**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

****

LUCRARE DE LICENȚĂ

**ServiceFinder**

**propusă de**

***Constantin-Iulian Olaru***

**Sesiunea:** *Februarie, 2020*

**Coordonator științific**

Prof. Colab. Florin Olariu

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

ServiceFinder

*Constantin-Iulian Olaru*

**Sesiunea:** *Februarie, 2020*

**Coordonator științific**

*Prof. Colab. Florin Olariu*

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Data \_\_\_\_\_\_\_\_\_\_\_\_ Semnătura \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemntatul Olaru Constantin-Iulian domiciliul în sat Bratuleni com. Miroslava jud. Iasi str. Principala nr. 14 născut la data de 20/07/1998, identificat prin CNP 1980720226743, absolvent al Universității „Alexandru Ioan Cuza” din Iași, Facultatea de Informatică specializarea Informatică, promoția 2016-2019, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul: ServiceFinder elaborată sub îndrumarea dl. Florin Olariu, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării fasificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, ………………………… Semnătură student …………………………

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*ServiceFinder*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de testetc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Constantin-Iulian Olaru* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

Contents

[1. Introducere 6](#_Toc31587794)

[1.1. Cylex 7](#_Toc31587795)

[1.2. Auto.ro 8](#_Toc31587796)

[2. Serviciile oferite de aplicație 9](#_Toc31587797)

[2.1. Vizitator (utilizator ce nu este conectat) 9](#_Toc31587798)

[2.2. Utilizator conectat 9](#_Toc31587799)

[3. Tehnologii utilizate 10](#_Toc31587800)

[3.1. Android 10](#_Toc31587801)

[3.2. Python 11](#_Toc31587802)

[4. Arhitectura 12](#_Toc31587803)

[4.1. Componenta client 12](#_Toc31587804)

[4.2. Componenta server 14](#_Toc31587805)

[4.3. Baza de date 15](#_Toc31587806)

[4.4. Funcționalitățile și implementarea lor 16](#_Toc31587807)

[4.4.1. Afișarea unei liste de service-uri 16](#_Toc31587808)

[4.4.2. Vizualizarea detaliilor despre un service 18](#_Toc31587809)

[4.4.3. Apelarea, trimiterea unui email și calcularea unei rute 19](#_Toc31587810)

[4.4.4. Cererea unei oferte, programarea și lăsarea unui comentariu 19](#_Toc31587811)

[4.5. Implementarea modulelor 21](#_Toc31587812)

[4.5.1. Autentificarea și schimbarea datelor de profil 21](#_Toc31587813)

[4.5.2. Pagina de *Acasă* și vizualizarea service-urilor (service-urile mele și toate service-urile) 23](#_Toc31587814)

[4.5.3. Cererea unei oferte și Programarea la un service 27](#_Toc31587815)

[5. Scenarii de utilizare 30](#_Toc31587816)

[5.1. Adăugarea unui service 30](#_Toc31587817)

[5.2. Programarea la un service și cererea unei oferte 30](#_Toc31587818)

[5.3. Vizualizarea cererilor și a programărilor unui service 31](#_Toc31587819)

[5.4. Filtrare service-uri 31](#_Toc31587820)

[6. Concluziile și posibilitățile de dezvoltare 32](#_Toc31587821)

[6.1. Concluziile lucrării 32](#_Toc31587822)

[6.2. Posibilități de dezvoltare 32](#_Toc31587823)

[7. Bibliografie 33](#_Toc31587824)

# Introducere

Având în vedere că în zilele noastre smartphone-urilor au devenit un element indispensabil și nevoia de a ne simplifica activitățile zilnice, minimiza timpul pierdut, aplicația ce va fi prezentată în această lucrare își propune centralizarea a cât mai multe service-uri auto din țara, cu scopul ca utilizatorii să poată găsi un service în cel mai scurt timp indiferent de locația sa. Fie că este într-un oraș complet necunoscut fie că este într-un oraș cunoscut, dar nu este sigur care este cea mai bună sau cea mai apropiată opțiune, aplicația îl va ajuta să găsească service-ul potrivit.

În viața de zi cu zi multe persoane s-au lovit de situația de a apărea o problemă la mașina, în timpul mersului și să nu știe niciun service în apropiere. Această situație este destul de neplăcută deoarece pune persoana respectivă în situația de a alege să meargă cu mașina defectă pană la un service foarte depărtat de locația în care se află el, în acest caz riscând să agraveze problema, sau să plătească o sumă foarte mare pentu a transporta mașina pe o platformă, evitând agravarea problemei. Folosind aplicația ServiceFinder să găsească cel mai apropiat service, pentru prima opțiune va reduce considerabil riscul să agraveze problema dacă merge o distanță mult mai mică, iar pentru a doua opțiune își va reduce considerabil cheltuielile de a transporta mașina pe o platformă, datorită faptului că va parcurge o distanță mai mică.

O altă situație neconfortabilă în viața de posesor de autotorism este acel moment când trebuie să schimbi anvelopele deoarece vine iarna sau vara, iar toate servece-urile ce oferă aceste servicii sau vulcanizările au foarte mulți clienți în acea perioadă, fapt ce duce la a sta câteva zeci de minute la o coadă pentru aceste schimburi. Sau când vine momentul să faci revizia automobilului tău și trebuie să umbli pe drumuri, sau trebuie să dai o mulțime de apeluri telefonice la diferite service-uri pentru a te programa la data dorită de tine. Toate aceste probleme sunt rezolvate de aplicația ServiceFinder, cu ajutorul funcțiilor de programare și cerere de ofertă, implementate în aplicație. Prin aceste funcții, utilizator se poate programa pentru orice tipuri de probleme(de exemplu pentru inspecția periodică sau pentru o constatare) ce se rezolvă într-o perioadă scurtă de timp sau poate cere o ofertă pentru repararea unor anumite probleme ce se pot întinde pe o durată mai lungă de timp.

Deasemenea, prin intermediul acestei aplicații și a posibilității de a da o părere, service-urile mai mici ce oferă servicii de calitate își vor putea promova mai ușor activitatea, astfel atrăgând clienți ce va avea ca efect final dezvoltarea sa. Totodată aceste păreri fiind utile și pentru utilizatorii ce caută un service, deoarece pe baza lor pot să-și facă o impresie despre un service, lucru ce îi va ajuta în găsirea service-ului potrivit.

In acest moment, pe piața din Romania mai sunt câteva site-uri web ce oferă funcționalități similare cu aplicația prezentată in aceasta lucrare, dar fiecare având plusurile și minusurile sale. Două dintre aceste aplicații sunt *Cylex.ro* și *Auto.ro*.

## Cylex

Cylex este un site web ce pune la dispozitie o întreagă listă de firme, unde poți sa afli diferite informații despre o firma, cum ar fi adresa firmei, numerele de telofon, adresa de email, o scurtă descriere și cu ce se ocupă acea firmă. În lista de firme puse la dispoziție de acest site se pot găsi deasemenea si service-uri auto.

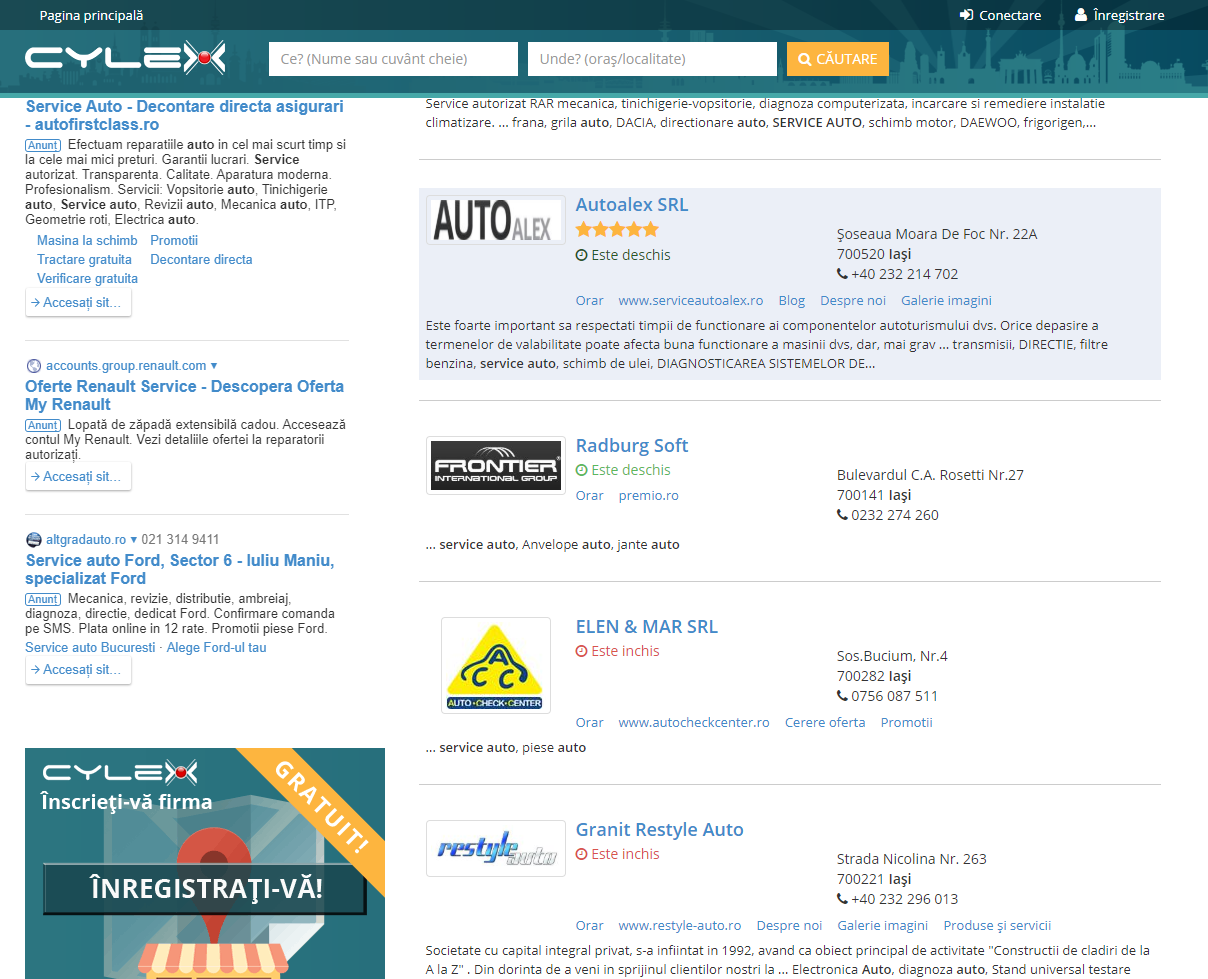


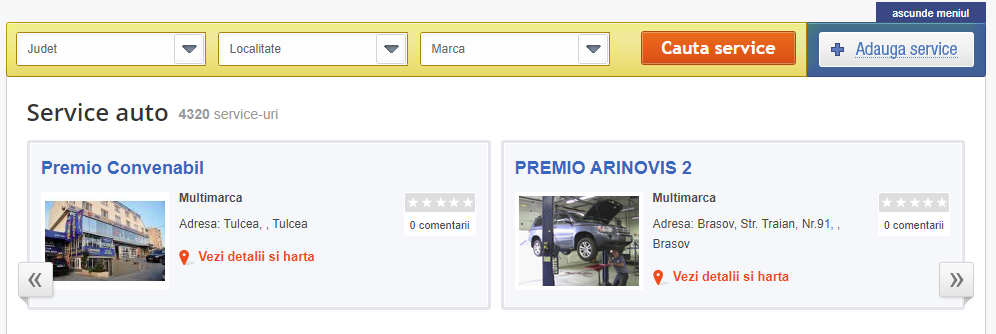
Figura 1 – Imagine de pe site-ul Cylex

Totodată, pe site-ul Cylex se poate lăsa un comentariu alături de un review pentru a ajuta ceilalți utilizatori ai platformei. Acest lucru, precum și cele de mai sus, compun și descrierea site-ului: „*Cylex este un catalog de firme online. Scopul nostru este să creăm legături între firme şi clienţi. Proprietarii de afaceri pot profita de înregistrări gratuite pentru a-şi promova firmele online şi pentru a câştiga mai mulţi clienţi. Utilizatorii pot verifica detaliile de contact şi orarele firmelor. De asemenea pot citi şi scrie păreri despre produsele şi serviciile firmelor căutate.”[[1]](#footnote-1)*

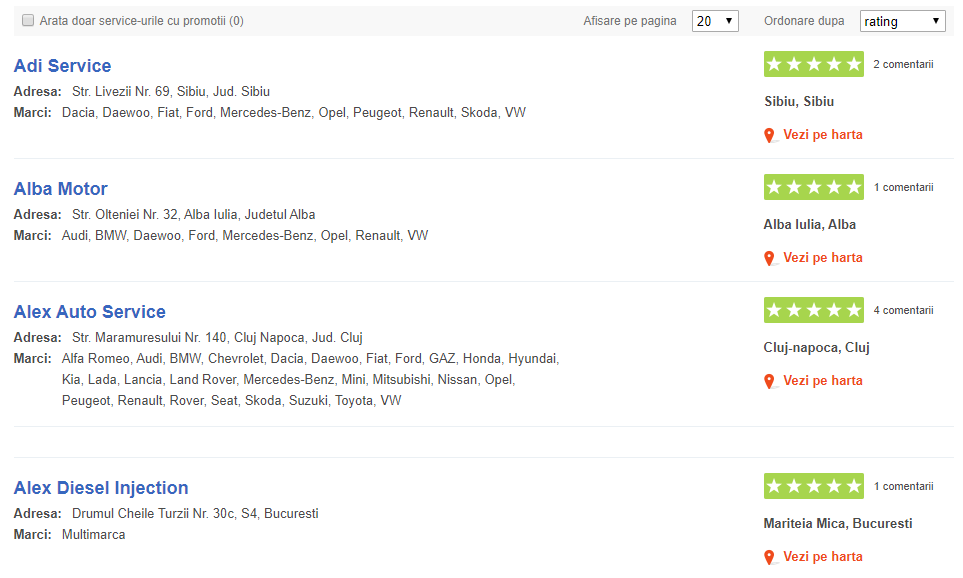
## Auto.ro

Auto.ro[[2]](#footnote-2) este o platformă web ce oferă utilizatorilor diferite service-uri auto din România, deasemenea pe această platformă se pot găsi diferite parcuri auto de unde utilizatorii pot să-și cumpere o mașină. O altă funcționalitate pusă la dispoziția utilizatorilor este posibilitatea de a căuta diferite piese sau anvelope pentru mașina lor personală.

La fel ca și aplicația precedentă, și aici se pot da review-uri și lăsa comentarii, pentru a ajuta alți utilizatori ai platformei.



Figură 1 - Imagine de pe site-ul Auto.ro



Figură 2 - Imagine de pe site-ul Auto.ro

# Serviciile oferite de aplicație

Această aplicație oferă diferite servicii, atât pentru un utilizator ce dorește să-și promoveze service-ul, dar cât și pentru un utilizator ce caută un service pentru a-și repara mașina. Astfel aplicația poate fi privită din două parți: vizitator, utilizator conectat.

## Vizitator (utilizator ce nu este conectat)

Această parte a aplicației este vizibilă tuturor utilizatorilor indiferent dacă sunt conectați sau nu și se adresează în special celor ce nu doresc să-și creeze un cont. Astfel aplicația pune la dispoziție cateva funcționalități de bază cum ar fi:

## Utilizator conectat

Spre deosebire de precedenta parte, aici intră utilizatorii ce au un cont și își caută un service, dar si utilizatorii ce au un service adăugat in aplicație. Astfel acest tip de utilizator are oricând posibilitatea de a deveni proprietar dintr-un utilizator ce caută un service. Pe lângă funcționalitățile enumerate mai sus, utilizatorii conectați, mai dispun și de următoarele:

* vizualizarea ofertelor si a programărilor făcute de el
* adăugarea unui service nou și editarea, precum si stergerea lui

# Tehnologii utilizate

Pentru realizarea acestei aplicații au fost folosite mai multe tehnologii, atât invățate in perioada facultății cum ar fi Python, Android, dar si tehnologii ce nu au fost predate, in special framework-uri ale limbajelor mai sus menționate, cum ar fi SQLAlchemy, Flask, Firebase Storage, Glide sau API-ul Google Maps.

## Android

Această tehnologie bazată pe limbajul java este folosită pentru a crea interfata dintre user si funcționalitățile aplicației. Deasemenea aplicația folosește diferite framework-uri/librării adiționale.

|  |  |
| --- | --- |
| **Firebase** este o platformă oferită de Google pentru dezvoltarea aplicațiilor mobile, dar si web ce oferă o arhitectură care decuplează serverul de client, comunicarea dintre cei doi făcându-se prin intermediul unei bazei de date in timp real. | a51.png (1024×506)  Figura 3 - Arhitectura unui sistem decuplat cu **Firebase** (Today Software Magazine, 2018) |

**Firebase Storage** este unul din diferitele servicii oferite de platforma Firebase. Prin acest serviciu, aplicațiile ce îl folosesc au posibilitatea de a salva fișiere într-o bază de date aflată în Google Cloud Storage. ServiceFinder folosește acest serviciu cu scopul de a salva toate imaginile încărcate de utilizatorii aplicației.

**Ion** este deasemenea o librărie pentru android, de tipul Fluent API ce permite trimiterea cererilor de tipul REST către un server HTTP. Librăria permite atât cereri asincrone, dar și sincrone. Prin această librărie este realizată comunicarea dintre serverul făcut in Python si interfata Android.

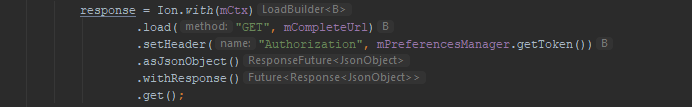


Figura 4 - Exemplu de cod ce folosește librăria **Ion**

**Glide** este o librărie implementată pentru android cu ajutorul căreia se poat încărca imagini. Deasemenea Glide oferă diferite beneficii cum ar fi posibilitatea de a folosi un cache ce crește viteza de incărcare a unei imagini a doua oară. Fiind o librărie de tipul Fluent API, cererile se pot face intr-o singură linie. Această bibliotecă este folosită in ServiceFinder pentru a aduce imaginile din Firebase Storage și pentru a le încărca în view-ul destinat imaginii.

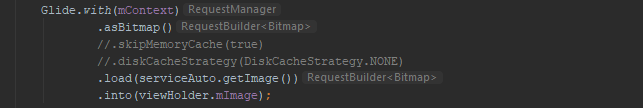


Figura 5 - Exemplu de cod ce folosește librăria **Glide**

**Caldroid** este o librărie similară cu interfața oferită de android[[3]](#footnote-3), dar spre deosebire de acea clasă, librăria Caldroid oferă posibilitatea de a bloca diferite zile sau intervale de zile. Astfel, cu ajutorul functionalității de mai sus, aplicația ServiceFinder poate oferi utilizatorului posibilitatea de a se programa la un service.

**Google Maps Platform,** librăria oferită de Google și integrată in aplicație pune la dispoziție posibilitatea de a adăuga unui service și locația sa, astfel fiind mult mai usor clienților să găsească un service.

## Python

Python este unul dintre cele mai flexibile și populare limbaje de programare, orientat pe obiecte, cu o sintaxă simplă și o semantică dinamică, motive ce m-au convins să aleg ca server-ul aplicației sa fie implementat în aces limbaj.

|  |  |  |
| --- | --- | --- |
| Pe lângă limbajul pur, pentru a realiza serverul am folosit **Flask**, framework prin care se pot crea servere web in Python și **SQLAlchemy**, un framework de tipul ORM ce permite interactionarea cu baze de date. | Figura 6 - Exemplu de cod ce folosește **Flask** și **SQLAlchemy** | |
|  | |  |

# Arhitectura

Din punct de vedere al arhitecturii aplicația se poate împărți în două componente: partea de client scrisă in android, aceasta fiind componenta ce interacționează cu utilizatorul, cu baza de date oferită de Firebase Storage și localizată în Google Cloud Storage, dar și cu cea de-a doua componentă, serverul scris în python, ce oferă o baza de date relațională și funcționalitățile majore a unei aplicații ce respectă principiul CRUD, prin intermediul protocolului HTTP.

Aplicația a fost dezvoltată folosind Android Studio și PyCharm, pentru android, versiunea minimă pe care rulează aplicația este 7.0[[4]](#footnote-4) iar pentru codul scris în python am folosit versiunea 3.7.

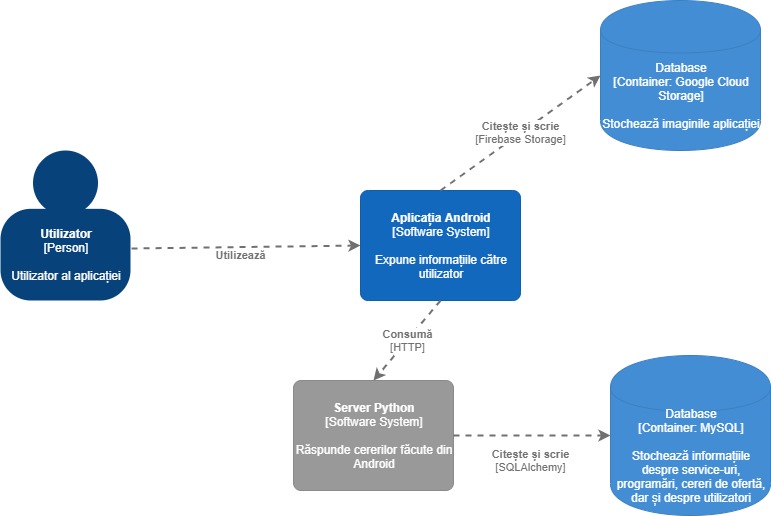


Figura 7 - Diagrama aplicației ServiceFinder

## Componenta client

Această componentă expune utilizatorului, printr-o interfață grafică, toate funcționalitățile aplicației. Astfel, utilizatorul va avea posibilitatea să intre în aplicație și să aleaga dacă dorește doar să vizualizeze lista de service-uri oferită de aplicație și detaliile despre acele service-uri sau pe lângă asta, să poată să facă o programare la un service, să ceară o ofertă de la un service, să lase un comentariu sau să-și adauge propriul service în aplicație. Acestea din urmă fiind posibile doar prin autentificarea userului în aplicație. Salvarea datelor primite după autentificare, pentru ca în următorul ciclu de viață utilizatorul să nu mai fie nevoit să se autentifice din nou, va fi realizată de către singleton-ul *SharedPreferencesManager* care interactionează cu interfața SharedPreferences[[5]](#footnote-5) oferită de android pentru stocarea datelor de tipul cheie-valoare.

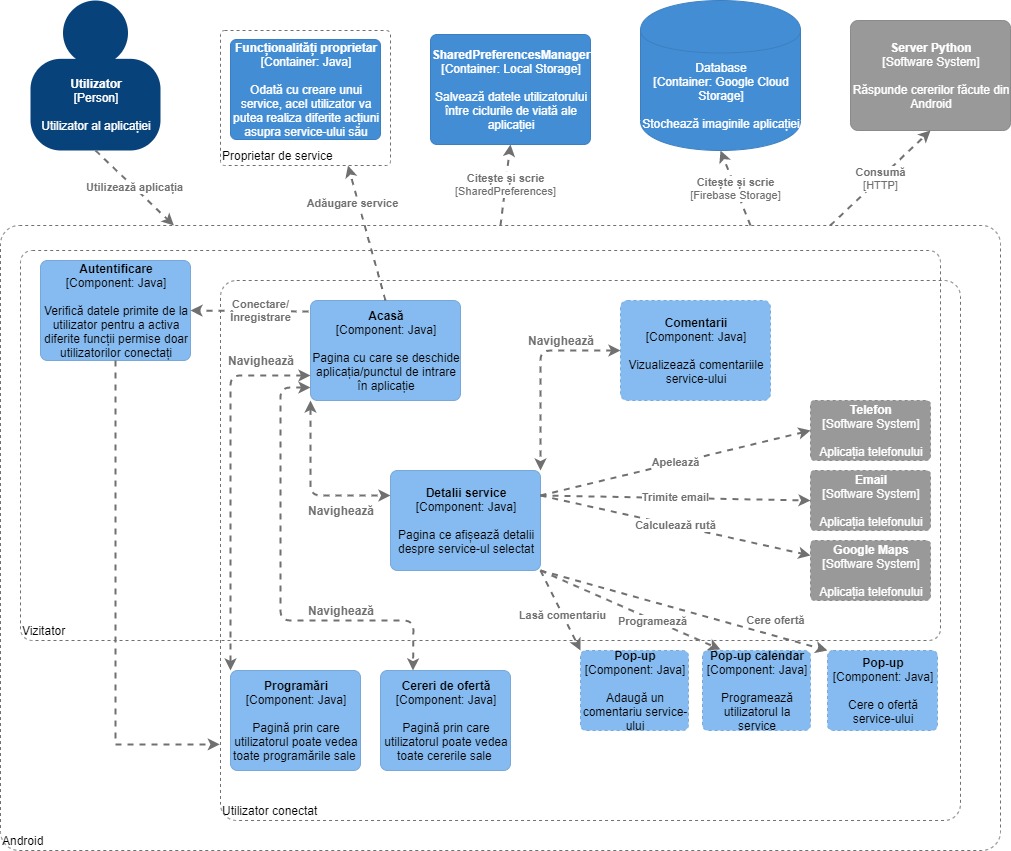


Figura 8 - Diagrama componentei android (utilizator conectat și vizitator)

După adăugare unui nou service sau a mai multor service-uri, utilizatorul va avea posibilitatea să vadă toate service-uril lui, să le șteargă sau editeze, dar și să gestioneze activitatea service-ului pe baza programărilor și cererilor de ofertă făcute de clienți. Pentru a fi mai ușor utilizatorilor să se orienteze dacă service-ul găsit este unul accesibil din punct de vedere al distanței, vor avea afișată distanța față de service, acea distanță fiind calculată pe baza locației curente[[6]](#footnote-6) a utilizatorului și a locației introduse de proprietar la crearea service-ului cu ajutorul unei hărți[[7]](#footnote-7).

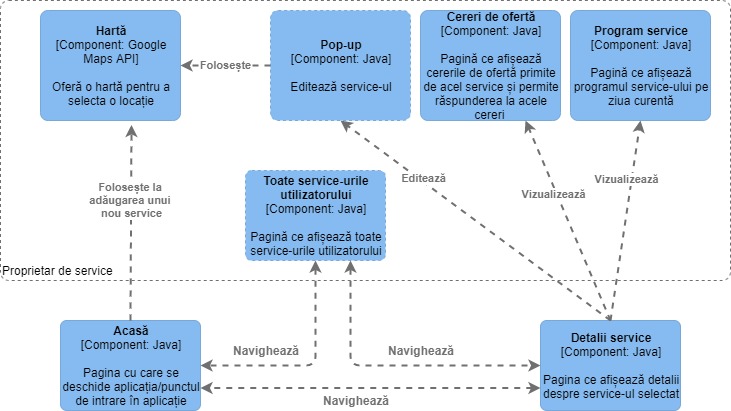
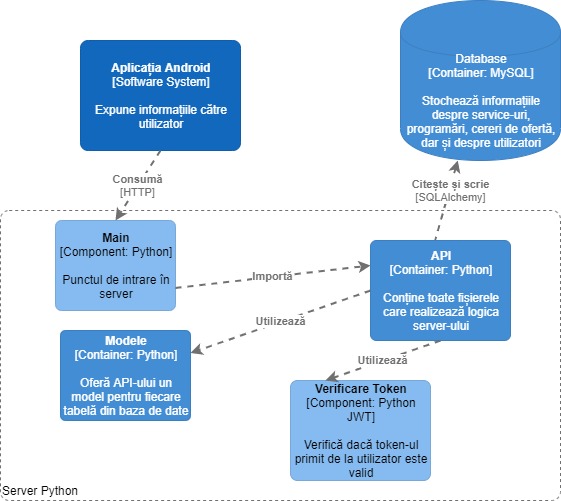


Figura 9 - Noile funcționalități oferite de aplicație după adăugarea unui service

## Componenta server

Server-ul este responsabil de primirea tuturor cererilor HTTP făcute de client și tratarea lor. Astfel, această componentă, este responsabilă de furnizarea tuturor service-urilor existente în aplicație, furnizarea comentariilor pentru un anumit service, autentificarea unui utilizator în aplicație, gestionarea programărilor și a cererilor de ofertă.

Deoarece unele funcționalități nu sunt disponibile pentru utilizatorii neconectați și pentru a evita accesarea unor rute fără a fi identificat, serverul dispune de un mecanism de verificare a identității implementat cu ajutorul librăriei **PyJWT**[[8]](#footnote-8). Acest mecanism presupune, pe partea server-ului, generarea unui token JWT la autentificarea utilizatorului în aplicație și returnarea sa odată cu succesul operațiunii, iar pe partea de client, presupune reținerea acelui token și folosirea lui la fiecare cerere către server (adăugarea acelui token ca valoare a cheiei *Authorization* la lista de headere pentru cererea HTTP curentă).

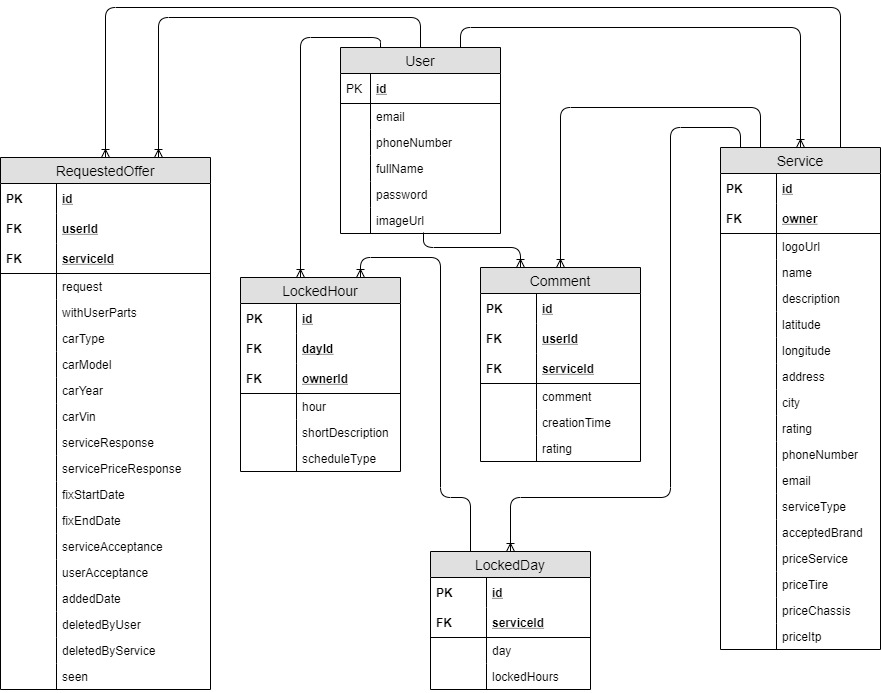


Figură 3 - Diagrama server-ului

## Baza de date

Pentru stocarea datelor am creat o bază de date Sql. Tbelele prezente în această bază de date sunt:

* User => această tabelă are responsabilitatea de a reține toți utilizatorii ce și-au făcut cont în aplicație
* Service => are rolul de a stoca toate service-urile introduse în aplicație, alături de detaliile aferente
* Comment => aici sunt memorate toate comentariile date de utilizatori, pentru toate service-urile
* RequestedOffer => în această tabelă sunt stocate toate cererile de ofertă -urile
* LockedDay => stochează toate zilele ce nu mai au ore disponibile pentru programări
* LockedHour => stochează toate orele ce au fost programate



Figură 4 - Diagrama bazei de date

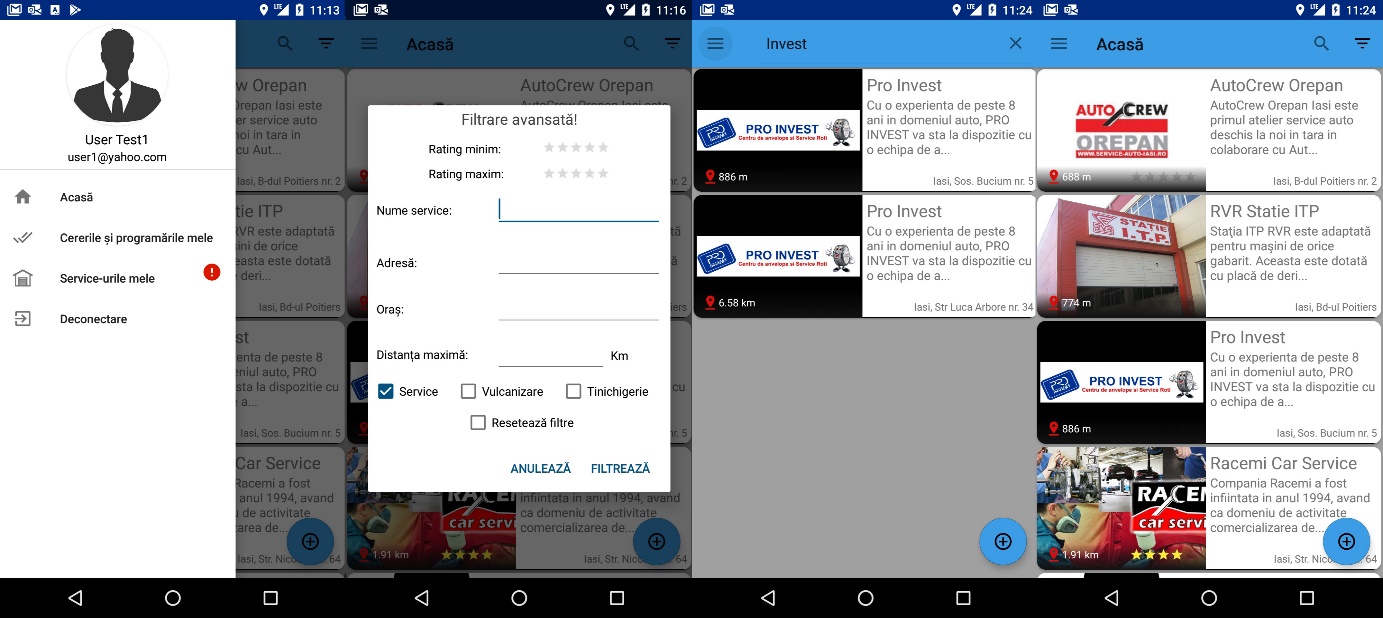
## Funcționalitățile și implementarea lor

În continuare, la acest punct, voi descrie funcționalitățile aplicației și cum am folosit tehnologiile mai sus prezentate în implementarea lor.

### Afișarea unei liste de service-uri

Această primă funcționalitate reprezintă principala funcționalitate a aplicației, aceea de a expune utilizatorilor o listă de service-uri. În realizarea acesteia au intervenit mai multe componente atât din android cât și de pe partea server-ului scris în python. Astfel, pentru a afișa această listă am creat activitatea HomeActivity, această activitate fiind și cea lansată la deschiderea aplicației. Ea este un container ce permite afișarea datelor oferite de un fragment (despre acestea voi vorbi imediat). Responsabilitatea sa este de a oferi o bară de instrumente ce conține un buton de meniu. Acel buton va deschide dinspre stânga spre dreapta un meniu ce afișează în partea de sus profilul utilizatorului conectat (dacă este unul) iar sub acesta sunt afișate butoanele de navigare între fragmente. Tot HomeActivity verifică dacă îi este permis aplicației să folosească datele de localizare, iar în caz negativ va intreba utilizatorul dacă permite ca de acum să folosească aceste date. Deoarece această activitate este doar un container pentru diferite fragmente, ea va trebui ca inițial să încarce un fragment, astfel acel fragment este HomeFragment.

HomeFragment este efectiv componenta ce oferă lista de service-uri, însoțită de câteva detalii despre fiecare service în parte. Pentru aceasta, în cadrul acestui fragment, am folosit un RecyclerView[[9]](#footnote-9), ArrayList de ServiceAuto (o clasă model proiectată de mine pentru reținerea service-urilor) și un adaptor personalizat. Primul lucru făcut de acest fragment este de a cere de la server primele 11 service-uri, cele mai apropiate (în caz că utilizatorul permite aplicației să folosească datele de locație) și le va insera în ArrayList, apoi cu ajutorul adaptorului se va face legătura dintre elementele din ArrayList și *view-urile* oferite de RecyclerView. În momentul în care utilizatorul a ajuns la sfârșitul listei, va apăre un buton în josul pagini pentru a încărca mai multe service-uri, dacă există. Tot în acest fragment am folosit un FloatingActionButton pentru adăugare unui nou service în aplicație. Aplicația permite și afișarea service-urilor filtrate după un anumit criteriu, dar și a service-urilor utilizatorului curent conectat (dacă are service-uri în aplicație). Deoarece aceste două funcționalități sunt similare cu afișarea tuturor service-urilor, am decis să le implementez tot în acest fragment. Asta înseamnă că în momentul în care utilizatoru apasă în meniu pe butonul ce ar trebui să afișeze doar service-urile lui, va fi chemat tot acest fragment, doar că el va cere de la server doar service-urile utilizatorului conectat. Pentru filtrare mecanismul este similar. Pe lângă această filtrare, mai există și un search bar care filtrează lista de service-uri în timp ce utilizatorul tastează, făra a fi necesar apăsarea butonului de confirmare. Dar această filtrare se aplică doar pe intrările deja aduse de pe server, adică dacă pe server sunt intrări ce se potrivesc cu textul dat prin search bar, dar acele intrări nu-s aduse în ArrayList-ul din HomeFragment, atunci acele intrări nu vor fi afișate.



Figură 5 - Imagini din aplicație cu lista de service-uri și meniul principal

### Vizualizarea detaliilor despre un service

Pentru această funcționalitate am creat o nouă activitate, ServiceAutoActivity, care este chemată în momentul apăsării pe un service. Din moment ce toate informațiile despre service-urile afișate sunt aduse de pe server atunci când se construiește ArrayList-ului din HomeFragment, la crearea acestei activități va fi necesar doar trimiterea lor din fragment pe această activitate. Pentru a chema această activitate și pentru a-i trimite totodată datele despre service-ul selectat am folosit un Intent[[10]](#footnote-10).

|  |  |
| --- | --- |
| După cum se poate vedea în imaginea alăturată, activitatea mai sus menționată, oferă toate detaliile despre service-ul selectat. Primul lucru pe care îl putem vedea este bara de instrumente ce afișează denumirea service-ului urmat de adresa acestuia, derulându-se de spre stânga. Apoi tot pe această bară putem observa în stânga butonul de „înapoi”, iar în dreapta se află un buton de optiuni, care afișează o listă de opțiuni, acestea fiind: posibilitatea de a edita service-ul, de a vedea comentariile, programările pe ziua curentă și toate cererile de ofertă realizate către acest service, în caz că utilizatorul conectat este și proprietarul service-ului. În caz contrar, acel buton va oferi o altă listă de opțiuni, prin care se va putea cere o ofertă, face o programare, lăsa un comentariu sau vizualizarea tuturor comentariilor. | C:\Users\Iulian\Desktop\SSApp\Screenshot_1580682753.png  Figură 6 – Imagine cu informațiile complete despre un service |

Sub bara de instrumente se află sigla service-ului însoțită de câteva detalii, cum ar fi distanța față de locația curentă, numărul de stele calculat pe baza părerilor oferite de ceilalți utilizatori, numărul de telefon și adresa de email. Pentru a evita neplăcerea de a nu vedea toate aceste informații din cauza spațiului insuficient, aceste elemente au fost introduse întru-un HorizontalScrollView, element care după nume ne dăm seama ca permite derularea orizontală a elementelor. După această mică secțiune de informații urmează tipul de servicii oferite (servicii de mecanică, de vulcanizare, de tinichigerie, se poate efectua ITP-ul), mărcile de mașini acceptate de acest service, prețurile pentru fiecare serviciu în parte și descrierea propriu-zisă a service-ului. Toate aceste elemente aflate sub bara de instrumente au fost introduse într-un ScrollView pentru a evita ascunderea unei părți dintr-o descriere mai lungă. Pentru un utilizator ce intră pe pagina de descriere a unui service, dar nu este proprietarul acelui service, în partea de jos a pagini, se va afla un buton care la apăsarea lui va afișa încă trei butoane, unul pentru a suna service-ul, unul pentru a scrie un email service-ului, iar al treilea buton este pentru a calcula o rută către service din locația curentă, cu ajutorul aplicației Google Maps.

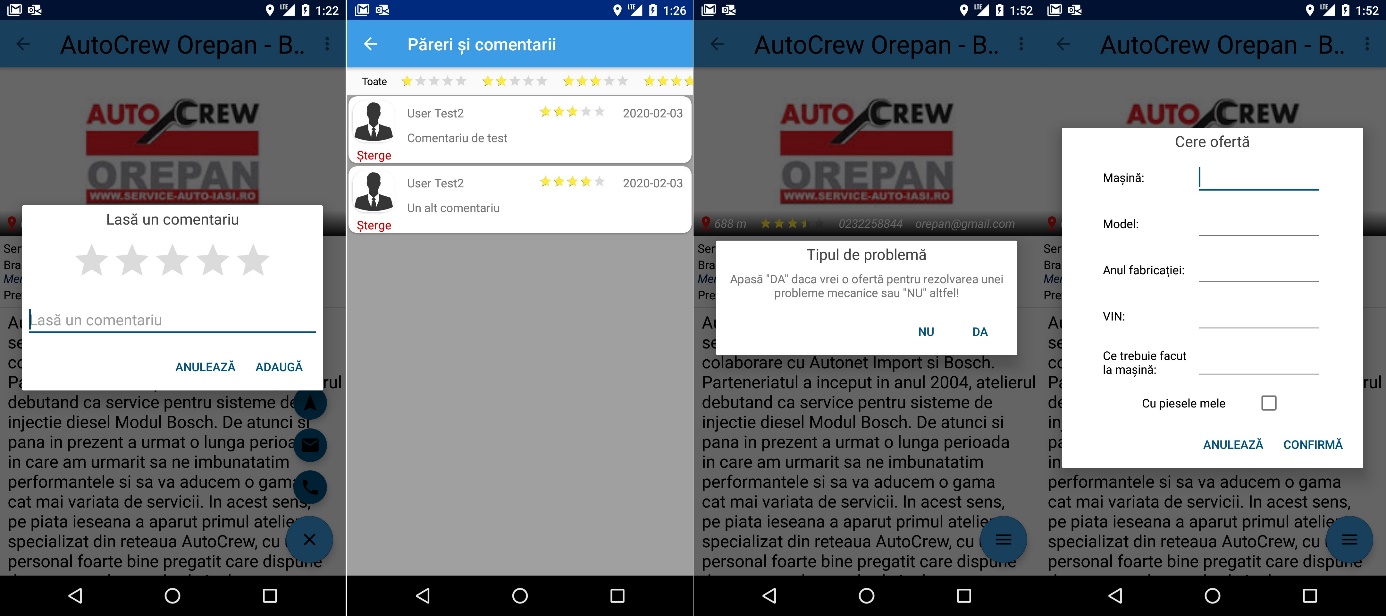
### Apelarea, trimiterea unui email și calcularea unei rute

După cum am zis mai sus, aceste trei funcționalități, se pot executa din activitatea ServiceAutoActivity. Pentru a realiza apelarea unui număr de telefon am folosit un Intent care lansează aplicația de telefonie a telefonului, și îi atașează ca mesaj numărul de telefon al service-ului. Pentru a trimite un email procesul este similar, doar că aici întâi este întrebat utilizatorul ce aplicație dorește să folosească pentru trimiterea email-ului. Iar pentru calcularea rutei se va lansa aplicația Google Maps, iar mesajul atașat va conține un URL de forma: <http://maps.google.com/maps?saddr=47.12412,28.12421&daddr=47.725511,27.12451>, unde coordonatele (47.12412, 28.12421) reprezintă punctul de plecare iar (47.725511, 27.12451) cel de sosire.

### Cererea unei oferte, programarea și lăsarea unui comentariu

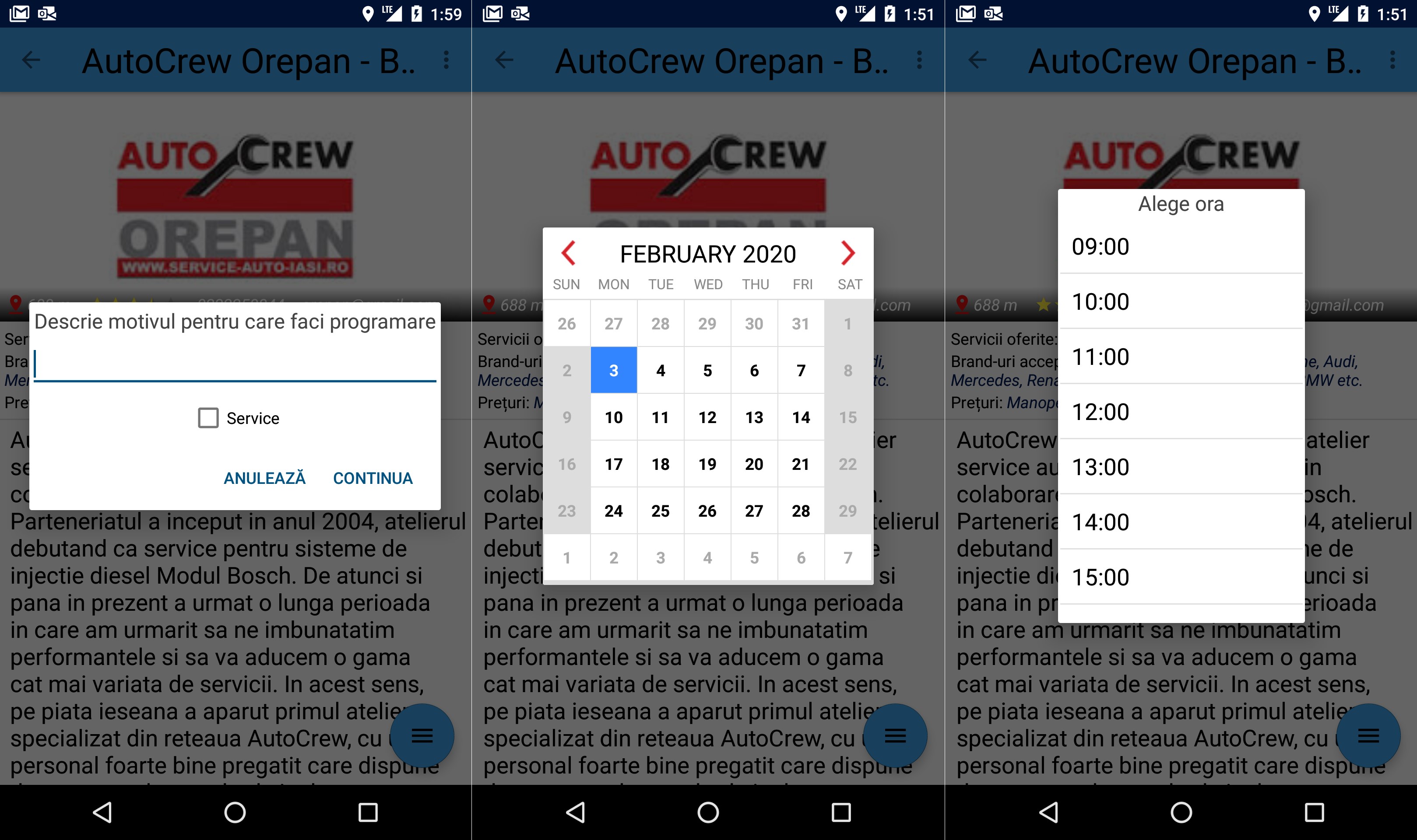
Aceste trei funcționalități pot fi executate de către utilizator de pe pagina de descriere a unui service, doar în cazul în care service-ul nu este al utilizatorului. Deasemenea butoanele responsabile de aceste funcționalități vor fi disponibile și utilizatorilor ce nu-s conectați la aplicație, iar dacă apasă pe unul din ele vor fi întrebați dacă vor să se conecteze. La apăsarea butonului „Lasă un comentariu” aplicația va afișa un *pop-up* prin care îi va cere utilizatorului un comentariu și un număr de stele, între 1 și 5. Acest *pop-up*, dar și restul create în această aplicație sunt realizate cu ajutorul clasei AlertDialog oferită de android. După apăsarea butonului de confirmare oferit de *pop-up*, aplicația va verifica faptul că utilizatorul nu a lăsat câmpuri goale, iar în cazul pozitiv datele vor fi trimise către server cu scopul de a fi introduse în baza de date. Serverul va primi aceste date, iar după ce le va introduce în baza de date va modifica și nota service-ului pe baza notelor existente și a notei abia adăugate. După primirea confirmării că noul comentariu a fost adăugat cu succes, utilizatorul va fi redirecționat pe o nouă pagină ce afișează toate comentariile service-ului. La realizarea acestei pagini am creat o nouă activitate denumită CommentsActivity, un adaptor nou, CommentsAdapter și clasa model Comment. Pe același principiu descris mai sus, în CommentsActivity am creat un RecyclerView și un ArrayList de obiecte de tipul Comment, iar cu ajutorul adaptorului fac legătura dintre elementele din ArrayList și RecyclerView. Tot pe această pagină utilizatorul are posibilitatea de a filtra comentariile după numărul de stele, această funcționalitate fiind executată prin apăsarea unui element oferit de *view-ul* de sub bara de instrumente și de a șterge un comentariu adăugat de el.

La realizarea unei cereri de ofertă dacă service-ul oferă diferite servicii printre care și mecanice, se va deschide un *pop-up* care întreabă utilizatorul dacă vrea să facă o cerere pentru o problemă mecanică. În caz că raspunsul este pozitiv se va deschide un al doilea *pop-up* care cere câteva date despre mașină iar ultimul element din acest *pop-up* este un checkbox de alegere dacă reparația se va face cu piesele clientului. Pentru răspuns negativ va fi afișat același *pop-up* doar că checkbox-ul nu va mai fi prezent. Dacă service-ul ales nu oferă servicii mecanice va apărea direct al doilea *pop-up* iar chekbox-ul nu va fi prezent. După apăsarea butonului de confirmare datele vor fi trimise spre server pentru adăugarea lor în baza de date.



Figură 7 – Adăugarea unui comentariu în primele două imagini și cererea unei oferte în ultimele două imagini

Pentru programarea la un service, utilizatorul va trebui să acceseze același meniu oferit de pagina de descriere a service-ului și să aleagă butonul de programare. Apoi se va deschide un *pop-up* unde trebuie completat cu motivul pentru care se face progrmaarea la service și aleasă căsuța corespondentă serviciului dorit din cele oferite de service. După apăsarea butonului de confirmare aplicația va afișa calendarul oferit de biblioteca Caldroid, de une utilizatorul va alege o dată disponibilă din calendar, apoi i se va afișa o lisă cu orele disponibile în acea zi. După alegere orei datele vor fi trimise spre server pentru adăugarea programării.



Figură 8 – Realizarea unei programări

## Implementarea modulelor

În continuare, la acest punct, voi descrie cum am implementat funționalitățile de bază și cum am folosit tehnologiile mai sus prezentate.

### Autentificarea și schimbarea datelor de profil

Pentru acest modul, în android, am creat activitățile *AuthenticationActivity.java* și *HomeActivity.java* (aceasta va fi menționată și în modulul Pagina de *Acasă* și vizualizarea service-urilor de la pagina 18), o interfață *Authentication.java* și două clase, *Login.java* și *Register.java* ce implementează această interfață. Pentru această funcționalitate, activitatea se ocupă de afișarea a două butoane (cu ajutorul layout-ului său), unul pentru conectare și unul pentru înregistrare, și de lansarea execuției acțiunii cerute de utilizator. Acțiunea este lansată la apăsare unui buton din cele de mai sus cu ajutorul funcției *performAuth()*, ce este suprascrisă in ambele clase, *Login* și *Register*. La apăsarea butonului de conectare se va apela funcția suprascrisă în *Login* și va afișa un pop-up care cere email-ul și parola, iar la apăsarea butonului de înregistrare se va apela funcția suprascrisă în *Register* ce va cere email-ul, numărul de telefon, numele complet al utilizatorului, o parolă și opțional o imagine. La încercarea de a alege o imagine de profil din galerie, aplicația va cere drepturi de accesare a galeriei, abia după acceptul uutilizatorului, se va deschide galeria. La apăsarea butonului de confirmare oferit de pop-up, fie pentru conectare sau înregistrare, mai întâi se va verifica, prin intermediul funcției *verifyInputOnClientSide()*, dacă câmpurile obligatorii nu sunt goale, în caz contrar se va seta un mesaj de eroare, abia după această verificare se va apela funcția *verifyInputOnClientSide()*, care trimite datele către server iar în cazul unui răspuns pozitiv de la server, va redirecționa utilizatorul către pagina de acasă și va salva pe telefon, în fișierul de persistență creat cu ajutorul clasei *SharedPreferences* din android, link-ul de descărcare al imagini, numele utilizatorului și alte date întoarse de server, folosite de aplicație când va fi deschisă din nou. Schimbarea datelor de profil se poate realiza de pe activitatea *HomeActivity*, accesând meniul aplicației și apăsând pe zona de header a meniului (zona unde se află imaginea de profil, numele utilizatorului și emailul). La apăsarea acelei zone, un pop-up se va deschide și va oferi utilizatorului să-și modifice datele introduse la înregistrare. După apăsarea butonului de confirmare pus la dispoziție de pop-up, procesul de verificare a datelor este similar ca la înregistrare, singurul lucru în plus fiind faptul că aici, pe lângă datele utilizatorului se va trimite către server și token-ul de identificare.

Pe server se află două fișiere *user.py* și *api\_auth.py*, primul fiind modelul cu ajutorul cărui framework-ul SQLAlchemy realizează operațiunile asupra bazei de date, iar al doilea este fișierul ce conține rutele aferente acestui modul. Aceste rute fiind:

* */login* => verifică datele primite de la utilizator cu baza de date și crează un token pe care îl întoarce alături de datele utilizatorului în format JSON însoțit de codul de eroare 200, în caz de concordanță cu baza de date, altfel va întoarce codul de eroare 401 și JSON aferent;
* */register* => introduce datele primite de la utilizator în baza de date, dacă emailul nu este deja în baza de date și returnează un token și datele utilizatorului la fel ca login-ul;
* */auth/changeProfile* => această metodă modifică datele utilizatorului doar dacă tokenul primit este valid, tokenul este verificat cu ajutorul decoratorului *check\_token* implementat înfișierul *api\_check.py*. Acest decorator verifică token-ul primit, iar în caz de succes se va execut funcția rutei apelate, altfel va întoarce codul de eroare 401 și JSON aferent
* */auth/checkLogged* => verifică dacă tokenu-ul unui client este valabil. La fel ca ruta destinată schimbarii profilului, este decorată de *check\_token* iar în caz de succes funcția acestei rute va returna JSON-ul *"response": "Token OK"* însoțit de codul de eroare 200;

### Pagina de *Acasă* și vizualizarea service-urilor (service-urile mele și toate service-urile)

Acest modul reprezintă pagina cu care se deschide aplicația, pagina de vizualizare a service-urilor deținute de utilizatorul curent, dacă deține măcar unul, pagina ce conține descrierea completă alături de toate informațiile despre un service și meniul ce permite navigarea între pagini. În android, acest modul este realizat de următoarele fișiere:

***HomeActivity.java*:** Activitate cu care se deschide aplicația, aceasta va verifica dacă utilizatorul a permis aplicației să utilizeze locația, în cazul negativ, întreba utilizatorul dacă permite, iar dacă raspunsul este pozitiv înregistrează un listener prin care aplicația va fi notificată de schimbările de locație, apoi va lansa fragmentul HomeFragment și va crea meniul de navigare între pagini. Acest meniu este reprezentat de butonul care duce pe pagina service-urilor utilizatorului curent, butonul care duce spre vizualizarea programărilor și a cererilor de ofertă dar, butonul prin care te poți întoarce pe pagina de acasă și butonul de deconectare, pentru utilizatorii conectați, iar pentru cei neconectați este reprezentat de butonul de acasă și de butonul de autentificare.

***HomeFragment.java*:** Acest fragment reprezintă pagina de acasă a aplicației. Este realizată dintr-un RecyclerView[[11]](#footnote-11)(implicit și un ArrayList), un FloatingActionButton și bara de meniu ce conține un SearchView și un buton de filtrare. La inițializarea acestui fragment se va crea URL-ul care cere primele 11 service-uri ordonate după distanța față de locația curentă, iar prin intermediul funcției *populateServiceAutoList()* se va folosi acest URL, apoi pentru fiecare id va cere service-ul aferent. Fiecare service se va adăuga în ArrayList-ul de mai sus, urmând a fi afișate in cadrul layout-ului RecyclerView, afișarea făcându-se cu ajutorul clasei ServiceAutoAdapter. Dacă în timpul derulării prin lista de service-uri, utilizatorul a ajuns aproape de finalul listei, în josul pagini va apărea un buton ce odată apăsat va aduce înca 11 service-uri sau câte au rămas daca sunt mai puține. FloatingActionButton-ul are rolul de a deschide un pop-up pentru adăugarea de service-uri, acest pop-up conține câteva view-uri de tip EditText care trebuiesc completate cu detaliile despre noul service, un ImageView care la apăsarea lui va deschide galeria dacă aplicația are permisiuni de accesare, și un MapView ce încarcă harta oferită de Google. Această hartă are rolul de a da coordonatele service-ului ce va fi adăugat prin intermediul unui marker ales de utilizator. La fel ca și la modulul precedent, la apăsarea butonului oferit de acest pop-up, datele introduse vor fi mai întâi verificate daca sunt completate și apoi trimise spre server alături de token. SearchView-ul are rolul de a face o căutare a service-urilor după denumire, adresă și oraș în timpul tastării, pentru această funcționalitate am setat un listener pe SearchView, iar la fiecare schimbare de text sau la submiterea textului, se va „cere” de la ServiceAutoAdapter lista de service-uri ce îndeplinesc condițiile, implementarea propriuzisă a căutarii fiind făcută în fișierul ServiceAutoAdapter. Ultimul element din acest fișier, butonul de filtrare, este responsabil, după cum reiese și din denumirea lui, de aducerea de pe server unei liste de service-uri după anumite criterii, acestea fiind numărul de stele, distanța față de locația curenta, orașul în care se află, adresa și tipul. Preluarea acestor date de la utilizator se va realiza cu ajutorul unui pop-up. Pentru a realiza această căutare pe partea de server, URL-ul care cere primele 11 service-uri va fi modificat astfel încât să ceară primele 11 service-uri ce indeplinesc criteriile date de utilizator, apoi apelându-se din nou funcția *populateServiceAutoList()*. Deasemenea este responsabil și de afișarea service-urilor utilizatorului conectat, la apăsarea butonului din meniul principal. Afișarea este realizată la fel ca pentru toare service-urile și pentru service-urile filtrate, singura diferență fiind modificare URL-ului care cere id-urile service-urilor. Aici se definește și se dă adaptorului listener-ului ce este chemat la selectarea unui element din listă, dar și a listener-ului responsabil de ștergerea unui service la apăsarea butonului de ștergere de către proprietar.

***ServiceAutoAdapter.java*:** Aceasta clasă extinde *RecyclerView.Adapter* și este un adaptor pentru a putea lega elementele ArrayList-ului din HomeFragment de view-urile din RecyclerView. Pentru asta, clasa are nevoie de ArrayList-ul cu service-uri, iar pe măsură ce un nou view este cerut, prin intermediul funcției *onBindViewHolder()*, sunt setate informațiile care trebui afișate, tot în această clasă fiind setați și litenerii pentru selectarea și ștergere unui element, pentru fiecare view, primiți din HomeFragment. O altă responsabilitate a acestui adaptor, după cum am zis și la punctul anterior, este de a face filtrarea după textul primit de la SearchView-ul din HomeFragment.

***ServiceAuto.java*:** Această clasă reprezintă tipul de elemente aflate în ArrayList-ul aflat în HomeFragment și trimis mai departe adaptorului. Deasemenea, pe baza unui obiect de acest tip se face popularea activității ServiceAutoActivity.

***ServiceAutoActivity*.*java*:** Pe această pagină se poate ajunge prin apăsarea pe un service. Listener-ul setat în HomeFragment ce este chemat la apăsare pe un service va seta un Intent[[12]](#footnote-12) ce lansează ServiceAutoActivity și va pune ca mesaj toate obiectele membre a instanței de tipul ServiceAuto corespondentă service-ului pe care s-a apăsat. La crearea activități se va prelua din acel Intent informația și se va stoca într-un obiect membru pentru popularea acestei activități folosiri ulterioare. Majoritatea acțiunilor ce pot fi făcute de aplicație fie sunt pornite, fie sunt implementate în această activitate. Astfel cu ajutorul meniului oferit de activitate, dacă service-ul ce populează activitatea este al utilizatorului conectat, se vor putea lansa activitatățile ce afișează comentariile pentru acest service, cererile de ofertă pentru acest service, programările pe ziua curentă, acestea făcând parte din alte module le voi discuta acolo, și se va putea edita service-ul, această funcționalitate fiind implementată aici sub forma unui pop-up similar cu cel de adăugarea unui service nou. Dacă service-ul nu este al utilizatorului conectat, acțiunile posibile vor fi: vizualizare de comentarii, lăsarea unui comentariu, cererea unei oferte, programarea la o anumită oră, crearea unei rute de la locația curentă la service, apelarea service-ului și trimiterea unui mail. La fel ca mai sus, vizualizare de comentarii se face prin crearea unui Intent ce lansează activitatea specifică. Lăsarea unui comentariu este implementată în această activitate sub forma unui pop-up care cere doar comentariul și numărul de stele, celelalte informații necesare fiind preluate din SharedPreferences. Cererea de ofertă și programarea sunt implementate aici, dar voi vorbi despre ele la modulul Cererea unei oferte și Programarea la un service de la pagina 22. Crearea unei rute este realizată prin lansarea aplicației Google Maps cu ajutorul unui intent, acest intent primind ca mesaj[[13]](#footnote-13) un URL ce conține coordonatele de pornire și de finalizare a rutei. Apelarea și trimiterea unui mail se va realiza similar cu creare rutei, diferența fiind aplicațiile ce vor fi lansate si mesajul trimis, pentru apelare fiind lansată aplicația telefonului de iar ca mesaj va fi setat numărul de telefon, iar pentru trimiterea unui mail va întreba utilizatorul prin ce aplicație de mesagerie instalată pe telefon vrea să trimită mail-ul, iar mesajul setat în intent este adresa de mail.

La fel ca și la modulul anterior pe server se află 2 fișiere: *service.py* și *api\_service.py*, primul fiind modelul iar al doilea conține rutele apelate pentru acest modul. Rutele din acest fișier sunt următoarele:

* */service/getIdsBetween/offset/<offset>/limit/<limit>/latitude/<latitude>/longitude/<longitude>*

Această rută este responsabilă de furnizarea a „limit” id-uri de service-uri, în cazul nostru limit fiind 11, ordonate după distanța față de perechea de coordonate primită prin latitude și longitude, începând cu poziția offset. Dacă numărul total de intrări este mai mic ca limit se vor returna câte sunt. Pentru a menține aceeași ordine între două apeluri ale rutei, de exemplu cand offset-ul este 0 și când este 11, ordonarea ambelor apeluri se va face după coordonatele primite la apelul rutei când offset-ul a fost 0, coordonatele primite la apelul cu offsetul 11 sunt ignorate. Id-urile sunt returnate sub forma unui JSON cu cheia id iar valoare este lista de id-uri.

* */services/getIdsBetweenForMyServices/<userId>/offset/<offset>/limit/<limit>*

La fel ca ruta de mai sus, întoarce câte „limit” id-uri de service-uri, diferența fiind faptul că nu le mai ordonează, dar le filtrează pe baza id-ului utilizatorului astfel încat toate id-urile returnate să corespundă service-urilor introduse de utilizatorul cu id-ul userId. Pentru a putea apela ruta, pe android trebuie setat în header token-ul de autentificare, deoarece ruta este decorată cu funcția check\_token.

* */services/getIdsBetweenWithFilter/offset/<offset>/limit/<limit>/minRating/<minRating>/maxRating/<maxRating>/name/<name>/address/<address>/city/<city>/type/<serviceType>/maxDistance/<maxDistance>/latitude/<latitude>/longitude/<longitude>*

Această rută întoarce tot o listă de „limit” id-uri de service-uri, limit fiind tot 11, lista de unde se selectează intervalul [offset, offset+limit) fiind realizată printr-o filtrare după parametrii primiți în rută. Deasemenea este decorată de check\_token.

* */services/getById/<serviceId>*

La apelarea acestei rute, serverul va căuta în baza de date service-ul cu id-ul serviceId și îl va returna sub forma unui JSON.

* */services/addService*

Prin această rută serverul adaugă un nou service în baza de date doar dacă token-ul primit în header este decodificat cu succes de decoratorul check\_token. Ruta este apelată prin metoda POST, iar datele sunt timise din prin intermediul unui JSON setat în android și citit la momentul apelării rutei, prin intermendiul obiectului request, oferit de flask.

* */service/deleteById/<serviceId>*

Ruta aceasta este responsabilă de ștergerea service-ului din baza de date, dar mai întâi se asigură ca toate dependințele către acest service au fost șterse. Deasemenea și această funcție este decorată cu check\_token pentru a nu permite ștergerea datelor fără o autorizare.

* */service/editService*

După cum se înțelege din denumirea sa, această ruta se ocupă de modificarea datelor despre un anumit service. Ruta este apelată cu ajutorul metodei PUT iar datele sunt transmise la fel ca și în cazul metodei POST, prin intermediul obiectului request. Ca și ruta precedentă, funcția sa este executată doar daca este oferit un token valid.

### Cererea unei oferte și Programarea la un service

Acest modul cuprinde toate funcționalitățile aferente unei cereri de ofertă și unei programări, adică cuprinde atât funcționalitățile oferite unui utilizator simplu dar și cele oferite unui proprietar de service. La realizare acestui modul ,în android, intervin următoarele, următoarele fișiere:

**ServiceAutoActivity.java:** Este fișierul care l-am descris și la modulul precedent, aici voi vorbi doar despre realizarea unei programări și cererea unei oferte. Pentru oricare din cele două funcții se va accesa meniul acestei pagini și se va alege de acolo acțiunea. Dacă este ales butonul de cerere de ofertă, iar service-ul oferă diferite servicii printre care și mecanice, se va deschide un pop-up care întreabă utilizatorul dacă vrea să facă o cerere pentru o problemă mecanică. În caz că raspunsul este pozitiv se va deschide un al doilea pop-up care cere câteva date despre mașină iar ultimul element din acest pop-up este un checkbox de alegere dacă reparația se va face cu piesele clientului. Pentru răspuns negativ va fi afișat același pop-up doar că checkbox-ul nu va mai fi prezent. Dacă service-ul ales nu oferă servicii mecanice va apărea direct al doilea pop-up iar chekbox-ul nu va fi prezent. La apăsarea butonului de programare se va deschide un pop-up în care se va introduce motivul pentru care se face programarea, după apăsarea butonului de confirmare aplicația va afișa calendarul oferit de librăria Caldroid, de une utilizatorul va alege o dată disponibilă din calendar, apoi i se va afișa o lisă cu orele disponibile în acea zi. După alegere orei datele vor fi trimise spre server pentru adăugarea programării.

**OfferRequestsActivity.java:** Această activitate, după cum am spus și la modulul anterior, este lansată din meniul activității ServiceAutoActivity și are rolul de a expune proprietarului service-ului toate cererile de ofertă primite de acel service. Pentru a oferi toate cererile de ofertă am folosit tot un RecyclerView cu un ArrayList de obiecte de tipul RequestOffer și adaptorul RequestOfferAdapter. Cererile de ofertă sunt aduse de pe server la fel ca service-urile, adică întâi id-urile apoi fiecare cerere după id, doar că în acest caz se cer câte 21 de id-uri de cereri, nu 11 ca în cazul service-urilor, această acțiune fiind făcută în funcția *populateRequestList()*. Alt lucru realizat aici este definirea listener-ilor de acceptare a ofertei, de refuzare a ofertei, de ștergere, dar și listener-ul ce va acționa la apăsarea pe numărul de telefon al persoanei ce a făcut cererea, afișat de layout-ul cererii, și un listener ce va seta oferta ca fiind văzută de proprietarul de service. Apoi, după definirea lor, vor fi setați adaptorului care îi va seta fiecărui element din listă. Listener-ul de acceptare presupune afișarea unui pop-up, unde proprietarul va introduce perioada în care se poate repara mașina și prețul, iar cel de refuzare presupune afișarea unui pop-up care cere motivul pentru care service-ul refuză cererea de ofertă. Pentru a șterge o cerere, proprietarul service-ului trebuie mai întâi să răspundă cererii, abia apoi i se va permite.

**RequestOfferAdapter.java:** Acest adaptor are rolul de a seta datele cererilor de ofertă din ArrayList în view-urile RecyclerView-ului și de a seta fiecărui element din fiecare view listener-ul corespondent, primit de la activitatea OfferRequestsActivity sau de la fragmentul MyRequestFragment.

**TodayAppointmentsActivity.java:** Activitatea este pornită în momentul în care proprietarul service-ului accesează butonul „Programări pe azi” din meniul service-ului. Implementarea acestei activități este puțin diferită față de celelalte care afișau o listă de elemente, diferența fiind că aici pot fi maxim 8 elemente, iar din acest motiv am ales să le aduc pe toate in aceeași cerere, restul lucrurilor fiind similare, cum ar fi faptul că se folosește tot un ArrayList și un RecyclerView pentru afișare elementelor din listă și că aici se setează litener-ul ce intervine la apăsarea pe numărul de telefon al persoanei ce a făcut programarea, afișat de layout-ul programării.

**AppointmentsAdapter.java:** Ca și adaptorul pentru cereri de ofertă, acesta are rolul de a seta datele programărilor din ArrayList în view-urile din RecyclerView și de a seta listener-ul primit de la activitatea TodayAppointmentsActivity.

**MyRequestsAndMyAppointmentsFragment.java:** Această clasă este chemată din meniul principal descris în primul modul, și oferă două tab-uri, unul pentru cererile făcute de utilizator iar unul pentru programările făcute de utilizator.

**MyRequestsAndMyAppointmentsAdapter.java:** Adaptorul acesta este diferit de restul folosiți în aplicație, deoarece acesta are rolul de a insera MyAppointmentFragment și MyRequestFragmen în cele două taburi din MyRequestsAndMyAppointmentsFragment.

**MyAppointmentFragment.java** și **MyRequestFragmen.java:** Funcționează pe acelaeași principii ca și activitățile TodayAppointmentsActivity și OfferRequestsActivity, dar aceste două fragmente au fost create la necesitatea de a popula taburile din fragmentul MyRequestsAndMyAppointmentsFragment. MyAppointmentFragment este responsabil de afișarea tuturor programărilor utilizatorului conectat, iar MyRequestFragmen de afișarea tuturor cererilor de oferte făcute utilizatorul conectat.

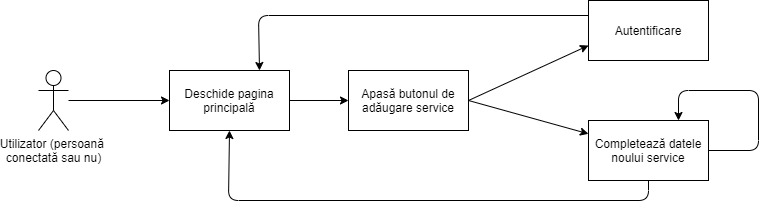
**Appointment.java** și **RequestOffer.java:** Aceste două clase reprezintă modelele pe baza cărora funcționează toate ArrayList-uril menționate în acest capitol.

De acest modul, pe server, se ocupă următoarele fișiere: lockedHour.py, lockedDay.py, requestedOffer.py, api\_lockedPeriod și api\_requestedOffer.py, primele 3 fiind modelele iar celelalte două sunt fișierele ce conțin rutele ce pot fi apelate.

# Scenarii de utilizare

La acest punct voi vorbi despre câteva funcționalități esențiale ale aplicației și voi evidenția pașii prin care trece aplicația, dar și utilizatorul pentru executarea acestor funcționalități cu succes.

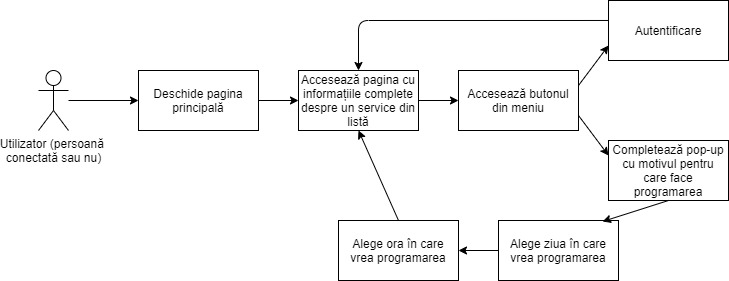
## Adăugarea unui service



Figură 9 - Diagramă use case de adăugare a unui nou service

Primul scenariu reprezintă adăugarea unui service nou în baza de date. Această acțiune este descrisă și din punctul de vedere al unui utilizator ce este conectat la aplicație dar și din punctul unuia neconectat. Astfel primul pas pe care actorul trebui să-l facă este de a deschide aplicația pe pagina principală, aceasta afișândui lista de service-uri deja existente și un buton în josul ecranului de adăugare. Al doilea pas este cel de apăsare al butonului, aplicația verifică dacă utilizatorul este conectat, iar dacă acesta nu este, îl va pune să se conecteze în caz că mai dorește adăugarea unui nou service. După conectarea utilizatorului, al treilea pas este completarea datelor despre noul service. În caz că datele introduse nu sunt complete, utilizatorul va trebui să revizuiască din nou datele. După ce datele introduse sunt complete, aplicația va afișa din nou pagina principală ce va afișa lista de service-uri, care acum conține și service-ul tocmai adăugat.

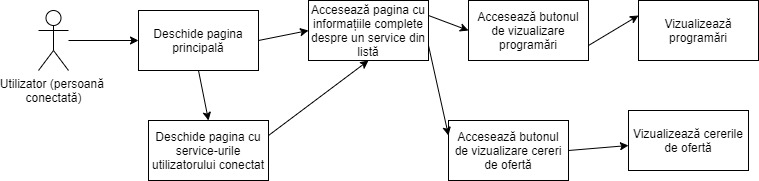
## Programarea la un service și cererea unei oferte



Figură 10 - Diagramă use case de programare la un service

În cel de-al doilea scenariu prezint mudul în care un utilizator poate să se programeze la un service. Și aici, ca și la primul scenariu, utilizatorul poate să fie conectat sau nu. Astfel primul pas este reprezentat de deschiderea aplicației pe pagina principală, după care actorul va trebui să aleagă un service din listă (cu mențiunea că service-ul ales nu trebuie să fie deținut de el) și să apese pe el pentru a deschide pagina ce afișează toate detaliile. Următorul pas este apăsarea butonului din meniu, care la fel, va verifica dacă tilizatorul este conectat, în caz contrar îl va întreba dacă vrea să se conecteze. Ultimii trei pași reprezintă efectiv completarea informațiilor necesare pentru realizarea programării. Alegerea orei reprezină efectiv realizarea programării. Un alt scenariu similar cu acesta este realizarea unei cereri de ofertă, diferențele fiind doar datele cerute de aplicație pentru realizarea unei astfel de cerere. Deasemenea și adăugarea și vizualizarea unui comentariu este similară.

## Vizualizarea cererilor și a programărilor unui service



Figură 11 – Vizualizarea cererilor și a programărilor unui service

În acest scenariu se poate vedea cum un utilizator, ce are un service adăugat în aplicație, poate să vizualizeze ce programări și cereri de ofertă a primit acel service. Primul pas în acest caz este deschiderea aplicației pe pagina principală, apoi de aici, utilizatorul poate alege dacă vrea să caute în toată lista unul din service-urile sale și să-l accesese de aici, sau poate alege să vadă o listă mai restrânsă doar cu service-urile sale, ca mai apoi să-l acceseze de aici. După deschiderea pagini cu descrierea service-ului, va trebui să acceseze meniul ce îi va oferi butoanele de vizualizare a cererilor și a programărilor. După apăsarea unui buton se va deschide pagina ce afișează cererile sau programările, după caz.

## Filtrare service-uri

|  |  |
| --- | --- |
| C:\Users\Iulian\Downloads\Untitled Diagram (25).jpg  Figură 12 – Filtrare service-uri | Acest scenariu mai mic, reprezintă pașii ce trebuie urmați pentru a filtra lista de service-uri. După deschiderea aplicației pe pagina principală, se va accesa butonul din bara de instrumente, după care se va completa un pop-up cu criteriile de filtrare. |

# Concluziile și posibilitățile de dezvoltare

## Concluziile lucrării

În ziua de azi aproape toate persoanele au o mașină, fiecare tânar ajuns la vârsta de 18 ani dorește să aibă o mașina, sunt familii ce dețin câte 2 sau chiar 3 mașini, ceea ce înseamnă că este un număr foarte mare de persoane ce își caută service-ul potrivit pentru mașină. Numărul acesta, foarte ridicat și faptul ca nu de puține ori se întâmplă să fim nemulțumiți de calitatea serviciilor oferite de un service subliniază necesitatea acestor aplicații de găsire a service-ului potrivit.

Consider că aplicația mea poate să rezolve aceste probleme, deasemenea oferă și alte soluții, mai eficiente, de realizare a programărilor și a cererilor de ofertă pe lângă clasicele metode (prin email, telefon sau deplasarea la un service).

## Posibilități de dezvoltare

Deși aplicația oferă diferite servicii, există diferite funcționalități ce ar putea fi integrate în aplicație cu scopul de a o îmbunătăți, următoare listă reprezintă doar câteva idei:

* posibilitatea de a adăuga mai multe imagini pentru un service.
* permiterea utilizatorlor să adauge și o imagine sau mai multe alături de un comentariu
* dezvoltarea unui mecanism de a adăuga un răspuns la un comentariu deja existent
* dezvoltarea unui mecanism ce permite comunicarea prin mesaje text între un utilizator ce caută un service și un utilizator proprietar de service
* reținerea informațiilor, cu permisiunea utilizatorului, despre o mașină în momentul realizării unei cerei de ofertă, astfel la realizarea unei a doua cereri se va putea alege autocompletarea cu datele despre o mașină reținută.
* posibilitatea de a distribui service-uri pe diferite rețele sociale

# Bibliografie

- <https://www.todaysoftmag.ro/article/2642/sisteme-decuplate-folosind-platforma-firebase-de-la-google>

- <https://firebase.google.com/docs/android/setup>

- <https://github.com/koush/ion>

- <https://github.com/roomorama/Caldroid>

- <https://bumptech.github.io/glide/>

- <https://developers.google.com/maps/documentation/android-sdk/intro>

- <https://docs.python.org/3/>

- <https://developer.android.com/docs>

- <http://flask.palletsprojects.com/en/1.1.x/>

- <https://www.sqlalchemy.org/>

- <https://about.draw.io/>

- <https://c4model.com/>

1. Descriere preluată de pe site-ul: <https://www.cylex.ro/> [↑](#footnote-ref-1)
2. <http://service.auto.ro/> [↑](#footnote-ref-2)
3. Android oferă clasa DatePickerDialog pentru a selecta o dată dintr-un calendar [↑](#footnote-ref-3)
4. Android Nougat [↑](#footnote-ref-4)
5. <https://developer.android.com/training/data-storage/shared-preferences> [↑](#footnote-ref-5)
6. Dacă acesta a permis utilizarea locației la intrarea în aplicație [↑](#footnote-ref-6)
7. Harta oferită de Google [↑](#footnote-ref-7)
8. <https://pyjwt.readthedocs.io/en/latest/> [↑](#footnote-ref-8)
9. RecyclerView este un layout de tip listă ce afișează intr-un mod eficient elementele listei, cu acest layout se creează view doar pentru elementele afișate plus un numar de elemente aflate înaite și după elementele afișate, spre deosebire de ListView ce crează view-uri pentru întreaga listă [↑](#footnote-ref-9)
10. Intent-ul este on obiect cu ajutorul căruia se poate cere lansa o nouă activitate componentă a aplicației sau o nouă aplicație, deasemenea poate conține și un mesaj ce îl va transmite activității ce o va lansa [↑](#footnote-ref-10)
11. RecyclerView este un layout de tip listă ce afișează intr-un mod eficient elementele listei, cu acest layout se creează view doar pentru elementele afișate plus un numar de elemente aflate înaite și după elementele afișate, spre deosebire de ListView ce crează view-uri pentru întreaga listă [↑](#footnote-ref-11)
12. Intent-ul este on obiect cu ajutorul căruia se poate cere lansa o nouă activitate componentă a aplicației, deasemenea poate conține și un mesaj ce îl va transmite activității ce o va lansa [↑](#footnote-ref-12)
13. Exemplu de mesaj care creează o rută de la coordonatele (47.12412, 28.12421) la (47.725511, 27.12451) http://maps.google.com/maps?saddr=47.12412,28.12421&daddr=47.725511,27.12451 [↑](#footnote-ref-13)