1 putem vedea producția autovehiculelor pe o perioada de 20 de ani, si observăm că producția anuala a mașinilor a fost în continuă creștere până în anul 2018, când a început brusc să scadă. Studii precum cel prezentat la capitolul 1.3 [3], indică totuși o nouă creștere a industriei în anii ce urmează.

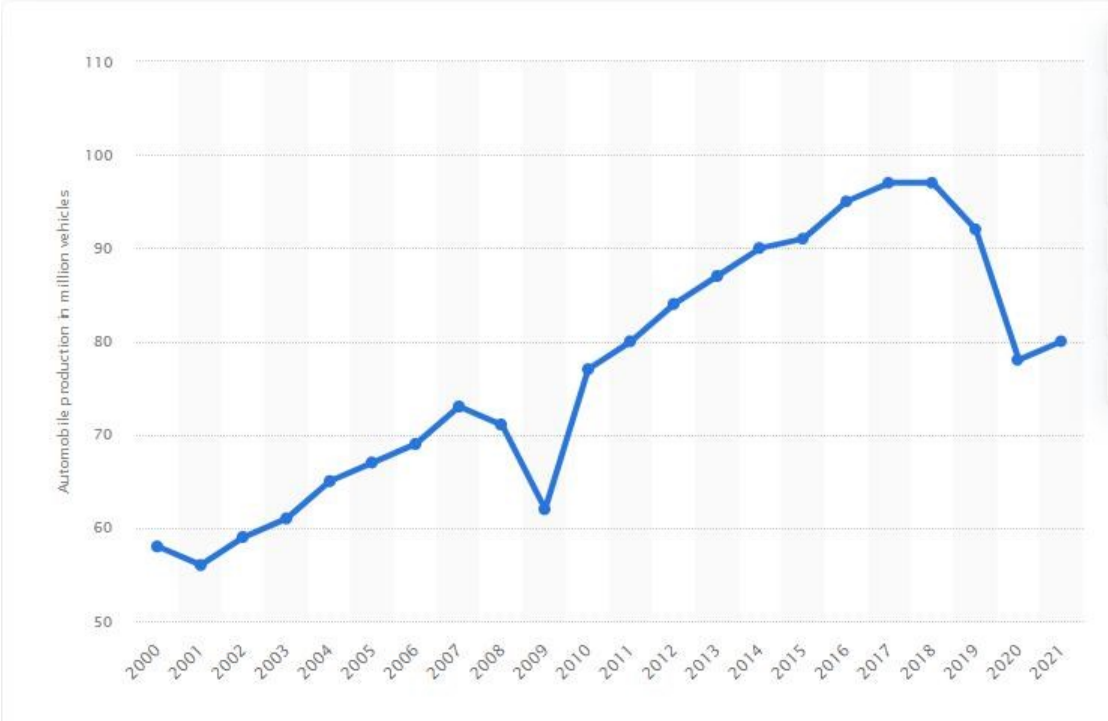

Figura 3.1. producția autovehivulelor în ultimii 20 de ani[1]

După o creștere masivă a producției in anul 2010 in ultimii 3 ani a început sa scadă, numărul de mașini produse fiind un factor de influentă a vechimii medie a mașinilor de pe străzi. Din moment ce vârsta mașinii este un bun indicator al bunei funcționări ale unei mașini, cu cât vechimea medie a mașinilor de pe străzi crește cu atât cerința reparațiilor pentru aceste mașini care vor avea mai multe probleme tehnice va crește direct proporțional.

[1] https://www.statista.com/statistics/262747/worldwide-automobile-production-since-2000/

3.2. Responsabilitățile unui service auto

Un service auto este responsabil pentru a oferi atât consultanță și diagnostic cât și de a oferii servicii de reparații pentru autovehiculele clienților. Fiecare service poate alege să ofere doar anumite reparații sau doar pentru anumite tipuri de mașini, dar, în mod normal un service oferă reparații atât din punct de vedere mecanic, cat si din punct de vedere electric al unui autovehicul rutier care în general sunt mașini, remorci si autovehicule de dimensiuni medii si mari. Sunt de asemenea service-uri specializate pentru vehicule mari, folosite in domeniul construcțiilor sau al agriculturii dar acum ne vom referi la service-urile destinate reparațiilor autovehiculelor rutiere. O descriere buna a serviciilor unui service este ca acestea se împart in două tipuri de servicii.

Primul tip este o serie de proceduri de mentenanță făcute la un anumit interval de timp sau după o anumită distantă parcursă de un vehicul pentru a menține motorul și mașina, in general, în stare bună de funcționare, prin schimbarea periodică a comsumabilelor. Intervalele de service sunt specificate de către producătorul autovehiculului într-o cărticică ce vine odată cu mașina. În unele mașini noi acestea au toate datele in computerul de bord al mașinii care, în general, te avertizează cu o perioadă de timp înainte să programezi o vizită în service pentru o programare.

Al doilea tip de serviciu este reprezentat de procedurile făcute în caz de necesitate pentru repara o mașină în caz că o defecțiune este descoperita. Această defecțiune poate fi descoperită de către client, sau chiar de către service, motiv pentru care o verificare de rutină la un interval fix de timp este recomandat pentru a putea preveni piesele stricate de a produce daune mai mari altor piese. Verificările de rutină sunt și o măsură de securitate pentru a reduce riscul accidentelor pe șosele și a proteja pasagerii la bord. De asemenea, este important pentru orice om să se poată baza pe buna funcționare a mașinii lui în special in cazul călătoriilor pe distante lungi sau în momente importante. Cu cât mașinile sunt mai noi, cu atât au mai mule echipamente și componente sofisticate, dotate cu cea mai nouă tehnologie. Una dintre acestea este o multitudine de senzori care detectează eventuale probleme și te avertizează că este o problemă la un anumit sistem sau o anumita parte a mașinii.

Acestea fiind spuse, putem deduce faptul că toți clienții sunt responsabili pentru reparațiile mașinilor pe care le dețin chiar daca unele mașini fac detecția acestor problemelor mai ușoară. Service-urile auto sunt responsabile pentru repararea mașinilor atunci când dorința reparației mașinii este specificată de către client. Clienții dețin această responsabilitate cu toate că nu toți clienții știu ce este mai bine pentru mașina lor, iar in acest capitol, intră persoanele specializate din service-uri în ajutor. Service-urile auto au oameni specializări care pot detecta problemele unei mașini și sfătui clientul ce reparații trebuie făcute.

3.3. Relația dintre service și client

În orice firmă, relația cu clientul este crucială și strâns legată de bunăstarea financiară a companiei. Construind o relație bună cu acesta, oferindu-i o experiență plăcută, construiește o formă de loialitate, bazată pe încredere. Această loialitate a clienților se construiește prin satisfacția clientului și calitatea serviciilor prestate și creează ceea ce numim clienți fideli [6]. În ziua de azi, gradul de informare devine din

ce in ce mai ridicat, clienții au din ce în ce mai multe opțiuni de unde sa aleagă, lucru care crește influența pe care aceștia o au peste o anumită industrie. Un studiu făcut de către Microsoft [7] arată ca 55% din consumatori/clienți au așteptări mai mari asupra experienței oferite de către firme decât anul precedent.

Aplicația web, AutoFocus țintește bifarea acestei loialități a clienților care își fac rezervare în service, oferindu-le o experiența plăcută, care vine cu o interfață user-friendly și ușor de folosit. Funcționalitatea de rezervări reduce timpul de rezervare în service a clientului evitând, apelurile cu personalul service-ului, Liniile telefonice ocupate și timpul de așteptare la telefon. Aplicația lasă clientul sa facă o rezervare doar in zilele disponibile, lucru care scapă clientul de nevoia de a aștepta o confirmare a rezervării. Dacă acesta nu poate face o simplă rezervare in câțiva pași simpli pentru o simplă rezervare atunci experiența lui nu va fi tocmai plăcută. O altă funcționalitate creată cu scopul de a oferi o reprezentare vizuală și concreta a lucrărilor, către client, este pagina în care poți vedea reparațiile făcute mașinii. Acesta poate accesa pagina mașinilor sale în service și poate vedea statusul oricând. Un chat este inclus în aplicație pentru a servi ca și metoda de comunicare între service și client.

3.4. Aplicații similare

Prostop Canada

Prostop Canada este o platformă de management pentru un atelier de reparații auto făcută pentru a oferi funcționalitate atât pentru operatorii din service cât și pentru clienți. Principala funcționalitate oferită de către aceasta platformă este destinata pentru uz intern, adică, pentru o persoană responsabilă de ordinea activităților din atelier. Platforma este una creată pentru a ajuta la buna funcționare a operațiilor de zi cu zi, cu care se confruntă orice service: operații precum managementul reparațiilor și a întreg service-ului, precum și a rezervărilor clienților. Clienții pot vedea disponibilitatea in timp real și primesc chiar și reminder prin email sau sms pentru a nu își rata rezervarea. Ca și admin ai posibilitatea de a iți customiza parțial aplicația, prin a-ți adauga o imagine de profil, detalii esențiale pentru clienți, și prețuri pentru anumite tipuri de reparații care sunt predefinite. Aceștia au acces și la o pagină de statistici și rapoarte concrete asupra performanței service-ului. Ca și client iți poți customiza rezervarea prin opțiunea de a selecta din start tipul serviciului dorit.

Speedy Auto Service

Speedy Auto Service este un website canadaian care rulează în mod funcțional de 60 de ani, și prestează servicii de reparații pentru autovehicule din Canada, în special în provincia Ontario. Website-ul conține o hartă cu toate service-urile, iar acestea au o pagină care conțin toate detaliile despre service, împreună cu un link către pagina în care faci o rezervare în service. Aceasta pagina difera la diferite service-uri și oferă obținea de a selecta dacă lași mașina în service sau aștepți după reparație, ceea ce este un feature folositor. Site-ul conține și feature-uri folositoare precum un localizator de service-uri în apropiere prin codul poștal, și descrieri detaliate legate de unele reparații posibile. Acesta este orientat pentru folosirea de către utilizator, fără a conține utilități pentru management-ul service-urilor. Un dezavantaj al site-ului este lipsa automatizării rezervărilor, acestea necesitând un răspuns și o confirmare.

Csn Highland

Compania Csn Highland este o firmă cu mai multe centre de reparare a mașinilor care au participat la coliziuni. A fost fondată in 2002, și se ocupă cu repararea tuturor părtilor unei mașini care a avut un accident. Aceștia prestează servicii precum reparații de caroserie, vopsire a vehiculului, reparații parbriz, etc. Firma are un website cu servicii similare cu Speedy Auto Service, continând un localizator pentru atelierele din apropiere. Pe pagina atelierelor au afișate recenzii ale clienților și o opțiune de rezervare automată.

Diferența dintre sistemul dezvoltat și cele de pe piată

Majoritatea aplicațiiloir similare se aseamănă atât prin caracteristici cât si prin modul acestora de rezentare, excepția fiind Prostop Canada care este o întreagă platformă de administrare a unui service. În figura 3.1 sunt prezentate diferențele principale dintre aplicația dezvoltatî și cele studiate.

Caracteristici	Aplicația dezvoltată	Speedy Auto Service	Csn Highland	Prostop Canada
Înregistrare	DA	NU	X	DA
Autentificare	DA	X	X	DA
Funcționalitate administrator	DA	X	X	DA
Funcționalitate client	DA	DA	DA	DA
Funcționalitate pentru ateliere multiple	DA	DA	DA	X
Sistem de rezervare	DA	DA	DA	X
Ofera detalii pentru reparații generale	X	DA	DA	?
Sistem de management	X	X	X	DA

Sistem monitorizare reparații	DA	X	X	DA
Confirmări pe mail/ sms	X	DA	X	DA
Chat	DA	X	DA	DA
Internaționalizare	x	DA	DA	DA

Tabelul 3.1. Diferența dintre sistemul dezvoltat și cele de pe piață

Capitolul 4. Analiză și Fundamentare Teoretică

4.1. Descrierea cazurilor de utilizare

4.1.1. Diagramele cazurilor de utilizare

Modelul cazurilor de utilizare prezintă o colecție de cazuri de utilizare și actori care oferă o descriere generală a modului în care va fi utilizat sistemul de către anumiți utilizatori, furnizează o privire de ansamblu a funcționalităților ce se doresc a fi oferite de sistem, arătând cum interactionează sistemul cu unul sau mai mulți actori.

Utilizatorul neautentificat

Acesta va putea accesa aplicația doar pentru a vedea opțiunile pentru ateliere și va putea să își creeze un cont nou și să se autentifice pentru mai multe funcționalități. Diagrama UML o putem observa in figura 4.1

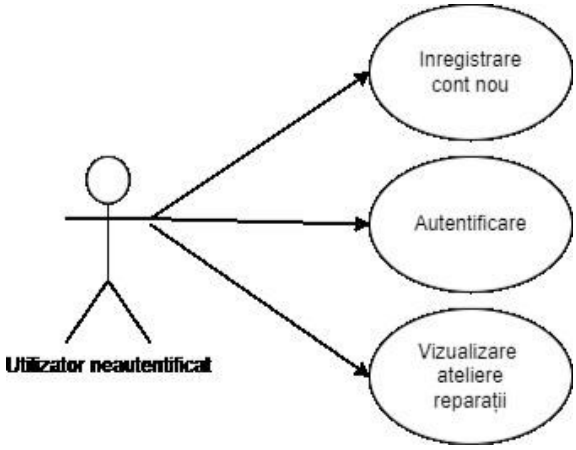

Figura 4.1. Use case-uri utilizator neautentificat

Utilizatorul autentificat

Acesta va putea accesa aplicația atât pentru a vedea opțiunile pentru ateliere cât și pentru a crea o programare pentru un anumit service. Are acces la a vedea statusul mașinilor sale și la mesageria aplicației pentru a lua legatura cu reprezentanții atelierului. Ca și orice tip de utilizator poate sa creeze un cont nou și să sa autentifice. Diagrama UML o putem observa în figura 4.2

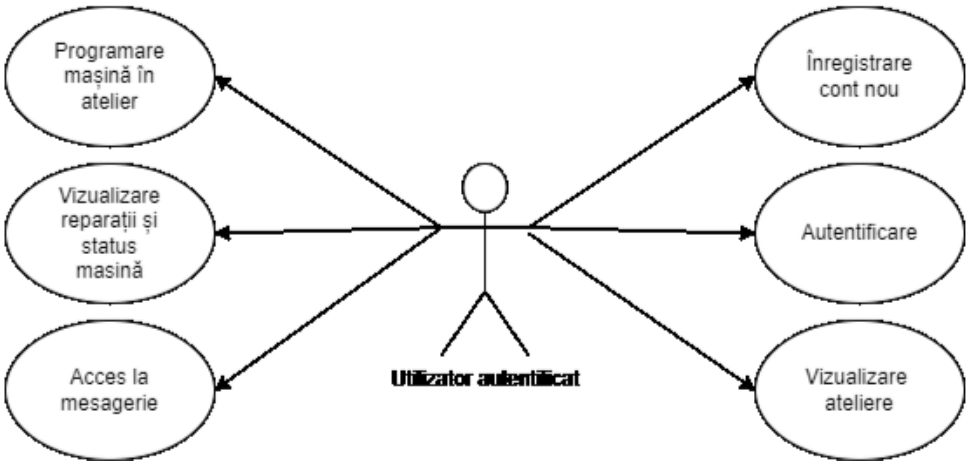

Figura 4.2. Use case-uri utilizator autentificat

Utilizatorul de tip admin

Un utilizator de tip admin pe lângă opțiunile de creare cont nou și autentificare, acesta poate edita complet lista de ateliere, și crea câte un cont de tip staff pentru fiecare. Are acces la dashboard-ul aplicației, unde poate vedea limita de rezervări din fiecare zi și o poate edita pentru orice zi din calendar. Poate intra pe pagina de mașini prezente în service, să editeze atât statusurile fiecăreia cât și lista de reparații de la fiecare mașină, și poate să folosească mesageria. Diagrama UML o putem observa în figura 4.3.

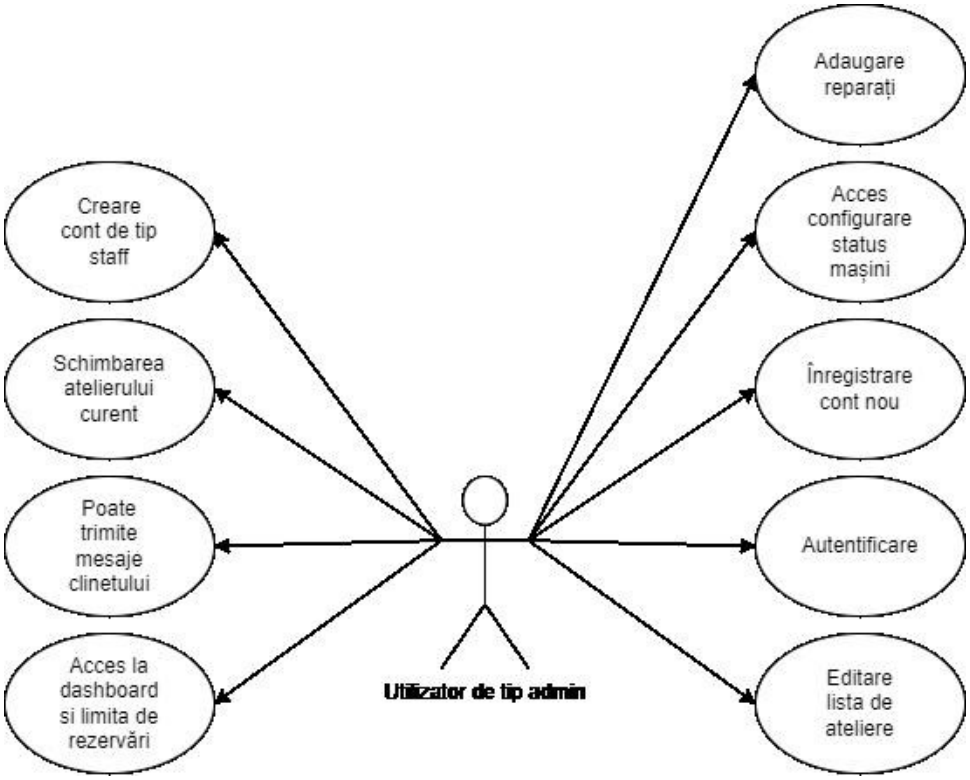

Figura 4.3. Use case-uri utilizator de tip admin

Utilizatorul de tip staff

Un utilizator de tip staff are mai puține permisiuni decât unul de tip admin. Accesul acestuia, pe lângă opțiunile de a crea un cont nou și de a se autentifica, se limitează la pagina de mașini (unde are acces complet la modificarea statusului și a reparațiilor făcute pe mașini), și la comunicarea prin intermediul mesageriei cu clienții. Diagrama UML o putem observa în figura 4.4

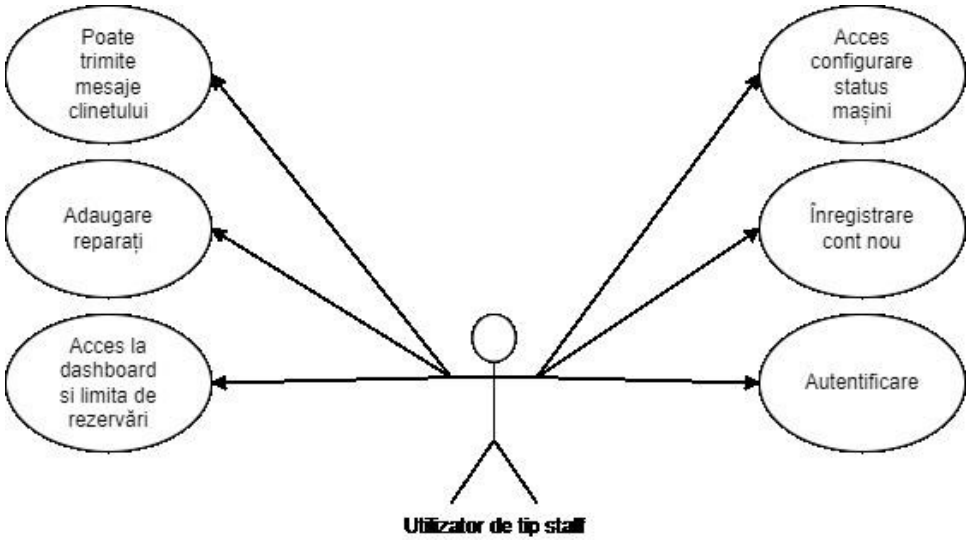

Figura 4.4. Use case-uri utilizator de tip staff

4.4.2 **Descrierea detaliată a cazurilor de utilizare**

Crearea unui cont nou

Descriere: Un utilizator neautentificat dorește să creeze un cont nou pentru aplicație.

Actor: Principalul actor este cel neautentificat.

Interesul actorului pentru realizarea cazului de utilizare: Crearea unui cont de utilizator pentru a se putea autentifica și beneficia de toate functionalitățile aplicației.

Pre-conditii: Aplicația trebuie a fi descarcată.

Post-conditii: Utilizatorul a fost înregistrat în baza de date a aplicației și acesta se poate autentifica.

Scenariul de utilizare:

Când intra pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul de "Înregistrare", aflat pe partea din stânga a aplicației, iar acesta îl redirecționeaza pe pagina de înregistrare cont nou.

Pe pagina de înregistrare se completează toate datele necesare: email, parolă, prenume, nume de familie, numărul de telefon, se confirm parola și se apasă butonul "Register".

Se validează datele, toate câmpurile trebuind să fie valide, apărând un mesaj de eroare pentru orice câmp care nu corespunde formatelor sau sunt goale.
Dacă toate datele sunt valide, după apăsarea butonului "Register" aplicația te redirectionează la pagina de autentificare, ceea ce semnifică crearea cu succes a contului.

Cazul de utilizare se poate observa în diagrama prezentată în figura 4.5:

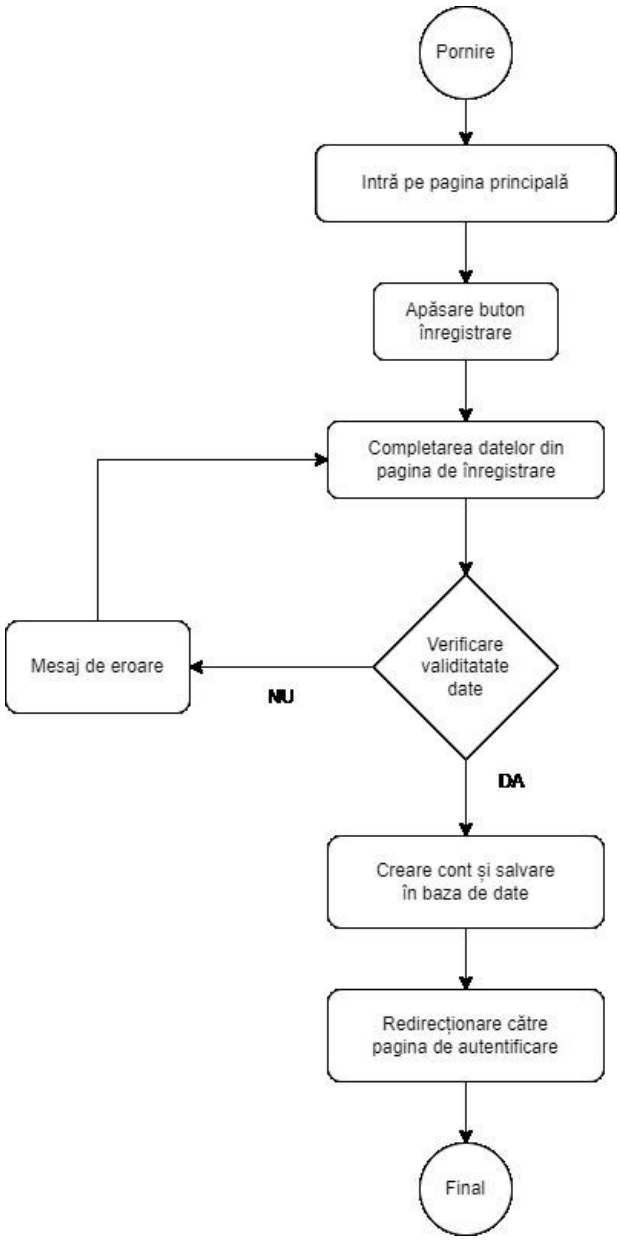

Figura 4.5. Crearea unui cont nou

Autentificare

Descriere: Un utilizator neautentificat dorește să se autentifice în cadrul aplicației.

Actor: Principalul actor este cel neautentificat.

Interesul actorului pentru realizarea cazului de utilizare: Autentificarea în cadrul aplicației, va oferi utilizatorului acces la intreaga aplicație, inclunzând posibilitatea de a face o rezervare.

Pre-conditii: Aplicația trebuie descarcată, iar pentru a completa procesul de autentificare, utilizatorul trebuie să detină un cont.

Post-conditii: Utilizatorul este autentificat pe aplicație.

Scenariul de utilizare:

Când intră pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul de "Login", aflat pe partea din stânga a aplicației, iar acesta îl redirectionează pe pagina de autentificare.

Pe pagina de autentificare se vor introduce emailul și parola și se va apăsa butonul "Submit".

Se validează datele, se verifică existenta contului în baza de date, iar în caz că nu există contul în baza de date se va tipări mesajul "Could not sign in".

În caz de succes, utilizatorul va fi redirecționat către pagina de acasă, ceea ce semnifică autentificare cu succes.

Cazul de utilizare se poate observa în diagrama prezentată in figura 4.6:

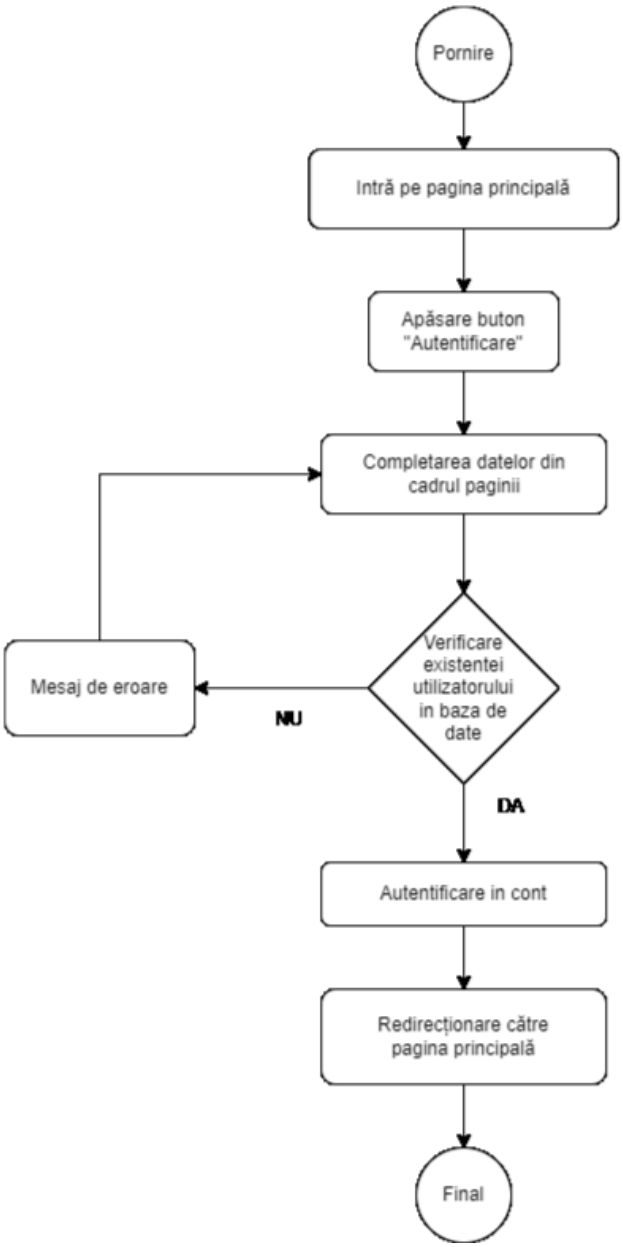

Figura 4.6. Autentificare

Vizualizare ateliere auto

Descriere: Un utilizator dorește să vadă lista atelierelelor din cadrul aplicației.

Actor: Principalii actori sunt atăt cei autentificați cât și cei neautentificați.

Interesul actorului pentru realizarea cazului de utilizare: Utilizatorii sunt interesați de a își găsi un service intr-o locatie convenabila, pentru a putea face o rezervare la acel service si a aduce masina in cauza cu probleme.

Pre-conditii: Aplicatia trebuie descarcată, iar pentru a completa procesul de autentificare, utilizatorul trebuie sa dețină un cont.

Post-conditii: Utilizatorul poate vedea lista de ateliere și poate vedea locația acestora prin intrarea pe pagina fiecăruia.

Scenariul de utilizare:

Când intră pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul de "View services", aflat pe partea din stânga a aplicației, iar acesta îl redirecționeaza către pagina cu lista de ateliere.

Pe pagina cu lista de ateliere se va putea selecta un atelier.

Cazul de utilizare se poate observa in diagrama prezentată în figura 4.7:

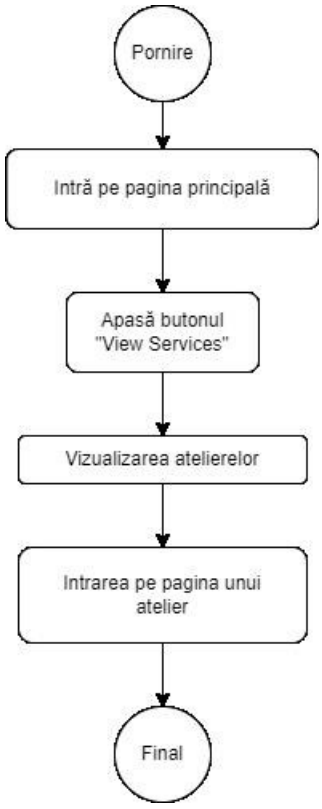

Figura 4.7. vizualizare ateliere auto

Programare masina in atelier

Descriere: Un utilizator autentificat dorește să facă o programare la un atelier din cadrul aplicației.

Actor: utilizatorul autentificat.

Interesul actorului pentru realizarea cazului de utilizare: Utilizatorii sunt interesați de a își găsi un service intr-o locație convenabilă, pentru a putea face o rezervare la acel service și a aduce mașina în cauza cu probleme.

Pre-conditii: Utilizatorul trebiue să fie autentificat.

Post-condiții: Utilizatorul va avea o rezervare făcută în cadrul aplicației la unul dintre ateliere.

Scenariul de utilizare:

Când intră pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul de “View Services”, aflat pe partea din stânga a aplicației, iar acesta îl redirecționează pe pagina cu lista de ateliere.

Pe pagina cu lista de ateliere se va selecta unul dintre ele.

Pe pagina atelierului se va apăsa butonul “Make an appointment”.

Utilizatorul ajunge pe o pagină unde este rugat să ofere detalii despre problemele sale, despre ceea ce dorește să repare, despre data în care vrea să facă rezervarea și despre detaliile mașinii.

Se verifică câmpurile introduse, și se va apăsa butonul “Save”. În caz de eroare, vor apărea mesaje corespunzatoare.

În caz de succes rezervarea este creată, iar utilizatorul va fi redirecționat către pagina service-ului.

Cazul de utilizare se poate observa în diagrama prezentată în figura 4.8:

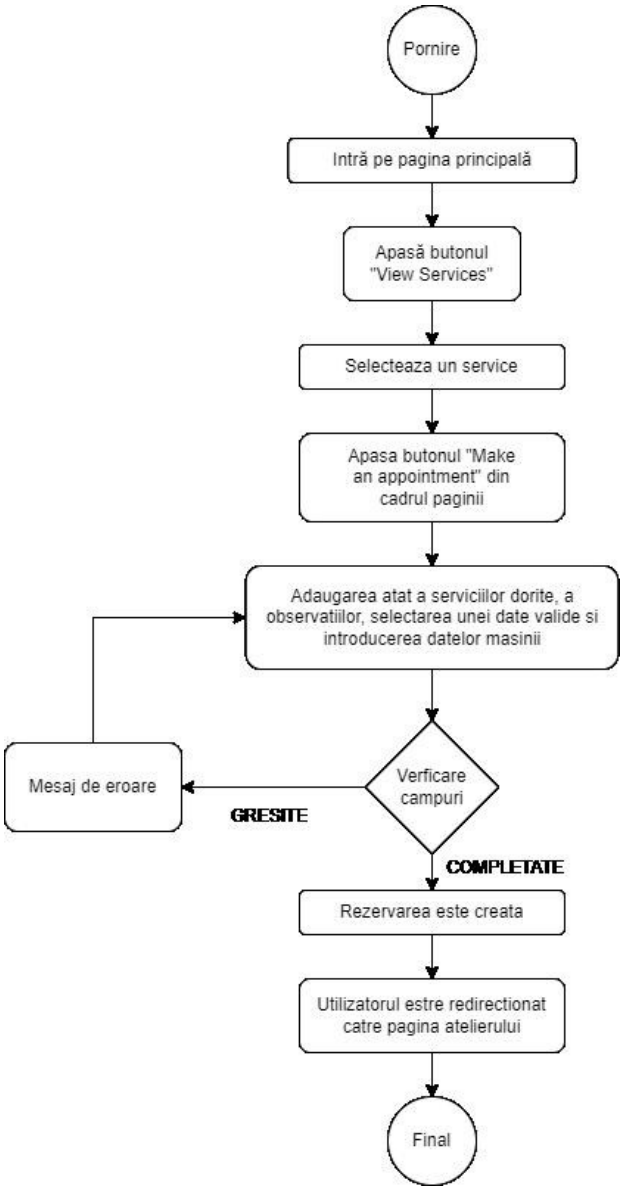

Figura 4.8. Programare masina in atelier

Vizualizare reparații și statusul mașinii

Descriere: Un utilizator autentificat, care a făcut o programare, dorește să vada statusul mașinii sale, raparațiile făcute pe aceasta și pretul total al acestora.

Actor: utilizatorul autentificat.

Interesul actorului pentru realizarea cazului de utilizare: Utilizatorii sunt interesați de a vedea cum decurg reparațiile pentru mașina acestora și pretul final pe care trebuie să il plătească.

Pre-conditii: Utilizatorul trebiue să detină o rezervare făcută în unul dintre ateliere.

Scenariul de utilizare:

Când intră pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul "Your Cars", aflat pe partea din stânga a aplicației, iar acesta îl redirectionează pe pagina mașinilor sale.

Pe pagina cu mașinile sale, acesta va putea vedea mașina sau mașinile pe care pe le are în atelier(unul sau mai multe), împreună cu statusul mașinii și reparațiile sau procedurile făcute la mașină.

Cazul de utilizare se poate observa în diagrama prezentată în figura 4.9:

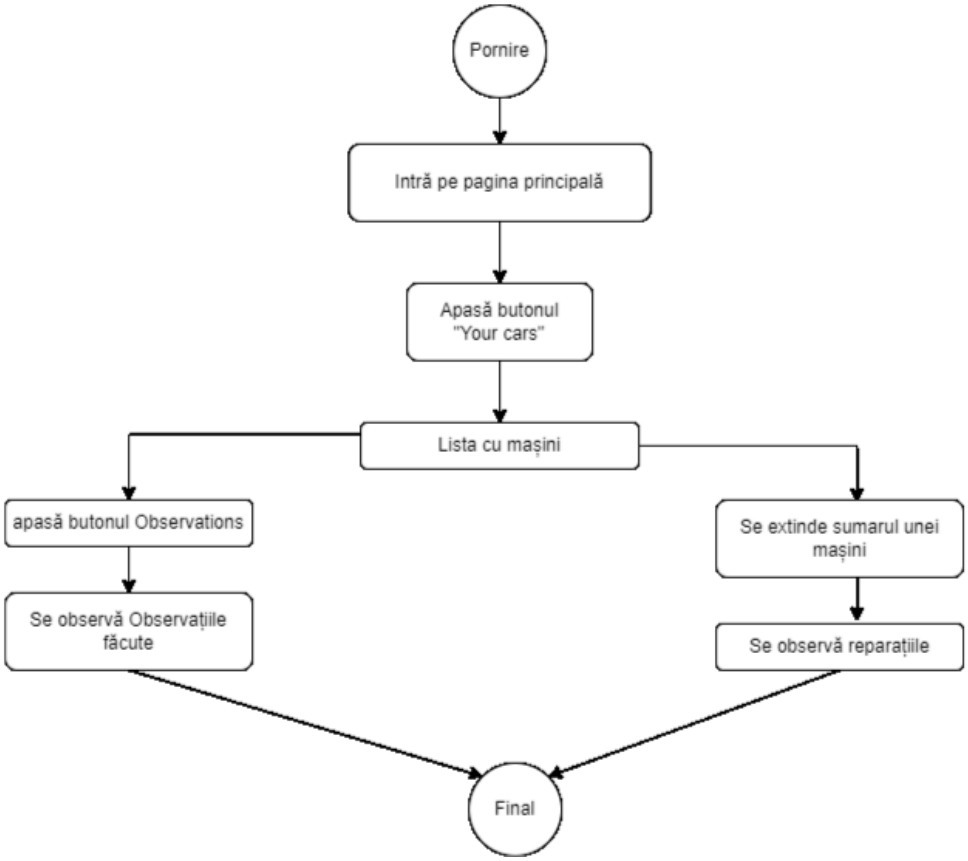

Figura 4.9. Vizualizare reparații și statusul mașinii

Comunicarea prin mesagerie

Descriere: Un utilizator autentificat dorește să comunice prin aplicația de mesagerie.

Actor: utilizatorul autentificat, utilizator de tip admin, utilizator de tip staff.

Interesul actorului pentru realizarea cazului de utilizare: Acest caz de utilizare are trei actori principali. Utilizatorul normal dorește să comunice orice tip de observație cu reprezentantul unu anumit atelier. Un utilizator de tip staff care este asignat unui anumit atelier, va folosi chat-ul pentru a comunica cu clienții atelierului, fie ca el inițiază conversația sau clientul. Utilizatorul de tip admin va folosi mesageria pentru a comunica cu clienții, în caz că acesta nu a asignat un utilizator de tip staff care să se ocupe de service.

Pre-conditii: Utilizatorul trebiue să dețină o rezervare facută în unul dintre ateliere.

Post-conditii: Utilizatorul va putea să comunice prin mesagerie.

Scenariul de utilizare:

Când intră pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul “Your Cars”, aflat pe partea din stânga a aplicației, iar acesta îl redirectionează pe pagina mașinilor sale.

Pe pagina de mașinilor sale, acesta va avea mai multe rezumate ale mașinilor sale. In Fiecare rezumat se afla un buton numit “Chat” pe care utilizatorul va apăsa pentru a deschide pagina de mesagerie. Odată ajuns pe pagina de mesagerie, utilizatorul va putea schimba service-ul cu care comunică prin întoarcerea la pagina mașinilor și apasarea buonului “Chat” din rezumatul unei mașini din alt service.

Cazul de utilizare se poate observa în diagrama prezentată in figura 4.9:

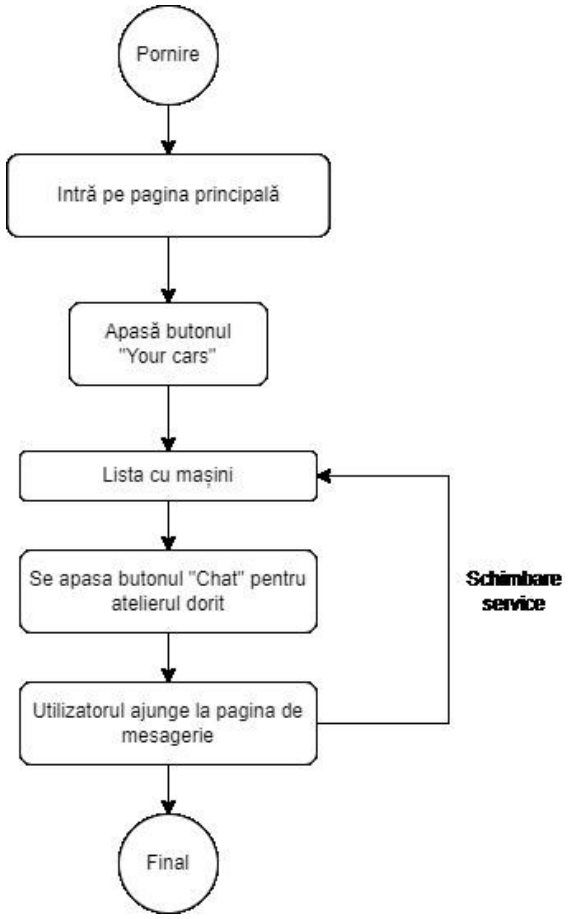

Figura 4.9. Comunicarea prin mesagerie

Editarea listei de ateliere

Descriere: Un utilizator de tip admin dorește să adauge, sa stearga sau sa editeze service-uri din lista de service-uri prezente in aplicatie, care vor fi afisate clientilor.

Actor: utilizatorul de tip administrator.

Interesul actorului pentru realizarea cazului de utilizare: Adminul va dori sa editeze această lista în caz că trebuie să schimbe detalii legate de service-urile existente în aplicație, deoarece această listă va fi afisată clienților.

Pre-conditii: Utilizatorul trebiue să fie autentificat cu un cont de admin

Post-conditii: Utilizatorul va adăuga, sterge sau edita lista de service-uri din aplicație.

Scenariul de utilizare:

Utilizatorul se autentifică cu contul de admin

Când intră pe aplicație, utilizatorul ajunge pe pagina principală.

Acesta apasă pe butonul “Services”, aflat pe partea din stânga a aplicației, iar acesta il redirectionează pe pagina ce contine lista de ateliere.

Pe pagina cu lista de ateliere, adminul are opțiunea de a adăuga un service care se adaugă prin introducerea datelor intr-un formular și se apasă butonul “Save”, apoi utilizatorul este redirectionat la pagina din urmă.

Utilizatorul are un set de 3 butoane, pe fiecare service din listă și anume "Edit”, pentru optiunea de editare, “Delete” pentru opțiunea de stergere și “Add Staff” pentru opțiunea de adaugă un cont de tip staff care va fi responsabil de service-ul respectiv. Operația de edit se parcurge identic cu cea de adăugare.

Cazul de utilizare se poate observa în diagrama prezentată în figura 4.10:

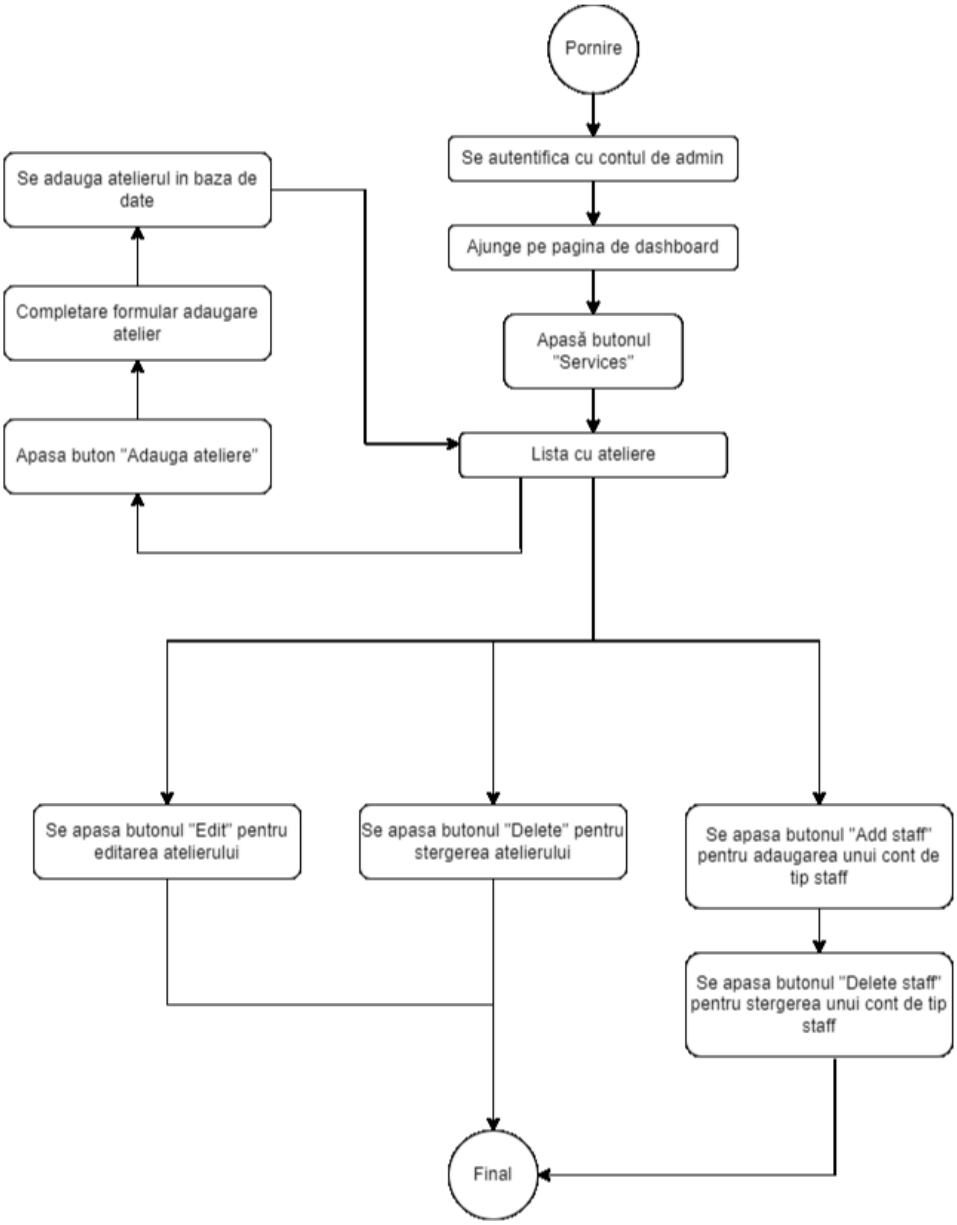

Figura 4.10. editarea listei de aliteliere

Schimbarea statusului unei mașini

Descriere: Un utilizator de tip admin sau de tip staff are permisiunea de a schimba statusul unei mașini.

Actor: Utilizatorul de tip admin și cel de tip staff.

Interesul actorului pentru realizarea cazului de utilizare: Un utilizator de tip admin sau staff are nevoie să schimbe statusul mașinilor în funcție de statusul mașinilor in atelier. Fiecare mașină are un status și căte unul sau două butoane cu care se poate schimba acesta.

Pre-conditii: Utilizatorul trebiue să fie autentificat cu un cont de admin sau de tip staff.

Post-conditii: Utilizatorul schimbă statusul unei mașini din cadrul aplicației.

Scenariul de utilizare:

Utilizatorul se autentifică cu contul de admin sau staff

Cand intră pe aplicatie, utilizatorul ajunge pe pagina principală.

Acesta apasa pe butonul “Cars”, aflat pe partea din stânga a aplicatiei, iar acesta îl redirecționează pe pagina ce conține lista mașini din atelier.

Pe pagina cu lista de ateliere, adminul are opțiunea de adăuga un service care se adaugă prin introducerea datelor într-un formular și se apasă butonul “Save”, apoi utilizatorul este redirectionat la pagina din urmă.

Utilizatorul are un set de 3 butoane pe fiecare service din lista și anume "Edit”, pentru opțiunea de editare, “Delete” pentru opțiunea de ștergere și “Add Staff” pentru opțiunea de a adăuga un cont de tip staff care va fi responsabil de service-ul respectiv. Operația de edit se parcurge identic cu cea de adăugare.

Cazul de utilizare se poate observa în diagrama prezentată în figura 4.10:

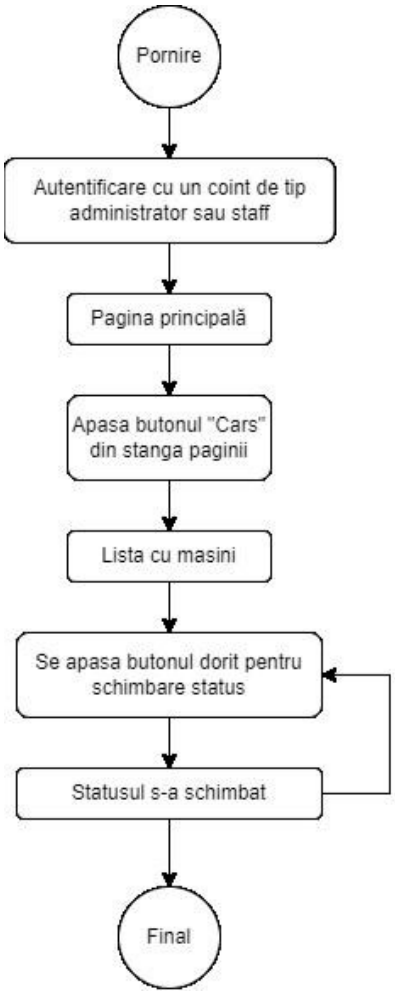

Figura 4.10. Schimbarea statusului unei mașini

4.2. Tehnologii utilizate

4.2.1. Spring boot

Spring boot este un framework open source bazat pe microservicii, care creează un environment complet configurabil, gata de producție, și oferă suport din punct de vedere infrasctructural pentru crearea aplicațiilor Java. Spring boot usurează crearea aplicațiilor de tip standalone, bazate pe Spring. pe care le poți rula cu usurință. Un feature Spring este posibilitatea creării aplicațiilor de mare performanța folosind POJOs (plain old java objects) [8]. Framework-ul spring are multe funcționalitati care îl fac unul dintre cele mai bune framework-uri pentru a crea rapid o aplicatie Java, printre acestea fiind auto-configurare și diverse verificări care indică statusul aplicației, precum statusul de încet sau indisponibil . În mod concret, Spring boot este construit peste Spring și contine toate funcționalitătile Spring.

În concluzie, motivele pentru pentru care am ales framework-ul Spring este atât ușurința de a dezvolta o aplicație de dimensiuni mari cât și flexibilitatea configurării aplicației. Acesta oferă un enviroiment gata de producție, cu auto-configurare, care ajută developerii să își folosească mai multe resurse de timp către elementele de logică ale aplicației. Spring boot de asemenea oferă posibilitatea de a folosi serviciile REST API folosind o configurație minimală(fără configurație XML).

4.2.2. Hibernate

Hibernate este un tool, open source, de tip object relational mapping, care oferă un framework care ajută la maparea modelelor domeniu de tip obiect la baza de date relatională, prin configurare folosing adnotări java și fisiere de tip XML. Am ales acest serviciu pentru capacitatea mapării claselor Java la obiecte de tip SQL, acesta eliberând developerul de a trebui sa se ocupe de persistența datelor. Este un serviciu ușor de folosit pentru prelucrarea și obținerea datelor din baza de date.

4.2.3. Maven

Apache Maven este un tool de tipul open source creat de către Apache Group pentru a crea proiecte in limbajul Java mai ușor, folosind funcționalități pentru un mai bun management al proiectului. Acesta ajută developerul să realizeze development-ul mai avantajos incurajând practici superioare pentru deployment și descrierea dependințelor acestuia intr-un XML numit pom.xml. Am ales să folosesc acest tool pentru capacitatea lui de a face procesul de implementare a proiectului ușor, oferind un blueprint implicit ca și structură al fisierului xml.

4.2.4. JWT (Json Web Token)

Json Web Token este un standard open source folosit pentru comunicarea dintre un client și un server folosind informații sensibile. Acesta este în sine un string de date ce reprezintă o identitate a unui utilizator, la autentificare. Json web token-ul este introdus in aplicație de catre Spring security și crește nivelul de securitate al aplicației. Atunci când este apelat un endpoint de pe back-end, acesta(back-endul) verifică headerul din request pentru un token. Dacă acesta este invalid sau nu există, apelul va returna un mesaj de eroare. De menționat este că tokenul are o durată de

viață limitată, deci va putea fi apelat doar într-un interval scurt de timp dupa creare, acesta conținând o semnătură digitală.

4.2.5. REST

REST este un stil arhitectural care specifică constrângeri precum interfața uniformă dintre clienți și servere, care simplifică și decuplează arhitectura, și ajută fiecare partiție să evolueze independent. Aceste constrângeri, aplicate pe un serviciu web, aduc proprietați precum performanța și scalabilitate, ajutând serviciile să funcționeze cel mai bine pe Web. În stilul arhitectural REST, data și functionalitătile sunt considerate resurse și sunt accesate folosind URI-uri (Universal Resource Identifier). Clienții ți serverele interschimbă reprezentatii ale resurselor folosind un protocol și interfată, standardizate.

REST foloseste un set de principii, bine definit, care incurajează aplicațiile RESTful sa fie simple și rapide [9]:

Interfață uniformă: asigură manipularea resurselor de către un set fix de 4 operații: GET, POST, DELETE, PUT. GET aduce starea curentă a resurselor, POST transfera o noua stare intr-o resursa, DELETE sterge o resursa iar PUT se ocupa de crearea unei noi resurse.

Identificarea resurselor prin URI: resursele sunt identificate printr-un URI deoarece serviciile REST expun un set de resurse care le indentifică, în interactiunea cu un client.

Interactiunea stateful prin hyperlink-uri: orice interactiune cu o resursă este lipsită de stare, toate informațiile necesare pentru tratarea unui request fiind prezente.

Mesaje de la sine descriptive: resursele sunt decuplate de reprezentarea acestora, în asa fel incât conținutul acestora să poată fi accesat în diferite formate precum XML, HTML, PDF, JSON, plain text, etc.

4.2.6. ReactJs

React este o librărie JavaScript folosită în web development, pe partea de front-end, care ajută la crearea elementelor interactive într-un website și este responsabil pentru layaerul view al aplicației. ReactJS folsește un DOM virtual care funcționează inteligent prin reîncărcarea elementelor DOM individuale, în locul încărcării întregului DOM la fiecare schimbare a acestuia [10]. O aplicație de tip React este compusă din multiple componente, care pot conține și un state intern, fiecare fiind responsabilă de afișarea unor elemente reutilizabile de HTML. Componentele pot fi compuse din alte componente mai mici pentru a permite crearea aplicațiilor complete dintr-un număr mic de blocuri mai mari. React permite scrierea acestor componente, folosind un limbaj specific domeniului, numit JSX, care folosește HTML, combinat cu diferite evenimente specifice JavaScript [11].

O reprezentare a ceea ce face React cu acest cod numit JSX este: React convertește acest JSX in DOM-ul virtual, care este o reprezentare a DOM-ului real al browser-ului, astfel acesta face mare parte din procesare pe DOM-ul virtual și apoi înlocuiește doar componentele care s-au schimbat de la ultima randare în DOM-ul browser-ului. O reprezentare vizuală a acestui proces poate fi observat in figura 4.11.

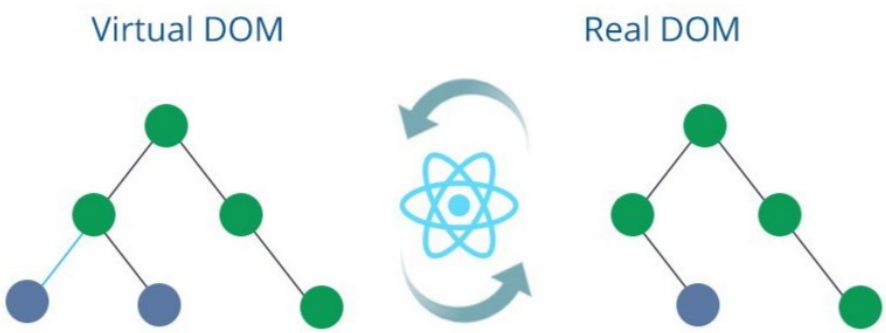

Figura 4.11. Procesul de randare a DOM-ului[2]

Elementele care îmbunătățesc procesul de development și sunt introduse de ReactJS, care au motivat folosirea acestuia în aplicația dezvoltată sunt limbajul JSX, DOM-ul virtual, performanța și simplitatea framework-ului.

4.2.7. CSS și SCSS

CSS, care vine de la Cascading Style Sheets, este un limbaj de stilizare folosit pentru a descrie prezentarea unui document scris într-un limbaj de tip makup precum HTML sau XML. Atunci când un browser afișează un document, acesta va combina conținul documentului cu informația de prezentare de tip CSS. O reprezentare vizuală a acestui proces se poate observa in Figura 4.12.

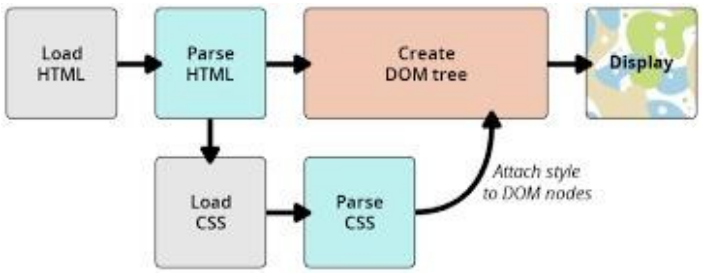

Figura 4.12. Procesul de afișare a documentului [3]

SCSS, numit si Sass (Syntactically Awesome Style Sheet) este un superset al limbajului CSS, care este compilat in CSS, și vine cu anumite avantaje precum diminuarea dimensiunii codului de stilizare. Dezavantajul SCSS fața de CSS este posibilitatea de a produce un cod css mai complex, odată ce este compilat, astfel fiind puțin mai costisitor.

[2] https://www.arrowhitech.com/virtual-dom-its-definition-and-benefits-that-you-must-know/
[3] https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/How_CSS_works

4.2.8. JavaScript

Un limbaj de scriptare este un limbaj high level, folosit pentru a interpreta și a executa câte o comandă pe rând, fiind mai rapide de învățat și de scris decât limbajele compilate, cu o structură completă precum C sau C++.

JavaScript este un limbaj de scriptare folosit pentru crearea și controlul conținutului dinamic al unui website. Conținutul dinamic se referă la toate schimbările făcute pe o pagină care nu necesită reîncărcarea manuală a paginii, conținut precum grafici animate, sau form-uri interactive. Acest limbaj este folosit în aplicație atât în forma JSX, cât și în forma JavaScript în fișiere de utilitate precum cele care exportă constante sau funcții.

4.2.9. HTML

Un limbaj de tip markup este un set de date dintr-un document electronic, prezent pe lângă conținutul propriu-zis al documentului, ce nu va fi afișat pe ecran, dar va dicta cum va fi afișat conținutul și în ce ordine, într-un mod in care permite utilizarea acestuia pentru a afișa conținutul pe o pagina web.

HTML, numit si HyperText Markup Language este un markup language folosit pentru a structura conținutul unei pagini web. Acesta poate fi asistat de tehnologii precum Cascading Style Sheets(CSS), sau limbaje de scripting precum JaaScript. Elementele HTML sunt delimitate printr-un set de tag-uri și pot contine alte elemente HTML. O reprezentare vizuală a unui element simplu HTML poate fi observată in figura 4.13.

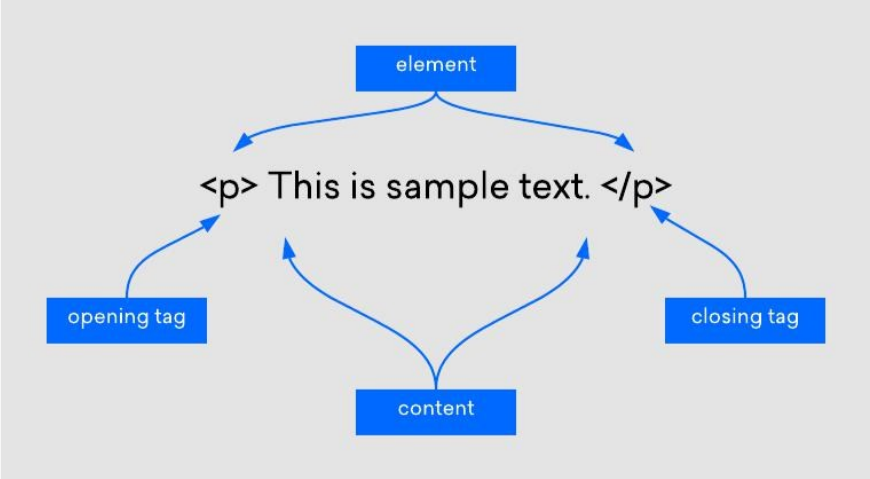

Figura 4.13. Structura unui element HTML

Capitolul 5. Proiectare de Detaliu și Implementare

În acest capitol se va prezenta arhitectura sistemului, si descrierea implementării aplicației, împreună cu deciziile luate si cu modul in care au fost implementate. Se vor explica toate deciziile de implementare separat pentru fiecare componenta a arhitecturii aplicației. Sistemul din figura 5.1 se împarte in 3 componente: componenta client care reprezinta fontendul aplicației, componenta server care est backend-ul aplicație si baza de date (PostgreSQL). Fiecare componenta va avea un subcapitol in care vor fii analizate in detaliu. Putem observa arhitectura generală a aplicației in figura 5.1.

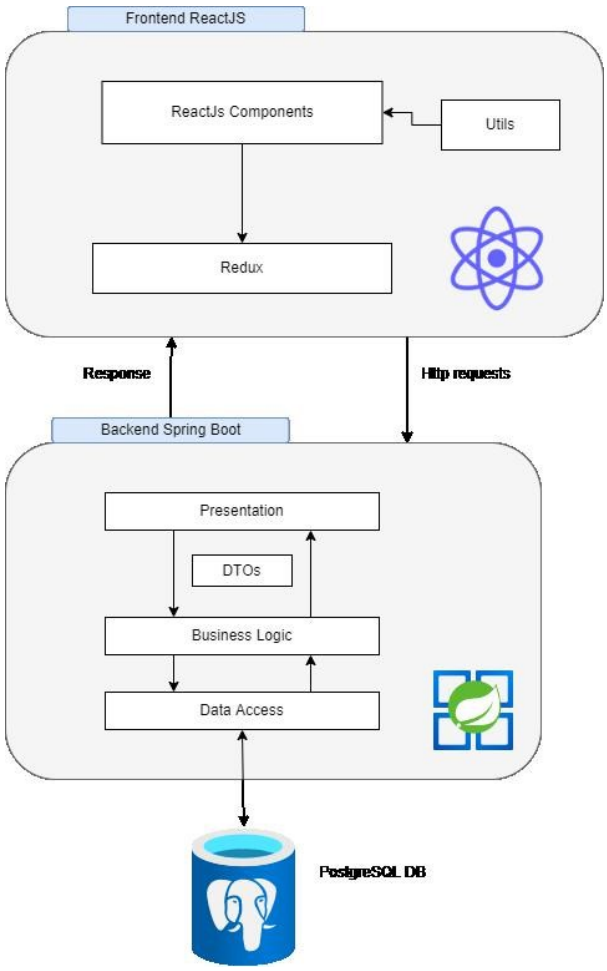

Figura 5.1. Arhitectura generală a aplicației

5.1. Arhitectura componentei server

Componenta de backend și server a aplicației reprezintă baza aplicației și are rolul de a furniza date clienților atunci când sunt solicitate. Această componentă este cea care se ocupă de majoritatea operațiilor utilizatorilor și de accesul la baza de date. Este implementată în limbajul Java, folosind framework-ul Spring boot care oferă suport infrastructurii pe care este creată aplicația.

Primul pas pentru crearea componentei de backend a aplicației a fost crearea proiectului și inițializarea aplicației folosind Spring Initializr. Aceasta este o aplicație care generează un proiect de tip Spring Boot pentru tine, creând structura aplicației cu o specificație de tip (in cazul meu, Maven), fără a genera cod al aplicației. Acesta a fost folosit folosind Spring Boot CLI, din mediul de dezvoltare Intellij. În procesul de creare a proiectului folosind Spring Initializr au fost selectate principalele dependințe ale proiectului necesare pentru crearea în stagii primare a unui backend, precum Spring Boot Dev Tools, Lombok si PostgreSQL Driver. Acestea au fost stocate de către Spring Initializr în fisierul pom.xml unde sunt stocate toate dependințele.

Urmatorul pas după crearea proiectului au fost introduse detaliile conectării la baza de date în fișierul application.properties, creat tot de către Spring Initializr unde au fost setate si diverse configurări pentru aplicație și pentru Hibernate. O parte din configurație se poate vedea în figura 5.2.

```
##############################################
### DATABASE CONNECTIVITY CONFIGURATIONS ###
##############################################
database.ip = ${DB_IP:localhost}
database.port = ${DB_PORT:5432}
database.user = ${DB_USER:postgres}
database.password = ${DB_PASSWORD:rbbo}
database.name = ${DB_DBNAME:AutoFocusDB}

spring.datasource.platform=postgres
spring.datasource.url = jdbc:postgresql://${database.ip}:${database.port}/${database.name}
spring.datasource.username = ${database.user}
spring.datasource.password = ${database.password}

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = validate
spring.jpa.open-in-view=false
spring.jpa.properties.hibernate.show_sql=true

```

Figura 5.2. Fisierul aplication.properties

Aplicația AutoFocus, după cum am menționat, este implementată folosind o arhitectură cu un număr de 3 nivele. Fiecare nivel are un rol bine definit, datele fiind prelucrate și distribuite către nivelul următor. Motivul pentru care am ales acest tip de layout este pentru a mări capacitatea de extindere a aplicației, motivul fiind faptul că fiecare nivel lucrează independent și poate fi extins fără a modifica restul nivelelor.

Nivelul de api este nivelul de sus al aplicației și se ocupă de cererile de tip http și de autentificare în cadrul aplicației.
Nivelul de business este cel care se ocupă de crearea logicii de business a aplicației și folosește nivelul de persistentă.

Nivelul de persistentă al aplicației, este ultimul nivel și conține întreaga logică de stocare a bazei de date fiind responsabilă de returnarea și modificarea datelor din baza de date

O descriere vizuală a nivlelor aplicației se poate vedea în figura 5.3

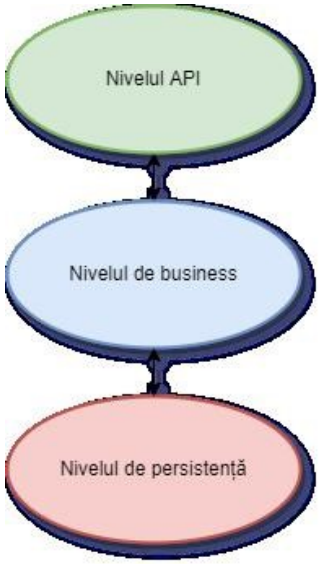

Figura 5.3. Nivelele componentei backend

5.1.1. Nivelul de API(Controller layer)

Nivelul cel mai de sus al arhitecturii este cel de API și conține pachetul de controllere care interactionează cu partea de frontend a aplicației. Controllerele se ocupă de requesturile de tip http venit din componenta de frontend și de realizarea autentificării utilizatorilor. Este responsabilă de convertirea campurilor JSON în obiecte de tip Java și vice-versa. Acest layer folosește nivelul de business pentru realizarea logicii.

Este reprezentat de pachetul “Controllers” și conține controllere care se ocupă de prelucrarea diferitelor requesturi. Pentru fiecare categorie de date din aplicație există un Controller care prelucrează requesturile care furnizează sau cer date legate de acea categorie. Exemple de controllere sunt:

“UsersController” pentru date legate de utilizatori, autentificare, etc.

“CarJobController” pentru cereri legate de adaugarea sau ștergerea lucrărilor la mașini

“MessagesController” pentru cereri legate de prelucrarea mesajelor trimise în aplicație.

Aceste 3 controllere sunt doar 3 dintr-un numar de 7 controllere. Variabila de tip Logger este folosită în fiecare controller pentru înregistrarea requesturilor făcute. Un set de adnotări, care se pot observa și în figura 5.4, se definesc pentru fiecare clasă ale controllerelor și pentru costructor și anume:

```java
@RestController
@CrossOrigin
public class MessagesController {

    private static final Logger LOGGER = LoggerFactory.getLogger(CarJobController.class);
    private MessagesService messagesService;

    @Autowired
    public MessagesController(MessagesService messagesService) { this.messagesService = messagesService; }
```

Figura 5.4. Declararea clasei MessageController

@RestController – este o adnotare folosită pentru a marca un controller restful, folosită la niveul clasei pentru a permite gestionarea requesturilor făcute de către client.
@CrossOrigin – este o adnotare la nivel de clasă care permite distribuirea resurselor din mai multe surse (cross-origin) pentru medodele din controller.
@Autowired – este o adnotare pentru a autoinitializa un bean pentru o medodă sau un constructor

In figura 5.4 se poate observa implementarea metodelor pentru maparea requesturilor folosite pentru a adăuga message în baza de date și pentru a modifica parametru "seen" al mesajelor atunci când sunt văzute de către destinatar.

Adnotarea @RequestMapping este unul dintre cele mai importante adnotări în Spring și este folosită pentru a mapa requesturi HTTP melodelor care le gestionează. În aplicațiile Spring de tip layered, DispatcherServlet este resposabilă pentru rutarea cererilor de tip HTTP către metodele don controllere.

Adnotarea @RequestBody mapează continutul requestului HTTP la un obiect de transfer, suportând deserializarea requestului într-un obiect Java, timpul Java fiind specificat.

Adnotarea @PathVariable este folosită pentru a extrage o anumită valoare din url-ul requestului într-o variabilă declarată în metoda care gestionează requestul.

```java
@RequestMapping(value = "/message/insert", method = RequestMethod.POST)
public ResponseEntity<?> insertMessage(@RequestBody MessagesDTO messagesDTO) throws Exception {
    messagesDTO.setId(UUID.randomUUID().toString());
    MessagesDTO dto = messagesService.insert(messagesDTO);
    return ResponseEntity.ok(dto);
}

@RequestMapping(value = "/message/seen/{id}", method = RequestMethod.PUT)
public ResponseEntity<?> seenMessageById(@PathVariable String id) throws Exception {
    MessagesDTO dto = messagesService.deleteUserById(id);
    try{
        return ResponseEntity.ok(dto);
    }catch (Exception e){
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body( b: "Error seeing message");
    }

}
```

Figura 5.5. Metodele insertMessage și seenMessage

5.1.2. Nivelul de business (Service layer)

Nivelul de business, sau nivelul de service este un termen arhitectural în domeniul software development. Rolul acestuia în aplicație este de a încapsula implementarea logicii aplicației, centralizarea accesului la date și definirea începutului și finalului unei tranzacții. Layerul de service se folosește de cel de persistentă pentru a accesa baza de date în urma modificărilor produse de logica de business. Service-urile definesc funcționalitățile pentru care sunt responsabile, cum sunt accesate, și ce trebuie să returneze în funcție de ce date primesc. Sunt independente atât de controllerele de care sunt folosite cât și de repository-urile pe care le folosesc. Ele primesc date de la unul sau mai multe repository-uri și aplică diverse filtre, transformări și agregări la acestea în funcție de obiectiv.

Layerul service este prezent în pachetul Services, unde se află un număr total de 8 service-uri. Sunt 7 service-uri, unul pentru fiecare controller și unul numit “EmailService” care conține metodele care folosesc Java Mail Sender pentru a trimite email-uri utilizatorului din aplicație. Un exemplu de metoda folosită poate fi observată în figura 5.6, metodă folosită pentru preluarea tuturor utilizatorilor activi din baza de date.

Toate service-urile au adnotarea @Service la nivel de clasă pentru a declara clasa ca și service provider și este folosită pentru clase care aduc funcționalități de business aplicației.

Adnotarea @Transactional este o adnotare folosita la metode prezente la nivel de service folosite pentru a combina execuția mai multor modificări la baza de date într-o operație atomica. Asta inseamnă că nu se poate scrie în baza de date pana nu se executa întreagă procedura si este folosita in metodele care editeaza anumite câmpuri din baza de date, de exemplu atunci când se schimba statusul unei mașini.

Metoda “findUsers” din “UsersService” preia toți utilizatorii folosind metoda findAll din JpaRepository și ii filtrează.

```
public List<UsersDTO> findUsers() {
    List<Users> usersList = usersRepository.findAll();
    List<UsersDTO> usersListDTO = new ArrayList<>();

    for (Users u: usersList
        ) {
        if(u.isActive()){
            if(u.getServiceAssigned() != null && u.getRole().equals("1"))
                usersListDTO.add(UsersBuilder.toUsersDTO(u, u.getServiceAssigned().getId()));
            else usersListDTO.add(UsersBuilder.toUsersDTO(u));
        }
    }

    return usersListDTO;
}
```

Figura 5.6. Funcția findUsers din UsersService

5.1.3. Nivelul de persistentă(Repository layer)

Nivelul de persistentă al aplicației contine logică de stocare a datelor și transformă obiecte Java din nivelul de business în rânduri din baza de date și vice-versa. Acesta conține operatiile CRUD necesare pentru interactionarea cu baza de date.

Layerul de presistentă este prezent în pachetul “Repositories”, unde se află câte un repository pentru fiecare entitate din baza de date. Fiecare service este responsabil pentru interacționarea cu datele dintr-o singură entitate și acestea extind clasa JpaRepository care este o extensie JPA(Java Persistence API) a unui Repository. Conține API pentru operații basic de tip CRUD și pentru operații de paginare și sortare. JpaRepository aduce către utilizare un set de funcții pentru manipularea datelor precum:

Save() - salveaza o entitate de tipul celei declarate în baza de date

FindById() - caută o entitate din baza de date care are id-ul egal cu parametrul oferit și returnează un Iterable.

FindAll() - returnează o listă cu toate entitățile din tabela existentă din baza de date.

JpaRepository conține de asemenea un mecanism oferit de infrastructura repository-ului Spring Data, pentru a crea query-uri asupra entităților din repository. Query-ul poate fi definit în mode manual ca și în figura 5.7 sau poate fi definit prin numele metodei, într-un pattern predefinit, cu o structura asemănătoare cu metoda “findById”. Acest tip de creare a qeuery-ului către baza de date este realizat prin cuuvinte cheie precum “find” sau “lessThan”.

In diagrama 5.7 se poate observa clasa repository folosită pentru a manipula datele unui utilizator. Pe lângă metodele predefinite, furnizate de către JpaRepository au fost definite încă 2 metode. Metoda findByEmail primește String-ul cu emailul și returnează utlizatorul care are email-ul respectiv. Metoda deleteEditUser setează campul active din entitatea utilizator cu un id primit în antetul funcției, la valoarea false și are funcționalitatea de a dezactiva utlizatorul pentru a evita stergerea acestuia din baza de date.

```java
public interface UsersRepository extends JpaRepository<Users, String> {

    Optional<Users> findByEmail(String email);

    @Modifying
    @Query("update Users u set " +
            "u.active = false " +
            "where u.id = :id")
    int deleteEditUser(@Param("id") String id);
}
```

Figura 5.7. Ștergerea unui utilizator prin modificarea câmpului active

5.1.4. Diagrama de pachete

In acest subcapitol vor fi prezentate pachetele componentei server și rolul care îl are fiecare în structura aplicației. Pachetele Controllers, Service și pachetul Repositories au fost discutate la subcapitolele anterioare (5.1.1, 5.1.2, respectiv 5.1.3) deci în acest capitol ne vom axa pe Pachetele Entities, DTOs, Builders și Security. Diagrama de pachete și relațiile dintre ele poate fi observată în figura 5.8. Rlația "folosește" din diagrama reprezintă reprezentarea relațiilor de componsitie și agregare dintre clasele pachetelor în cauză.

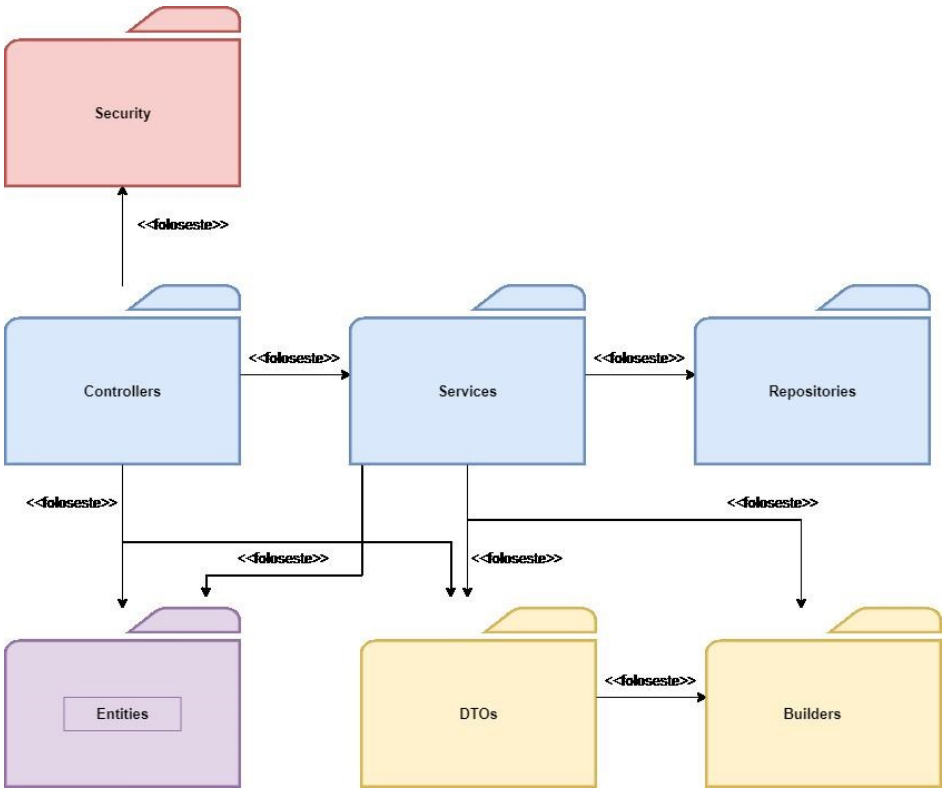

Figura 5.8. Diagrama de pachete

Pachetul Entities conține clasele adnotate la nivel de clasă cu @Entity, care specifică structura clasei ca fiind o entitate, și o mapează la un tabel din baza de date. Adnotări din libraria Java Lombok au fost folosite la nivel de clasă pentru crearea automată a metodelor de tip getter și setter (adnotarea @Data) și pentru crearea constructorilor necesari entității (@AllArgsContructor pentru pentru crearea unui constructor cu toate elementele ca și argumente și @NoArgsConstructor pentru crearea unui constructor gol). Câmpurile din tabela CarJob care ține date despre o lucrare facută pe o mașina se poate observa în figura 5.9.

Pentru crearea field-urilor din baza de date au fost folosite adnotări precum:

@Id – anotația specifică cheia câmpul ca fiind cheia primară a entității, care în cazul proiectului prezentat este de tip String pentru fiecare entitate.
@Column – adnotarea specifică numele colanei entității din baza de date. Dacă un nume nu este specificat câmpul din baza de date va lua numele variabilei.
@OneToOne – adnotarea este folosită pentru a mapa sursa entității cu o entitate dorită în dorința de a defini o relație one-to-many între acestea. Aceasta poate fii unidirctională sau bidirectională. În aplicație, toate relațiile au fost declarate in mod bidirectional.
@OneToMany – este adnotarea care specifică colecția de date ca fiind un desdendent al entității curente. Mai multe entități dintr-o tabelă se vor putea asocia cu o singură entiate din tabela curentă.
@ManyToOne – este adnotarea care specifică asociația dintre un părinte și mai multe entități al tabelului copil. Prin urmare, mai multe entități de tipul curent se pot asocia cu o singură entitate a altei tabele.

```
@Id
private String id;

@Column(name = "name", nullable = false)
private String name;

@Column(name = "description", nullable = false)
private String description;

@Column(name = "active", nullable = false)
private boolean active;

@Column(name = "price", nullable = false)
private int price;

@Column(name = "duration", nullable = false)
private String duration;

@ManyToOne(fetch = FetchType.EAGER, cascade=CascadeType.ALL)
@JoinColumn(name="car")
private Cars car;
```

Figura 5.9. Entitatea CarJob

Pachetul DTOs conține clase cu o structură de Data Transfel Object(DTO). Sunt clase fără metode având câmpuri folosite pentru a încapsula date, acesta obiecte fiind folosite pentru a transfera informații de server la client și vice-versa. Aceste

obiecte au rolul de a reduce mărimea de date a obiectului transferal, acesta putând fi creat cu câmpuri diferite de entitatea din care ia datele. Câmpurile unui obiect de tip DTO sunt create în funcție de necesitățile requestului fără a conține date care nu sunt necesare.

Pachetul builders conține un set de clase care nu conțin câmpuri de date ci metode publice și statice, care au rolul de a crea obiecte de tip DTO din diverse entități, și vice-versa. Builder este un design pattern folosit pentru a ușura crearea obiectelor complexe.

Pachetul Security conține toate clasele care sunt responsabile pentru securitatea aplicație. Și clasele folosite de acestea precum clasa MyUserDetails care implementează interfața UserDetails și ii suprascrie metodele. În acest pachet sunt prezente toate configurațiile pentru generarea și verificarea codului JWT.

5.2. Structura bazei de date

În acest subcapitol se va prezenta baza de date, și va fi explicat fiecare tabel. Pentru stocare datelor s-a folosit baza de date relațională open source postgreSQL. Motivele care au dus la alegerea acestei baze de date este sunt expandabilitatea sa, posibilitatea de a accesa diferite tipuri de date complexe și comformarea sa largă cu standardul SQL. Datele pentru conectarea la baza de date au fost completate în aplication.properties.

Cheia primară a fiecărui tabel din aplicație este un string de tip UUID(Universally Unique Identifier) generat folosind clasa UUID din Java.

Tabela **users** este tabelul în care se stochează datele privind utlizatorii sistemului. Câmpurile email și parola sunt datele pentru autentificare a contului utilizatorului. Emailul este de asemenea folosit pentru trimiterea de email-uri în scop informativ, spre exemplu la terminarea reparației mașinii. Câmpurile firstName și lastName reprezintă numele utilizatorului și câmpul phone reprezintă numărul de telefon al utilizatorului și sunt folosite in scop informativ. Campul role reprezintă rolul clientului și poate avea 3 valori (de la 0 la 2), câte o valoare pentru fiecare tip de utilizator(admin, staff, respectiv client), iar câmpul active reprezintă starea curenta a contului. Dacă câmpul active este false atunci contul nu mai este activ, deci nu mai poate fi folosit.

Tabela **services** este cel care stochează informații despre service-urile prezente în aplicație. Un service conține datele legate de locația sa precum: city(orașul), address(adresa), lat (latitudine) și lng(longitudine). Compurile lat și lng sunt folosite pentru a afișa pentru client locația service-ului pe hartă pe pagina service-ului. Câmpul active reprezintă starea curentă a service-ului. Atunci când are valoarea false, acesta nu mai este activ și a fost sters.

Tabela **customReservationLimit** este un tabel care se ocupă de stocarea limitelor de rezervare customizabile de către admin și staff. Într-un service există o limită de rezervări care se aplică pe toate zilele. Un utilizator de tip admin sau staff

poate alege o anumită dată, sau mai multe pentru a seta o limită anume în acele zile. Astfel, când se află limită de rezervări pe o anumită zi într-un anumit service aceasta se va calcula în funcție de valorile din tabelul customReservationLimit. Acesta are un field date care stochează o dată și reservation_limit care stochează limita de la data respectivă. Campul serviceId stochează id-ul service-ului în cauza, tabelele customReservationLimit și services avand o relație de tip one-to-many.

Tabela **cars** este tabelul care se ocupă de stocarea detaliilor legate strict de mașina. Acestea sunt în primul rând mark, model, generation pentru a stoca marca, modelul, respectiv generația mașinii. Sunt stocate fabricationDate (data fabricatiei mașinii), combustibile (combustibilul) statusul mașinii și prețul total al reparațiilor în câmpul totalPrice. Tabelul are două relații de tip many-to-one deoarece trebuie să salveze id ul service-ului în care se face reparație și utilizatorul care este proprietarul mașinii. Are de asemenea două relații one-to-one, una pentru tabelul problems iar cealalta pentru tabelul reservation.

Tabela **reservation** a fost creat pentru a stoca informații despre rezervarea facută pentru o mașina. O rezervare a mașinii are un singur cămp, și anume resDate care stochează data rezervării. Tabela are o relație one-to-one cu tabela cars deoarece fiecare mașina are câte o singură rezervare.

Tabela **problems** a fost creată pentru a stoca date strict legate de problemele mașinii clientului și opțiunile de reparație alese. Aceasta are campul notes care putem observa că este de tip CHARACTER VARYING(2500) deoarece stochează descrierea problemelor clientului și trebuie să poată stoca un set de observații lungi astfel încât acesta să nu fie limitat. Pe lângă acest câmp, mai are un număr de 6 câmpuri de tip boolean care stochează dacă a fost selectată o anumită operație sau reparație care necesită mașina pe pagina de rezervare. Acestea sunt oilChange(opțiunea de schimbare de ulei), tireChange(opțiunea de schimbare cauciucuri), wheenAlignment(opțiunea de aliniare a direcției mașinii), changeBreakes(opțiunea de schimbare a frânelor), acProblems(pentru probleme legate de funcționarea aerului condiționat) și carCheck(pentru opțiunea de testare și verificare a mașinii). Aceste câmpuri reprezintă tipuri de reparații și operațiuni considerate de baza, într-un atelier de reparații auto.

Tabela **carJob** este tabelă care se ocupă de a stoca date legate de reparațiile și operațiunile făcute pe o mașina. Acesta are cămpul name(numele), description(descriere), price(pret), duration(durată de timp a lucrării). Tabela are o relație de tip one-to-many cu tabelă car deoarece trebuie să stocheze id-ul mașinii pe care au fost făcute lucrarile. Câmpul description este nullable în cazul că operația nu necesită o descriere.

Tabela **messages** stochează mesajele trimise în mesageria aplicației. Acesta stochează mesajul text în campul message, stochează în campul seen de tip boolean dacă a fost citit mesajul, și în campul time data și ora exactă a trimiterii mesajului. Acesta are doua relații de tip many-to-one cu tabela users deoarece ține în memorie atât id-ul utilizatorului care a trimis mesajul (sender) cât și id-ul utilizatorului care a primit mesajul (receiver).

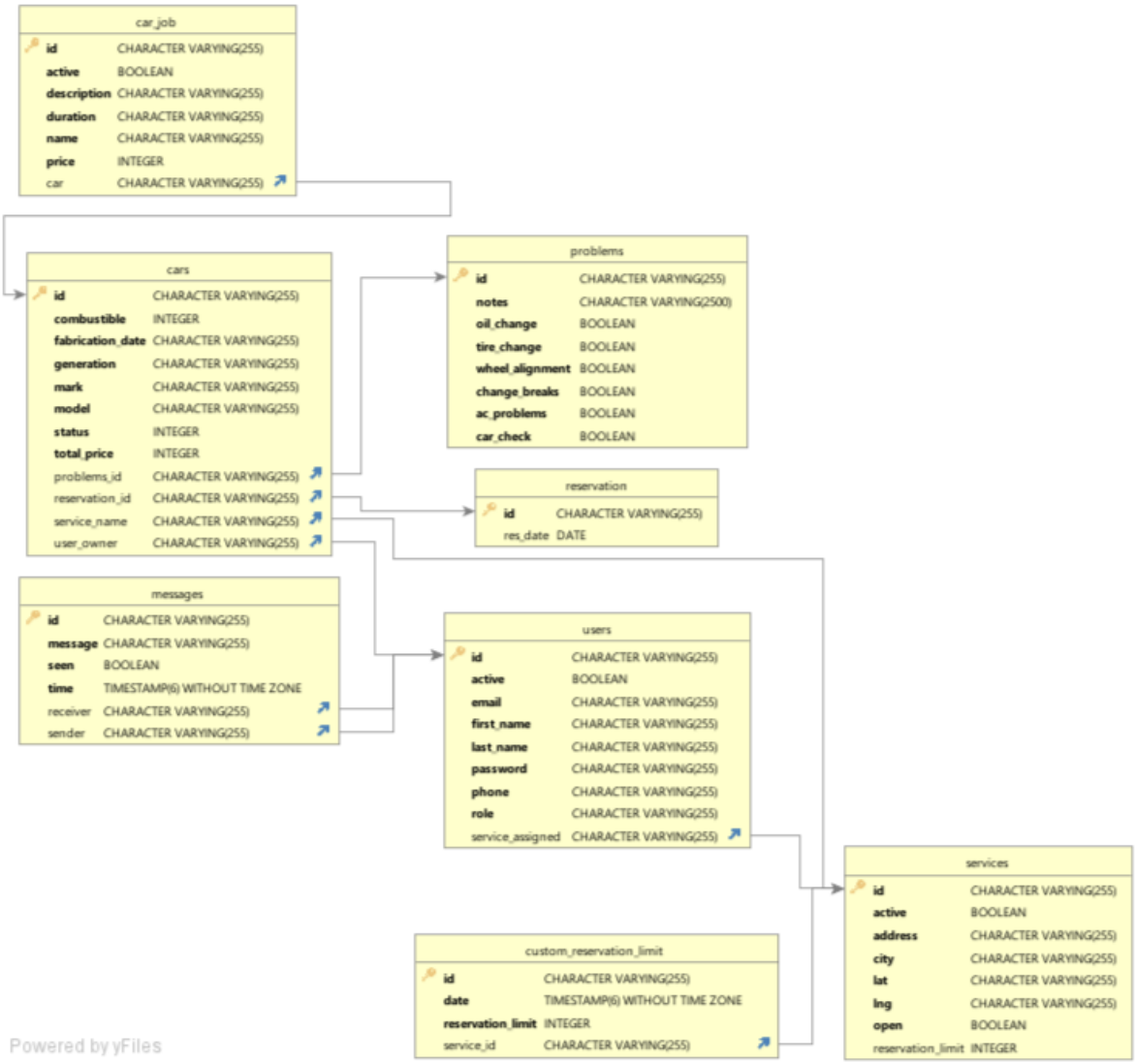

Figura 5.10. Diagrama bazei de date

5.3. Arhitectura componentei client

Componenta de frontend a aplicației reprezintă interfața aplicației și are rolul de a afișa datele printr-o interfață grafică pentru a putea fi interpretate și modificate de către utilizatori. Prin aceasta, utilizatorii aplicației au acces la datele necesare pentru a realiza ficționalitățile aplicației, având posibilitatea de a interacționa cu aceasta.

Este implementată folosind limbajul JavaScript, prin intermediul framework-ului React, bazat pe componente. În locul folosirii bucăților mari de cod pentru a crea elemente vizuale, framework-ul React încurajează împărțirea codului în componente mai mici, reutilizabile și independente. Aceste componente sunt folosite de către alte component, astfel creează componente de complexitate vizuală mai complexa. Aceste componente sunt create cu ajutorul codului JSX, care combină limbajul HTML cu JavaScript. În figura 5.11 putem observa o descriere vizuală a arhitecturii modulare a componentei client. Fiecare componentă a digramei va fi discutată în detaliu în continuare, împreună cu modul de implementare a fiecăruia.

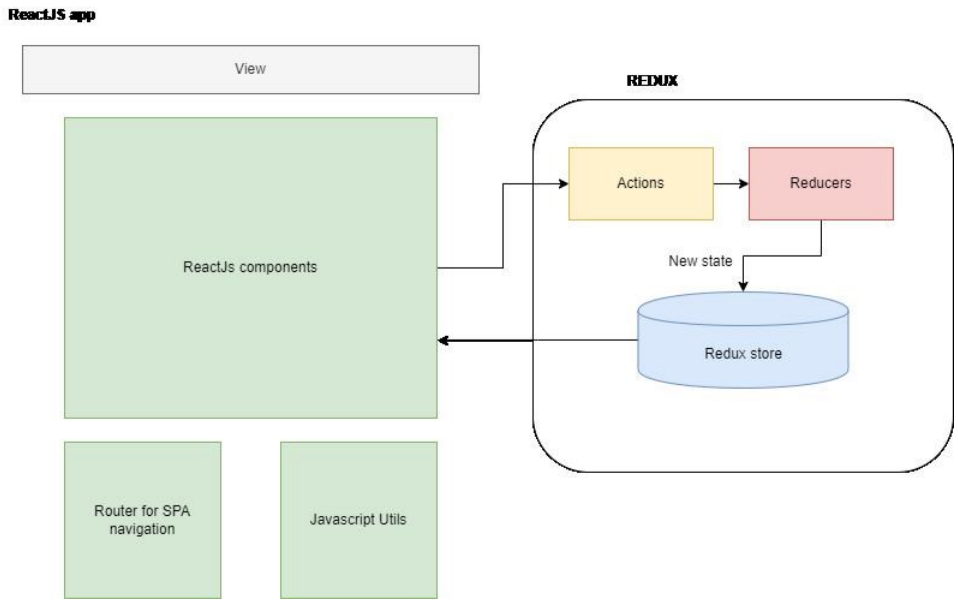

Figura 5.11. Arhitectura componentei clienți

5.2.1. Procesul de creare al aplicației

Vom incepe cu modul de creare a aplicației. Pentru început, s-a creat aplicația folosind comanda **npx create-react-app AutoFocus**. Create React App este un tool folosit pentru crearea unui proiect pentru o aplicație React care instalează toate dependințele necesare și setează environment-ul pentru a suporta ultimele functionalitați JavaScript. Acesta crează întreg pipeline-ul, și foloseste Babel și webpack, folosite pentru a crea un fis

șier împachetat pe care îl poate descărca browser-ul.

Node.js permite folosirea JavaScript pentru crearea aplicațiilor care rulează pe serve, având acces la API-uri și diverse resurse. În acest fel, aplicația conținând configurația de software și hardware necesară pentru execuția codului. Prin urmare,

Node.js este responsabil pentru gestionarea dependințelor și crearea pașilor pentru transformarea limbajului JSX în JavaScript.

Npm (Node Package Manager) este un manager de pachete pentru JavaScript și este principalul manager pentru software-ul Node.js. Acesta ne oferă acces la npm registry, care este o bază de date care conține pachete JavaScript de tip open-source compuse din software și metadate.

5.2.2. Componentele de rutare a aplicației

Componenta montată pe elementul "root" declarat în fișierul html al aplicației începe cu fisierul index.js, care folosește componentă App.js. App.js este compoenta principală a aplicației care actionează ca un container pentru restul componentelor. Aici este folosită libraria React Router, libraria standard de rutare pentru React, care sincronizează componetele UI cu url-ul paginii, având un API cu functionalități precum rutare dinamică sau lazy code loading. Aceasta este responsabilă de intrreaga navigare a aplicației fiind o librărie ușor de folosit ci su multe functionalități. Un set componente sunt folosite în clasa App.js și anume "BrowserRouter", "Switch" și "Route". Componenta BrowserRouter în react-router-dom este folosită pentru rutarea componentelor cu segmente de tip URL. În componenta BrowserRouter a fost declarată o componentă de tip Switch care parcurge totate componentele descendente si afisază prima a cărui path este egal cu URL-ul curent. Aceasta conține un set de componente de tip Route care randează o componenta de tip top-lever UI, dacă path-ul declarat coincide cu cel curent.

În figura 5.12 poate fi observată modul de rutare al aplicației în componenta de tip funcție numită App.js.

```
init();

function App() {
  return (
    <Router basename='/'>
      <Switch>
        <Route path='/login' component={Login} />
        <Route path='/register' component={Register} />
        <ProtectedAdminRoute path="/admin/" component={AdminMain} />
        <Route path="/" component={Main} />
      </Switch>
    </Router>
  );
}

export default App;
```

Figura 5.12 Componanta App.js a aplicației

Sunt declarate rutele "/login" pentru componenta ce afișază pagina de login și "/register" pentru cea care afisază pagina de creare a conturilor noi. Apoi urmeaza rutele "/admin/" care duce către componenta top-level care afiază paginile pentru admin și "/" care duce către componenta top-level pentru clienti. Putem observă că ruta care duce la componenta "AdminMain" este o ruta protejată, care

verifică validitatea contului autentificat, ruta a cărui implementare va fi discutată în cele ce urmează și se poate observa în figura 5.13.

```
function ProtectedAdminRoute({ component: Component, ...rest }) {
  const isUserLogged = useSelector((state) => state.login.loggedIn);
  const userRole = useSelector((state) => state.user.currentUser)?.role;

  return (
    <Route
      {...rest}
      render={(props) => {
        if (isUserLogged && userRole.toString() === USER_TYPE.admin.toString()) {
          return <Component {...props} />;
        }
        return <Redirect to={`/${APP_PAGE_URLS.login}`} />;
      }}
    />
  );
}

ProtectedAdminRoute.propTypes = {
  component: PropTypes.func.isRequired,
};

export default ProtectedAdminRoute;
```

Figura 5.13. Componenta de protecție a rutelor pentru admin

Componenta “ProtectedAdminRoute” primește în parametrul component, componenta top-level ce urmază a fi incarcată și foloseste un Route din react-router-dom. Aceasta funcție verifică tipul utilizatorului și dacă este autentificat prin accesul la redux state și returnează componenta în caz afirmativ, sau redirectionează la pagina de login în caz negativ. Aceasta componentă previne utilizatorii neautentificați cu un cont de admin să acceseze orice url specific acestui rol, adaugând un strat secundar de protecție a aplicației.

Atunci când un utilizator se află pe alt link în afară de “/login” si “/register”, acesta se află pe una dintre componentele AdminMain sau Main, în funcție de rolul acestuia. Utilizatorii autentificați și neatutenficați vor naviga în interiorul componentei Main iar cei de tip staff sau admin în interiorul componentei AdminMain. Compnenta Main Exista atât URI-uri de tip protejate cât și neprotejate, în funcție de cine are acces la ele. Paginile ce pot fi accesate de către utilizatori neautentificați nu au nicio restrictie pentru accesare.

5.2.3. React Redux

React Redux este o librărie JavaScript open source folosită pentru gestionarea și centralizarea state-ulu unei aplicații în React. Utilizarea acestei librării adoptă principiile de design React și anume scrierea delacativă de componente. React este în general rapid, și în general atunci când intervine o schimbare într-o componentă, React reincarcă componenta respectivă, împreună cu toate subcomponentele sale. React Redux este cea mai bună opțiune pentru a imbunatăți performanța aplicației prin evitarea reincarcărilor nenecesare ale componentelor acesta implementează mu$%& lte optimizări de performantă în mod intern astfel incât să reguleze reîncărcarea componentelor pentru a reincărca doar atunci cand vrea developerul. React Redux concentrează state-ul aplicației folosind un reducer numit rootReducer unde este posibilă crearea state-ului în ordinea dorită, care este folosit de către store.js pentru configurarea de persistentă și combinarea reducerilor aplicației. State-ul

aplicație poate fi văzut folosind un extensie pe Chrome numită React Teveloper Tools. O descriere vizuală reducerelor ce compun Redux-ul aplicației AutioFocus se poate observa în figura 5.14.

Login – stochează informațiile legate de autentificare precum tokenul de acces si valoarea booleană loggedIn care devine true atunci când cineva este autentificat pe sistem.
User – stochează utilizatorii sistemului, impreună cu utilizatorul curent autentificat și informațiile acestuia.
Service – contine service-urile aplicației și service-ul curent al state-ului aplicației, în caz că utilizatorul este de tip admin sau staff. Service-ul curent al unui utilizator de tip staff este service-ul pentru care este acestă responsabil. Adminul are optiunea de a selecta service-ul curent intr-un dropdown din bara de top a aplicației.
Car – conține toate mașinile din service, impreună cu reparațiile făcute pe acestea(carJobs) și observațiile și opțiunile clientului(problems).
Reservation – conține datele legate de rezervările făcute în service și au id-ul mașinii care a fost rezervată.
Register – conține doar câmpul registerError, folosit pentru afișarea unei erori pe pagina de înregistreare, în caz ca apelul către api returnează un cod de erore.
Message – conține mesajele dintre utilizatorii aplicației și câmpul partenerId care salvează id-ul partenerului aplicației de mesagerie.

Figura 5.14. Starea Redux a aplicației

5.2.4. Descrierea structurii aplicației

Prin intermediul subcapitolelor 5.3.1 și 5.3.2 a fost prezentată crearea aplicației și a structurii de rutare. În acest subcapitol vom vorbi despre structurarea fișierelor aplicației și rolul acestora. În fisierul src sunt stocate fisierele aplicației, acesta fiind împărțit într-un total de 5 foldere, fiecare având fișiere cu un snumit scop în aplcație.

components – acest fișier conține setul de componente reutilizabile ale aplicației. Sunt componente făcut cu scopul de a fi utilizare în multiple locuri cu diverse functionalități. Fiecare componentă reutilizabilă se află într-un folder cu același nume cu al componentei și un fisier de tip scss pentru stilizare. Exemple de astfel de componente sunt: AppLogo, Button, CarSummary, Message, Modal, Sidebar, etc.

features – acesta conține, în afară de cele aflate în components, toate fișierele ReactJs (JSX) ale aplicației. Aici se află toate toate componentele responsabile de fișierele eplicatiei.

I18 – conține fișierele framework-ului de internationalizare i18n. Conține fișierul JavaScript numit index.js în care este prezentă configurația și limba prezentă în aplicație fisierul en.json în care sunt prezente cuvine și propozitii afisate pe ecran în aplicație. Acestea sunt folosite în aplicație după numele cămpului din fisierul json cu ajutorul i18n. Aplicația AutoFocus este în limba engleză, împreună cu toate denumirile și textul afisate pe ecran. Libraria i18n de internaționalizare permite extinderea numărului de limbi cu usurință, prin crearea unui simplu fișier Json care are câmpurile cu aceeași denumire cu cele folosite în cod. Astfel, librăria mărește posibilitatea de extindere a aplicației și a dezvoltărilor ulterioare, datorinta usurintei cu care pot fi implementate.

redux – conține fisierul store.js al librariei React Redux impreună cu câte un folder pentru toate action-urile și reducer-ele aplicației.

Utils – folderul utils contine fisierele JavaScript reutilizabile care ajuta la crearea aplicatiei. Fiind fișiere compuse din funcții JavaScript și constante, folosite în mai multe locuri în aplicatie, au fost stocate într-un folder care are rolul de a furniza utilizați aplicației. Fișierul constants.js conține constante folosite în aplicație precum tipurile existente de utilizator sau url-urile de tip API către backend. În fișierul utils.js sunt stocate funcții precum compareExactDates pentru compararea datelor sau composeName pentru compunerea unui nume folosind numele și prenumele.

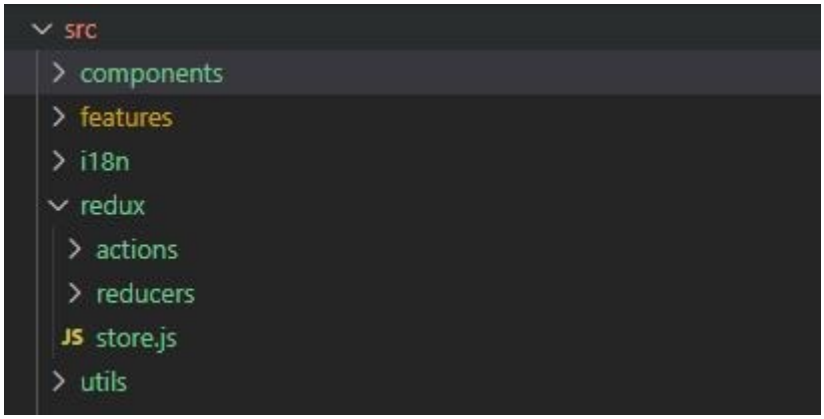

Figura 5.15. Structura fișierelor din aplicație

Capitolul 6. Testare și Validare

În acest capitol se vor prezenta metodele de testare folosite pentru a testa aplicația implementată. Testarea software este o metodă pentru a verifica dacă produsul software implementat se ridică la nivelul așteptărilor și asigură lipsa defectelor produsului. Testarea nu se efectuează pentru a demonstra că aplicația dezvoltată este fără bug-uri, ci pentru a găsi orice tip de problemă pentru a fi rezolvată.

Pe sistemul implementat au fost efectuate 2 tipuri de testare:

Testare manuală
Integration testing

6.1. Testare manuală

Testarea manuală a sistemului este un tip de testare software în care testarea este făcută manual fără a fi folosite tool-uri automate. Testarea manuală este un mod mai primitiv de testare dar este necesar pentru găsirea bug-urilor critice în aplicație. Orice aplicație nou creată trebuie să fie testată manual înainte sa fie folosite alte tipuri de testare automată. Aceasta necesită mai mult efort dar face posibilitatea testării automate fezabilă. Testarea manuala a fost făcută în timpul procesului de implementare care au fost reparate pe rând, pentru a putea atinge cerințele aplicației.

În tabelul 6.1 poate fi observat un model de testare manuală pentru cazul în care un client dorește să facă o rezervare pentru o mașină în service. În cazul afișat, rezultatele obținute sunt aceleași cu cele așteptate, ceea ce reprezintă trecerea testului. În cazul în care unul dintre rezultatele obținute nu sunt aceleași cu cele așteptate testul a picat și eroarea trebuie investigată pentru a putea fi rezolvată.

Acțiune	Rezultat așteptat	Rezultat obținut
Clientul apasă butonul “Make an appointment” de pe pagina service-ului	Pagina de rezervare este afisată	Pagina de rezervare este afisată
Clientul selectează opțiunile din pagina și scrie o observație pentru atelier	Butoanele pentru selectarea opțiunilor își schimbă culoare și textul la apăsare	Butoanele pentru selectarea opțiunilor își schimbă culoarea și textul la apăsare
Clientul selectează o dată dintre cele	Data apare selectată în selectorul de date	Data apare selectată în selectorul de date

disponibile		
Clientul introduce detaliile mașinii și apasă butonul de salvare “Save”	Clientul este redirectionat către pagina atelierului	Clientul este redirectionat către pagina atelierului
Clientul apasă butonul “Your Cars” din bara din stânga paginii	Clientul este redirecționat către o pagină unde este afisată rezervarea făcută	Clientul este redirectionat către o pagină unde este afișată rezervarea făcută

Tabelul 6.1. Model de testare manuală

6.2. Testare automată

Testarea automată este testarea folosind un software care automatizează procesul de validare a unui produs.

Integration testing este testarea software realizată prin gruparea componentelor software. Într-un sistem software mai complex, există multe module sau componente interconectate, care procesează o anumită funcționalitate, lucru care face testarea de integrare dificilă. Ceea ce face testarea de integrare este să verifice modul de interactionare intre cele 2 componente mari, backend și frontend și răspunsul pe care îl oferă, având capacitatea de a descoperi problemele sau inconsitențele care apar.

În aplicația dezvoltată, testarea de integrare a fost realizată simulând cereri de tip HTTP către server. Programul folosit se numește Postman, prin care vom crea cereri HTTP simulând cele efectuate de către subsistemul frontend, testand răspunsul oferit. În figura 6.1 putem observa un request de tip POST către backend cu url-ul “/authenticate” care este apelat de către aplicație atunci când un utilizator execută comanda de autentificare. În conținutul request-ului sunt prezente email-ul și parola unui utilizator din baza de date, prin urmare este returnat Json Wen Token-ul autentificării și datele utilizatorului. Din moment ce url-ul de autentificare este permis de către SecufiryConfigurer, acesta poate fi executat fără autorizare in header.

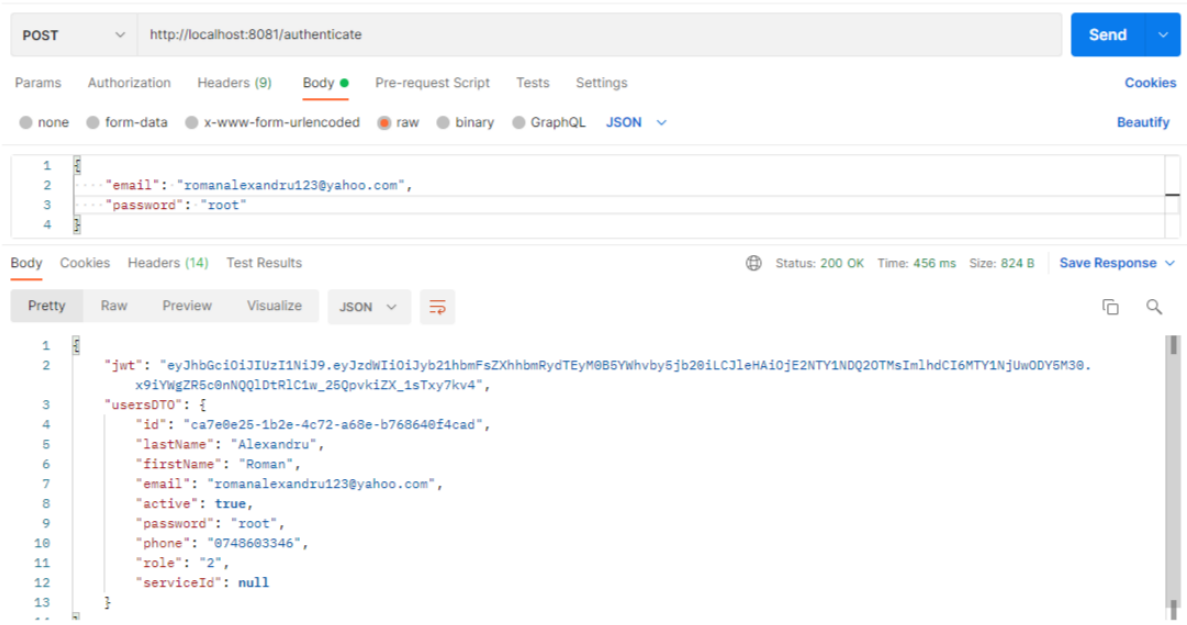

Figura 6.1. Apelul către metoda pentru autentificare

În figura 6.2 se poate observa requestul de tip PUT făcut pentru editarea informațiilor unui service. Aceasta este o fucționalitate a aplicației pe care doar un admin o poate face pentru un service existent. URL-ul pentru editatrea service-ului etste "/service/edit/{serviceId}", id-ul fiind dat ca și variabila în url. Răspunsul pentru editarea cu succes a unui service este returnarea id-ului acestuia, iar în cazul de față putem observa că operația de editare s-a realizat cu succes. În cazul unei erori, un mesaj informativ va fi returnat în locul id-ului.

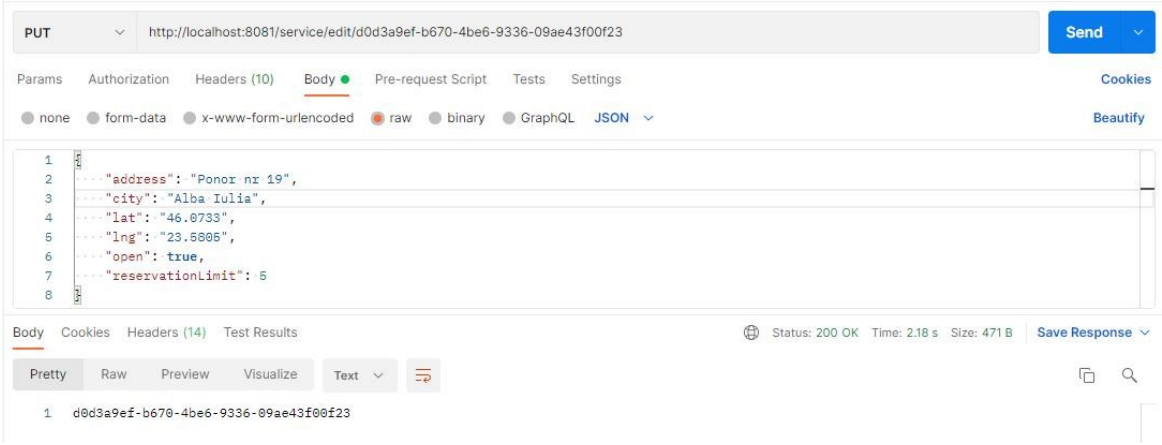

Figura 6.2. Apelul către metoda de editare a unui service

Capitolul 7. Manual de Instalare și Utilizare

7.1. Instalare și rulare

În această secțiune se vor prezenta pașii care trebuie urmați pentru a prezenta și rula aplicația creată.

7.1.1. Instalarea resurselor necesare

Pentru a rula aplicația și a instala React pe windows sistemul va necesita o configurație cu resurse hardware de minim:

Windows XP, Windows 7 (32/64 bit) sau mai

mult. Minimum 4 GB RAM sau mai mult.

10 GB spatiu liber pe disc.

Cel putin un browser de internet precum Chrome, Firefox, Microsoft Edge etc.

Un set de elemente trebuie instalate înainte de rularea aplicatiei:

Pentru rularea componente de backend trebuie instalată **Java**. Pentru a face asta se instaleaza **JDK (Java Development Kit)** pentru sistemul deținut și se adaugă la variabile de sistem variabila JAVA_HOME. Aceasta poate fi descarcată de pe site-ul Oracle: https://www.oracle.com/java/technologies/downloads/ . Trebuie instalat Java **JRE (Java Runtime Environment)** și adaugată variabila JRE_HOME. Ambele variabile de sistem trebuie să dețină calea către folderul instalat pentru JDK, respectiv JRE. Versiunea folosită pentru rularea proiectului este *openjdk-17*. Versiunea recomandată de Java este Java 9 sau mai mare.

Pentru a folosi baza de date trebuie instalată baza de date **PostgreSQL** de pe https://www.postgresql.org/download/windows/ și se execută pașii indicati de către installer. Informațiile de conextare la baza de date sunt prezente în aplication.properties și trebuie folosite la conectare sau schimbate. Acestea pot fi observate în figura 7.1. Baza de date poate fi accesată în pgAdmin 4.

```
##############################################
### DATABASE CONNECTIVITY CONFIGURATIONS ###
##############################################
database.ip = ${DB_IP:localhost}
database.port = ${DB_PORT:5432}
database.user = ${DB_USER:postgres}
database.password = ${DB_PASSWORD:rbbo}
database.name = ${DB_DBNAME:AutoFocusDB}
```

Figura 7.1. Datele de configurare către baza de date

Pentru a putea rula aplicația React mai intâi trebuie instalat Node.js. Acesta poate fi descarcat de pe: https://nodejs.org/en/download/. Versiunea acestuia după instalare poate fi verificată prin comanda *node –v*.

Pentru rularea componentelor de backend și frontend au fost folosite urmatoarele medii de dezvoltare care vor face ușoară importarea și rularea proiectului.

Intellij IDE - backend

Visual studio code – fontend

https://www.jetbrains.com/idea/download/#section=windows .
https://code.visualstudio.com/download .

7.1.2. Pornirea și rularea aplicației

După conectarea aplicației backend cu baza de date putem porni aplicația din mediul de dezvoltare ales cu ajutorul butonului Run.

Pentru afișarea aplicației în browser trebuie pornită componenta de frontend. În cazul folosirii Visual Studio Code se apasă butonul "Open folder" de la secțiunea "File" și se alege folderul care conține codul sursa. Se deschide un terminal și se rulează comanda **npm install** și apoi se rulează **npm start**. Astfel, aplicația va rula pe http://localhost:3000. Un terminal poate fi deschids direct din Visual Studio Code.

În fișierul de configurare aplication.properties din aplicatia server, este setată valoarea *spring.jpa.hibernate.ddl-auto* pe create. Dupa ce aplicația a fost pornită, putem observa crearea tabelelor în baza de date. Dupa prima rulare, acest câmp se va seta pe **validate** pentru a păstra tabelele în baza de date.

7.2. Manual de utilizare

La prima rulare a aplicației, baza de date este goală. Administtratorul sistemului va introduce un cont de tip admninistrator în baza de date a aplicației. Id-ul entității poate fi generat pe https://www.guidgenerator.com/online-guid-generator.aspx pentru a avea un id unic în baza de date. Datele ce trebuie introduse după introducerea identificatorului unic sunt: valoarea true, la câmpul active, pentru a marca contul ca fiind unul activ, email-ul administratorul, prenumele, numele de familie, parola, numele de telefon și rolul, în această ordine. Aplicația are în total 3 roluri: admin, staff și client. Acestea sunt stocate în baza de date ca și nume întregi de la 0 la 2. Prin urmare la rolul utilizatorlui se va introduce valoarea 0, pentru admin. Campul service_assigned este doar pentru contul de tip staff și va ramane null pentru crearea entitatii.

Odată cu rularea comenzii npm start, se va deschide o fereastră din browser cu url-ul http://localhost:3000 și va intra pe pagina de home a site-ului.

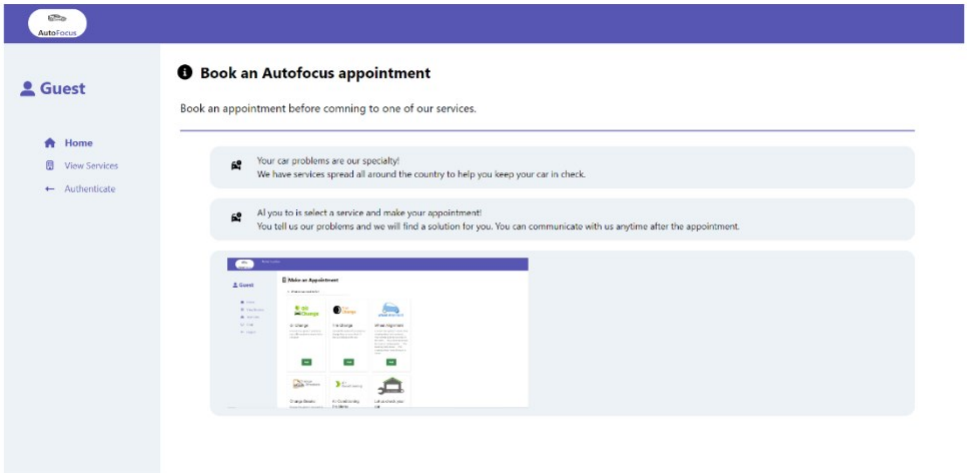

Figura 7.2. Pagina de home a aplicației

Interfața aplicației conține bara de sus cu logo-ul care duce către pagina de home și service-ul selectat în caz că utilizatorul este conectat cu un cont de staff sau admin. Bara din dreapta a aplicatiei este bara de meniu și are rol de navigare a aplicației.

Se apasă butonul din bara de navigare numit “Autentificare” pentru navigarea către pagina de login. Aici se introduc datele administratorului introduse în baza de date și se intră in cont.

Primul lucru care trebuie făcut este adăugarea de service-uri, în funcție de numarul dorit. Se apasă butonul “Services” din navigare iar apoi pe pagina de service-uri se apasă butonul “Add Service”. Administratorul ajunge pe o pagină în care este prezent un formular de adaugare service. Se compleetază datele și se apasă butonul “Save”. Pentru întoarcere există butonul “Cancel”.

După adăugarea service-urile, avem posibilitatea de a adăuga un cont de tip staff prin butonul “Add staff”, prezent în dreptul fiecărui service adăugat în tabelul de service-uri. Prin apăsarea butonului, adminul va fi redirecționat către un form, similar cu cel din pagina de înregistrare pentru utilizator și va introduce datele pentru un cont ce va fii de tipul staff. La adăugarea unui cont în cadrul aplicației AutoFocus se va trimite un mail adresei de mail introduse, deci timpul de așteptare după apsarea butonului save s-ar putea sa fie undeva între 2 si 4 secunde. Acest cont de staff poate fi șters din butonul delete staff. Fiecare service poate fi șters sau editat, după preferință.

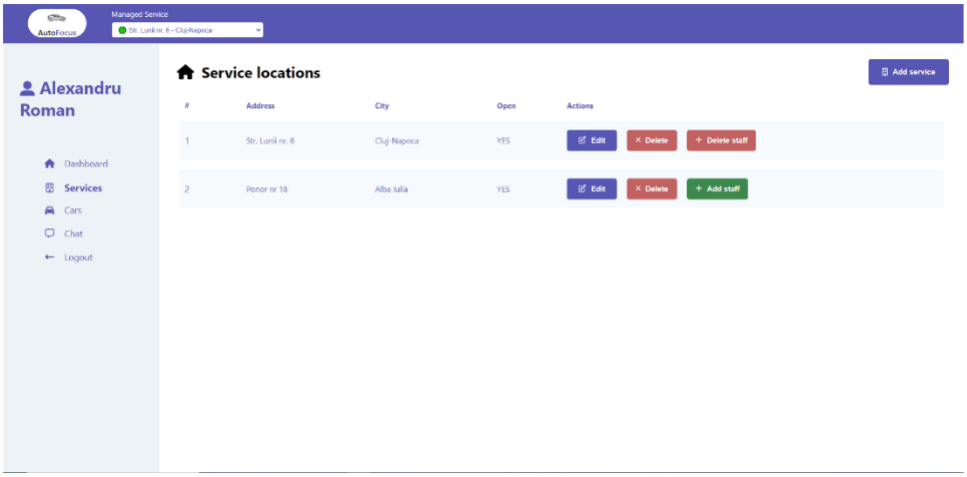

Figura 7.3. Pagina de configurare a atelierelor pentru admin

Adminul are la dispoziție un selector un bara de top a aplicației pentru a alege service-ul a căror date să fie afișate în aplicație. Acesta va fii preselectat cu primul și poate fi schimbat fără a necesita un refresh la pagină. Odata cu selectarea acestuia, adminul va dori să intre pe pagina dashboard care se poate accesa cu ajutorul butonului “Dashboard” din bara de navigare.

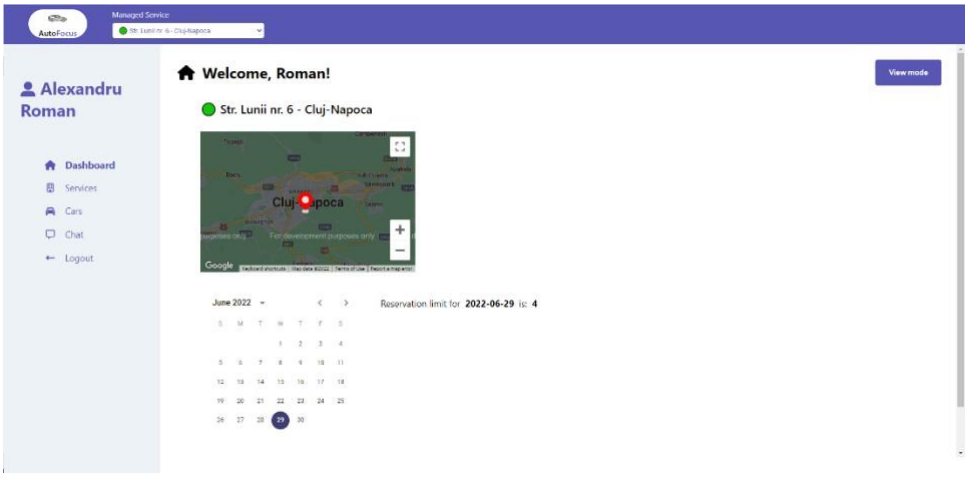

Figura 7.4. Pagina de dashboard pentru admin

Aici se poate vizualiza locația service-ului, limita de rezervări, și se poate edita limita de rezervări pentru fiecare zi în parte. Pagina are 2 mduri: “View mode” și “Edit mode”. În “View mode” se poate interoga limita de rezervări într-o zi cu scop informativ. Limita de rezervări este data de câmpul rezervationLimit la intruducerea unui service. Acesta este valalabil pentru toate zilele din calendar. Pentru a schimba limita se poate intra în modul “Edit mode”. Aici se poate seta limita de rezervări și se poate alege o limită diferită pentrun anumite zile. Formularul de configurare a limitelor de rezervare pentru zile specifice se poate observa în figura 7.5. Această limita se poate schimba atât pentru o singură zi cât și pentru o samptamană întreagă. Se introduce limita și se salvează. Dacă a fost selectată deja o limită diferită pentru o anumită zi aceasta se poate seta din nou fără probleme.

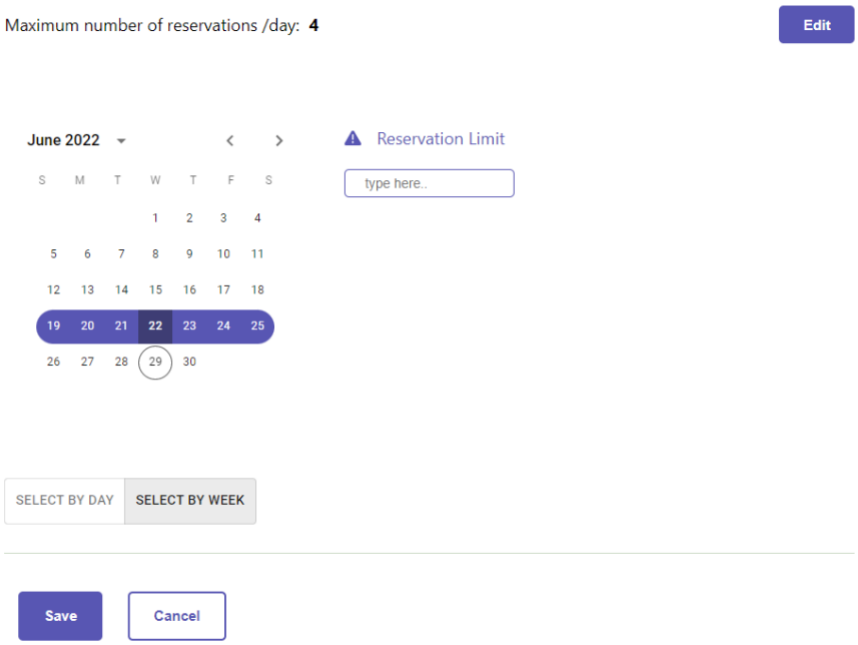

Figura 7.5. Schimbarea limitei de rezervare pentru admin

Administratorul și staff-ul atelierelor trebuie să modifice statusul mașinilor, în funcție de statusul reparației, și să adauge lucrările sumarului mașinii.

În momentul de față toate configurațiile pentru aplicație sunt complete și clienții pot accesa aplicația pentru a face rezervări.

Un client care va intra pe site va ajunge pe pagina de home a website-ului. Acesta va putea intra și alege unul dintre service-urile introduse, după preferința locației și disponibilitate. Pentru a efectua o rezervare aceștia trebuie să fie autentificați. Prin urmare trebuie să creeze un cont în caz ca nu au deja. Pe pagina de login este prezent un buton numit “Dont have an account?” care duce către pagina de înregistrare. Acolo clientul își introduce datele și apasă butonul “Register” pentru a crea contul. Acesta va primi și un email cu parola contului pentru a avea un backup la ea în caz că a introdus o adresă de email corectă. Apoi acesta își introduce datele în formul de login și se autentifică. Acesta pe pagina unui service poate vedea disponibilitatea acestuia.

Prin apăsarea butonului “Make an appointment” clientul va fi direcționat către pagina de creare rezervări. Aici se află un form în care se selectează problemele mașinii și clientul scrie observațiile sale, se selectează o zi din cele disponnibile, și se introduc detaliile mașinii. În selectarea datei se va putea selecta una dintre cele disponibile, cele cu roșu fiind ocupate. Se apasă “Save” și rezervarea a fost creată.

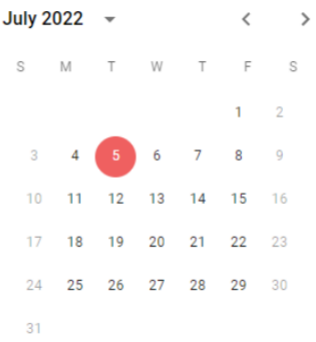

Figura 7.6. Calendarul de rezervări pentru client

Clientul prin accesarea butonului “Your Cars” din bara de navigare, clientul poate intra pe pagina în care ii vor apărea mașinile programate în service. Aici se poate observa progresul mașinilor în service. La fiecare mașină se pot vedea opțiunile de reparație și observațiile făcute.

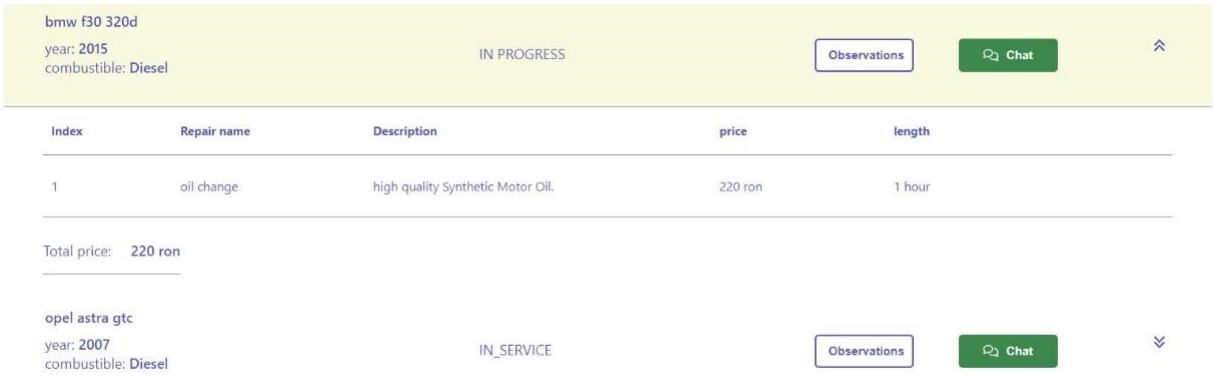

Figura 7.7. Pagina de vizualizare a mașinilor pentru client

Prin apasarea butonului “Chat” se va dechide pagina de chat cu reprezentantul service-ului. În caz ca un cont de staff a fost ales pentru un service, reprezentantul va fi acela. Dacă nu, reprezentantul va fi adminul.

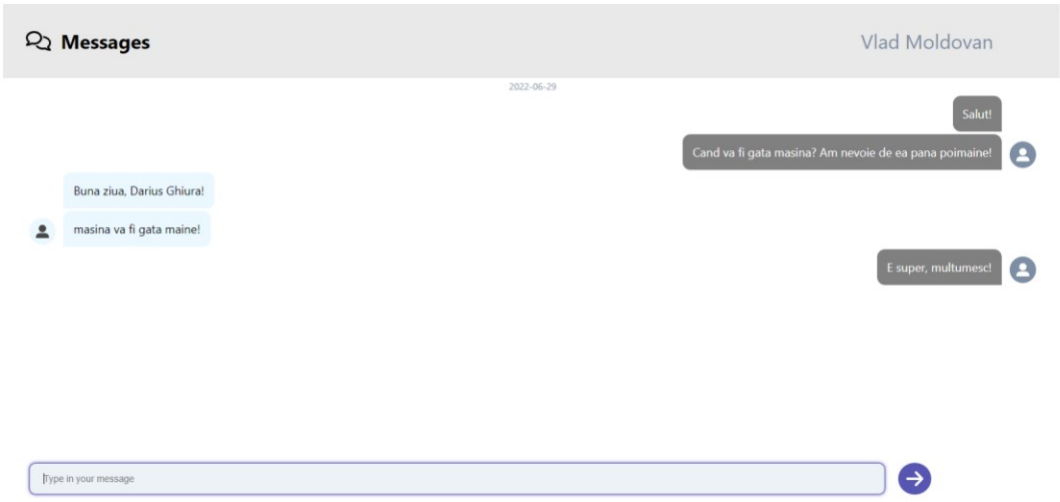

Figura 7.8. Pagina de mesagerie a aplicației

Capitolul 8. Concluzii

În cadrul acestui capitol se vor analiză rezultatele obținute, realizarea obiectivelor propuse și posibile dezvoltări ulterioare care ar fi utile pentru producție.

8.1. Rezultate obținute

Sistemul creat, AutoFocus, îndeplinește toate obiectivele propuse fiind o aplicație care este destinată folosirii de către ateliere sau lanțuri de ateliere mici care își propun dezvoltarea afacerii prin folosirea unui website care a fost creat pentru relaționarea cu clienții.

Aplicația a fost creată în primul rând pentru a face mai ușoara atât comunicarea atelierelor cu clienții cât și pentru a digitaliza dinamica interacțiunii dintre aceștia. În primul rând un service auto are nevoie de un website sau o platformă în care clienții pot vizualiza ce are atelierul de oferit și în primul rând chiar de a afla de existența sa. În lumea modernă, clienții își caută resursele necesare pentru a își rezolva problemele pe internet, lucru care beneficiază firmele concurente care au un website unde clienții se pot informa. Un website fără funcționalități utile este doar o platformă pentru informare, așa că scopul proiectului a fost să creeze o platformă utilă care suportă atât funcționalitatea de rezervare a mașinilor pentru a face viața mai ușoară clienților, cât și o pagina unde clienții pot monitoriza vizual progresul mașinii, asistata de un chat in care se pot comunica posibile probleme si observații.

Site-ul desigur, nu vine decât cu funcționalitate pentru clienți, ci și pentru admin. Acesta are posibilitatea de a configura detaliile necesare pentru funcționarea optimă a aplicației, și pentru eventualele schimbări de program ale atelierelor. Ținând cont de faptul că aplicația suportă mai multe ateliere, cu un număr care nu are nicio limită superioară, limita fiind momentul în care performanța aplicației are de suferit, aplicația vine cu soluția de a crea un nou tip de utilizator cu rolul de staff care este asignat ca fiind responsabil de un service, pentru a distribui responsabilitățile, lucru care îmbunătățește dinamica generală a aplicației.

Contribuțiile personale în vederea aplicației AutoFocus au constat în proiectarea și implementarea unei aplicații de baza care are în vedere administrarea programărilor și comunicației cu clienții, oferind o interfață prietenoasă și ușor de folosit. Pentru proiectarea acestei aplicații a fost acordat un interes crescut dinamicii relaționării unui service cu clienții săi pentru a putea gestiona în mod eficient atât timpul cât și resursele.

Obiectivul acestui proiect a fost rezolvarea problemei de comunicare dintre client și ateliere prin soluționarea problemelor de gestionare a clienților pentru un lanț de ateliere auto. Prin urmare, AutoFocus a îndeplinit cu succes obiectivul propus.

8.2. Dezvoltări ulterioare

După cum am menționat în capitolul precedent, AutoFocus îndeplinește cu întregime obiectivul acestui proiect, însă, există funcționalități ce ar putea fi adăugate cu scopul de a îmbunătății proiectul și de a avea o mai bună gestionare a atelierului. Funcționalitătile care ar putea influența într-un mod pozitiv funcționarea proiectului în lumea reală sunt:

Un sistem de rezervare mai complex – un atelier auto ar putea avea posibilitatea de a oferi clienților alegerea de a aștepta până verificarea sau reparația mașinii este gata, lucru care ar ajuta reparațiile de complexitate minima și verificările de rutină a mașinilor. Prin urmare ar putea fi adăugată o oră fixă alături de data pentru motive de organizare.

Descărcare pdf sumar al reparațiilor – o funcționalitate utilă este descărcarea unui pdf care conține toate detaliile legate de lucrările efectuate pe o mașină, cu scopul de a putea fi printate, cu tot cu prețul total.

Sistem de management intern – o funcționalitate extrem de utilă pentru service ar fi că acesta să ofere suport pentru management intern al service-ului. În momentul de fața aplicația are un rol bine definit în comunicarea cu clienții atelierelor, managementul intern fiind lăsat la o parte. Firmele care sunt atente la detalii vor folosi întotdeauna un sistem de management intern, prin urmare acesta ar putea oferi atelierelor avantajul de a avea o platformă care se ocupă de toate tranzacțiile service-ului. Pentru implementarea acestui concept, un administrator ar trebuii sa poată introduce întreg personalul, împreună cu un program inteligent care sa le poată gestiona munca în funcție de rol. Ar trebui de asemenea adăugate toate echipamentele folosite în service pentru a avea un sistem de management complet.

Bibliografie

[1] I. Peter, „So, who really did invent the Internet?,” 3 Septembrie 2011. [Interactiv]. Available: http://www.nethistory.info/History%20of%20the%20Internet/origins.html. [Accesat 25 Iunie 2022].

[2] M. Nehra, „The Evolution of Web Development & Its Modern Trends,” 24 Septembrie 2020. [Interactiv]. Available: https://www.decipherzone.com/blog-detail/evolution-web-development. [Accesat 25 Iunie 2022].

[3] Intelligence, Mordor, „UNITED STATES AUTOMOTIVE SERVICE MARKET - GROWTH, TRENDS, COVID-19 IMPACT, AND FORECAST (2022 - 2027),” mordorintelligence, 2021. [Interactiv]. Available: https://www.mordorintelligence.com/industry-reports/united-states-automotive-service-market. [Accesat 25 Iunie 2022].

[4] IEEE Transactions on Software Engineering, „Characterizing Architecturally Significant Requirements,” 29 noiembrie 2012. [Interactiv]. Available: https://ieeexplore.ieee.org/document/6365165. [Accesat 25 iunie 2022].

[5] A. Chesterton, „carsguide,” 20 Septembrie 2018. [Interactiv]. Available: https://www.carsguide.com.au/car-advice/how-many-cars-are-there-in-the-world-70629. [Accesat 25 Iunie 2022].

[6] C. Fontanell, „What is Customer Relations? Everything You Need to Know,” 2 noiembrie 2021. [Interactiv]. Available: https://blog.hubspot.com/service/customer-relations. [Accesat 25 iunie 2022].

[7] Microsoft Corporation., „Global State of Customer Service,” 2020. [Interactiv]. Available: https://clouddamcdnprodep.azureedge.net/gdc/gdcPiLLQw/original?ocid=mkto_eml_EM582302A1LA1. [Accesat 25 Iunie 2022].

[8] JAVA FRAMEWORKS | JAVA APPLICATION DEVELOPMENT, „jrebel,” 5 August 2020. [Interactiv]. Available: https://www.jrebel.com/blog/what-is-spring-boot. [Accesat 25 iunie 2022].

[9] Oracle and/or its affiliates, „The Java EE 6 Tutorial,” 2013. [Interactiv]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. [Accesat 25 iunie 2022].

[10] K. Dyrr, The Complete Beginner’s Guide to React, Zenva Pty Ltd, 2018, p. 89.

[11] S. O. contributors, Learning React, p. 138.