**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 5](#_Toc31683994)

[1.1. Cylex 6](#_Toc31683995)

[1.2. Auto.ro 7](#_Toc31683996)

[2. Arhitectura 8](#_Toc31683997)

[2.1. Componenta client 8](#_Toc31683998)

[2.2. Componenta server 10](#_Toc31683999)

[2.3. Baza de date 11](#_Toc31684000)

[3. Funcționalitățile aplicației și implementarea lor 13](#_Toc31684001)

[3.1. Afișarea unei liste de service-uri 13](#_Toc31684002)

[3.2. Vizualizarea detaliilor despre un service 14](#_Toc31684003)

[3.3. Apelarea, trimiterea unui email și calcularea unei rute 16](#_Toc31684004)

[3.4. Cererea unei oferte, programarea și lăsarea unui comentariu 16](#_Toc31684005)

[3.5. Vizualizarea cererilor și a programărilor pentru un service 18](#_Toc31684006)

[3.6. Vizualizare cereri de ofertă și programările unui utilizator 19](#_Toc31684007)

[3.7. Adăugarea, editarea și ștergerea unui service 20](#_Toc31684008)

[3.8. Autentificarea și schimbarea datelor de profil 22](#_Toc31684009)

[4. Scenarii de utilizare 25](#_Toc31684010)

[4.1. Adăugarea unui service 25](#_Toc31684011)

[4.2. Programarea la un service și cererea unei oferte 25](#_Toc31684012)

[4.3. Vizualizarea cererilor și a programărilor unui service 26](#_Toc31684013)

[4.4. Filtrare service-uri 26](#_Toc31684014)

[5. Concluziile și posibilitățile de dezvoltare 27](#_Toc31684015)

[5.1. Concluziile lucrării 27](#_Toc31684016)

[5.2. Posibilități de dezvoltare 27](#_Toc31684017)

[6. Bibliografie 28](#_Toc31684018)

[7. Anexe 29](#_Toc31684019)

[7.1. Android 29](#_Toc31684020)

[7.2. Python 30](#_Toc31684021)

# 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 pentru 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 autoturism este acel moment când trebuie să schimbi anvelopele deoarece vine iarna sau vara, iar toate service-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.

În acest moment, pe piața din România 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 dispoziție o întreagă listă de firme, unde poți sa afli diferite informații despre o firma, cum ar fi adresa firmei, numerele de telefon, 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 de asemenea și service-uri auto.

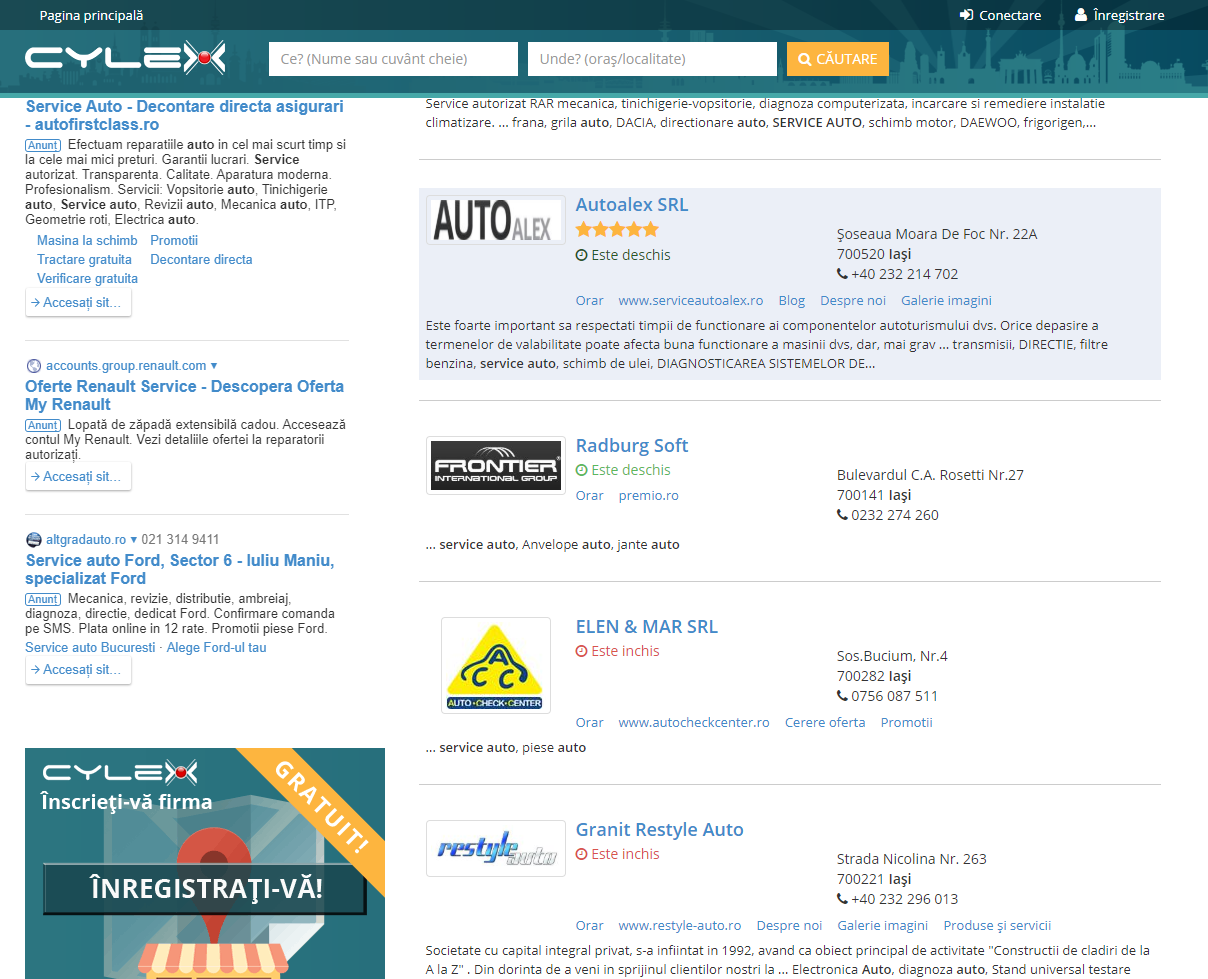


Fig. 1 - Imagine de pe site-ul Cylex

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.

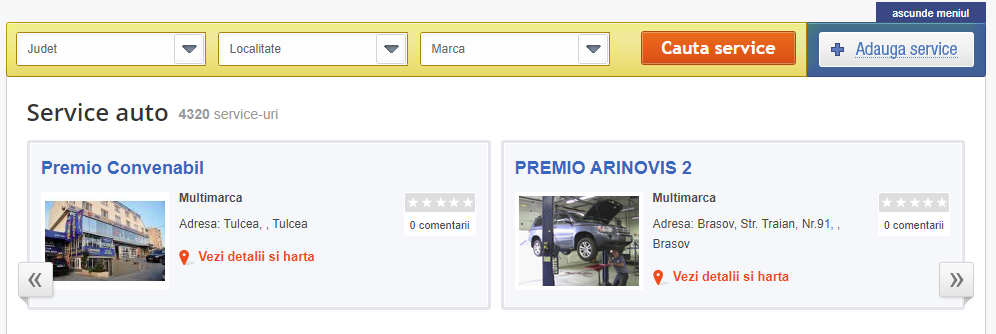


Fig. 2 - Imagine de pe site-ul Auto.ro

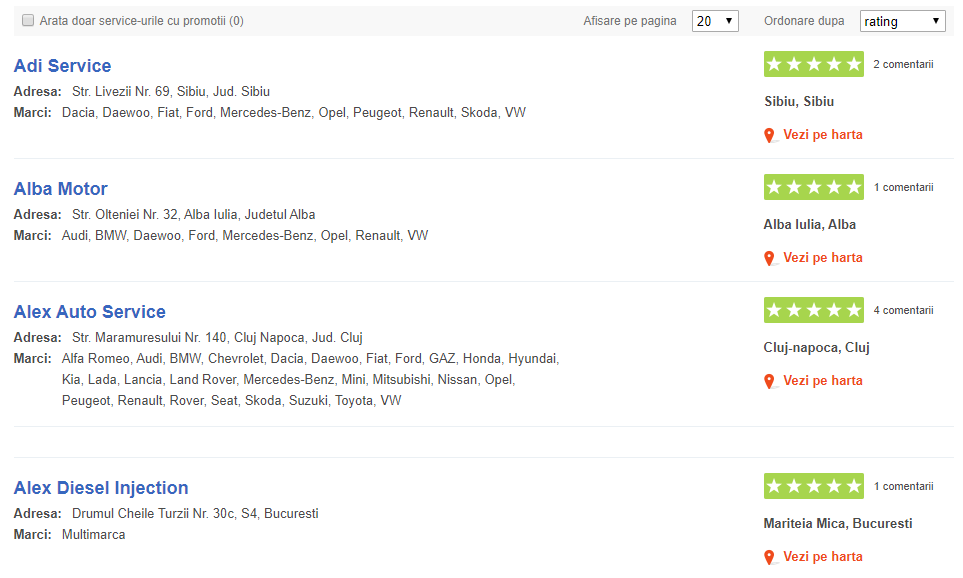


Fig. 3 - Imagine de pe site-ul Auto.ro

# Arhitectura

Din punct de vedere al arhitecturii aplicația se poate împărți în două componente: partea de client scrisă în 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 scrisă pe următoarele versiuni, pentru android am folosit versiunea 7.0[[3]](#footnote-3) iar pentru codul scris în python am folosit versiunea 3.7. Astfel, aplicația nu va rula dacă se va instala pe un telefon ce are o versiune de android sub 7.0 (pe telefoanele cu o versiune mai nouă va rula), iar serverul nu va rula pe versiunea de python 2.x.

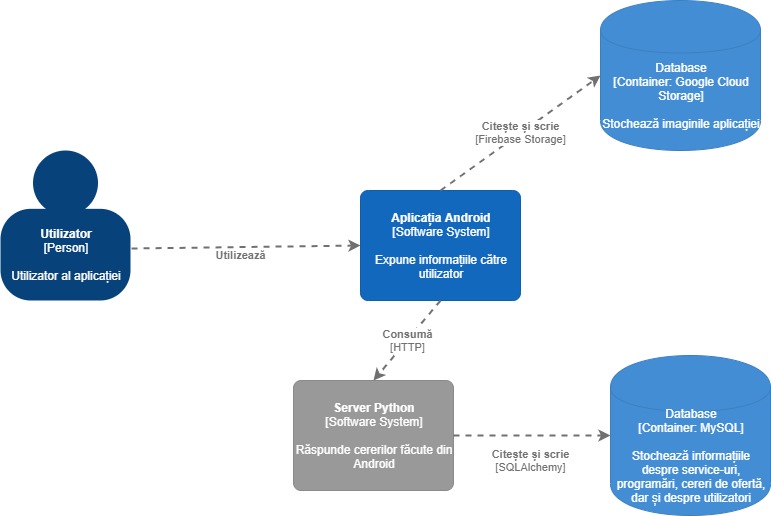


Fig. 4 - Diagrama aplicației ServiceFinder

## Componenta client

Această componentă expune utilizatorului, printr-o interfață grafică, toate funcționalitățile aplicației. Astfel, utilizatorul va avea posibilitatea să intre în aplicație și să aleagă 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 *singletonul* *SharedPreferencesManager* care interacționează cu interfața SharedPreferences oferită de android pentru stocarea datelor într-un format de tipul cheie-valoare.

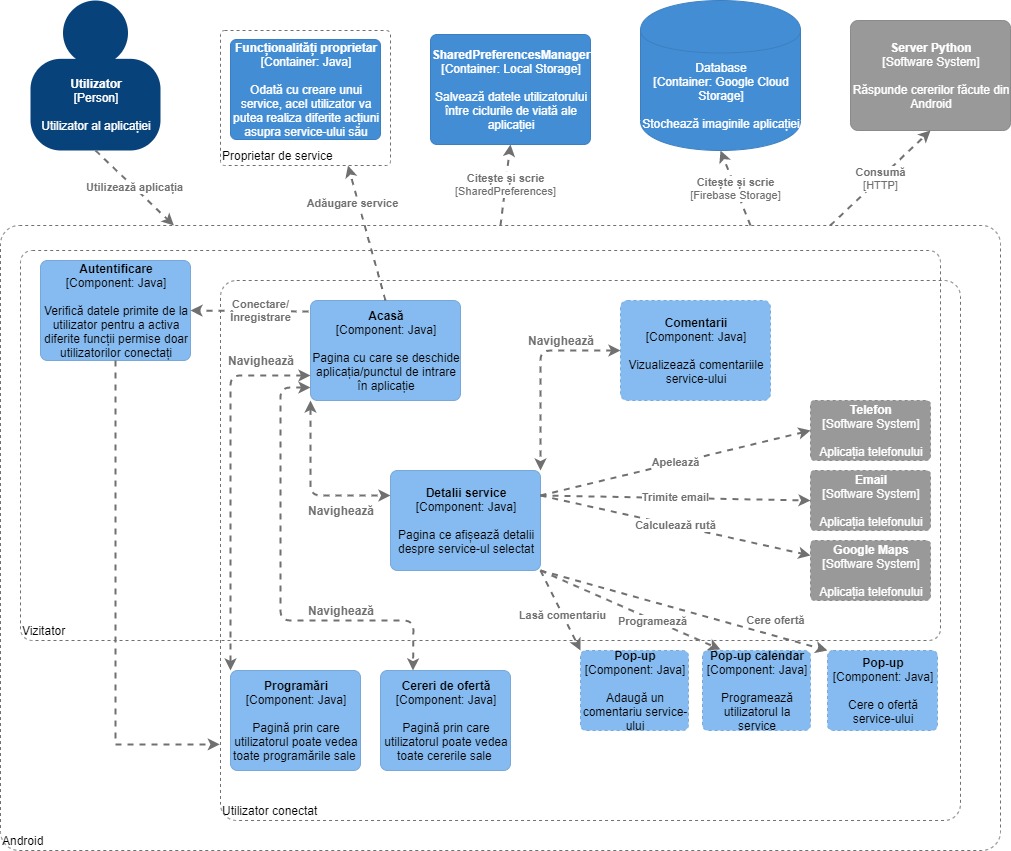


Fig. 5 - Diagrama componentei android (utilizator conectat și vizitator)

După adăugare unui nou service sau a mai multor service-uri, utilizatorul va avea posibilitatea să vadă toate service-urile 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[[4]](#footnote-4) a utilizatorului și a locației introduse de proprietar la crearea service-ului cu ajutorul unei hărți[[5]](#footnote-5).

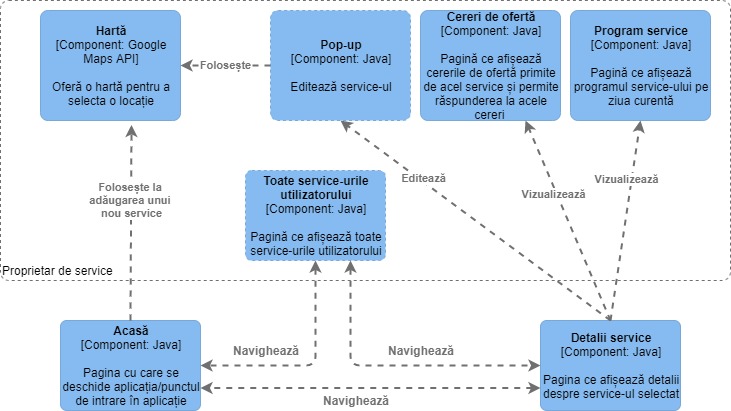


Fig. 6 - Noile funcționalități oferite de aplicație după adăugarea unui service

## 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 rutele oferite de flask pot fi folosite și dintr-un browser, serverul ar trebui să nu permită accesul apelurilor ce nu sunt identificate asupra rutelor ce pot executa operații critice în baza de date. Astfel serverul dispune de un mecanism de verificare a identității implementat cu ajutorul librăriei PyJWT. Acest mecanism presupune, pe partea server-ului, generarea unui token JWT (JSON Web Token) 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). Totodată sunt și rute ce nu necesită un astfel de token, acestea sunt cele prin care se realizează autentificarea și cele ce stau la baza funcționalităților disponibile utilizatorilor neconectați.

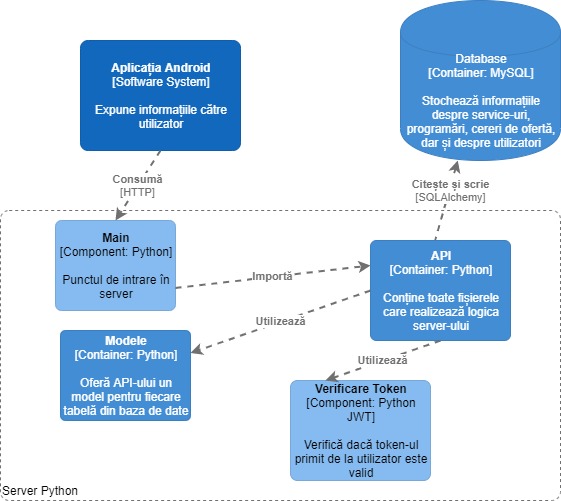


Fig. 7 - Diagrama server-ului

## Baza de date

Pentru stocarea datelor am creat o bază de date Sql. Tabelele 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

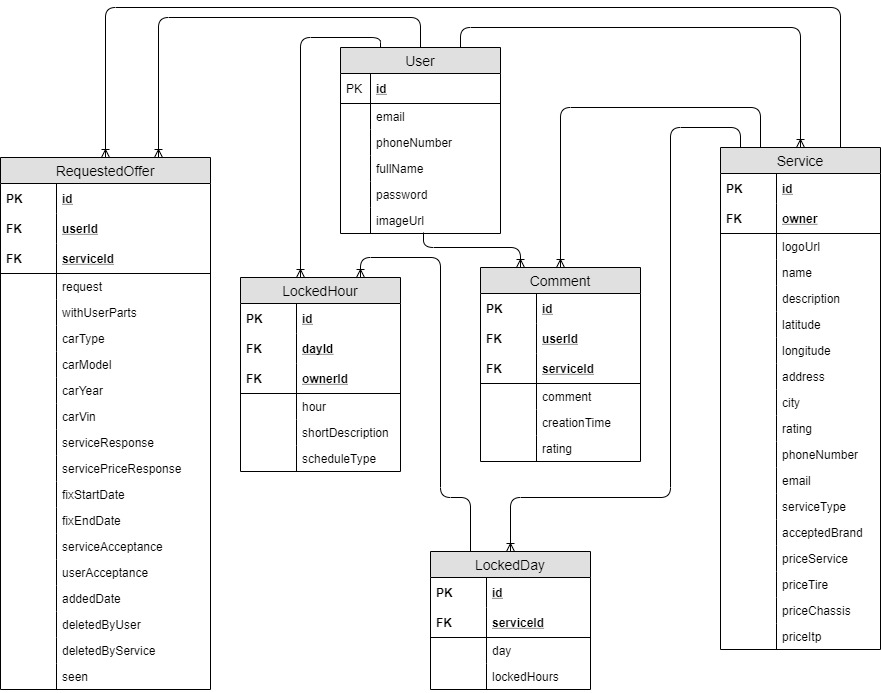


Fig. 8 - Diagrama bazei de date

# Funcționalitățile aplicației 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 întreba 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[[6]](#footnote-6), 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 utilizatorul 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.

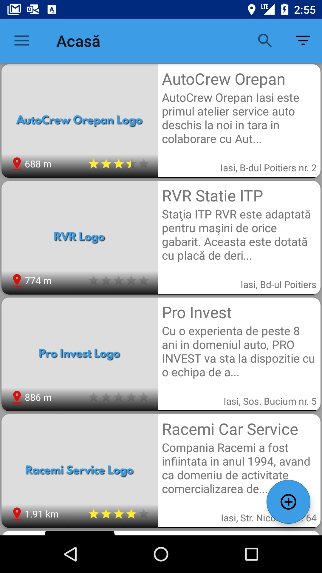
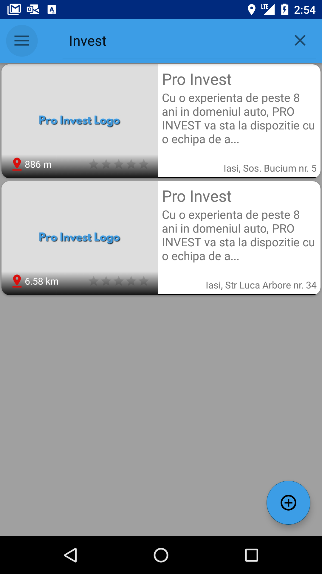
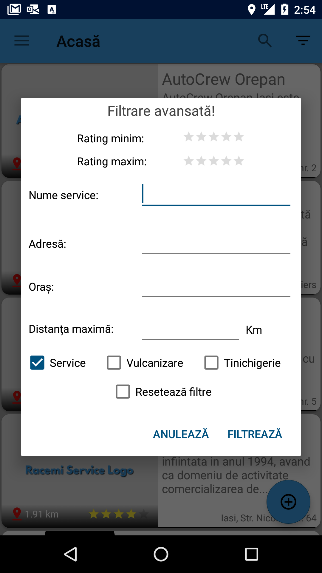
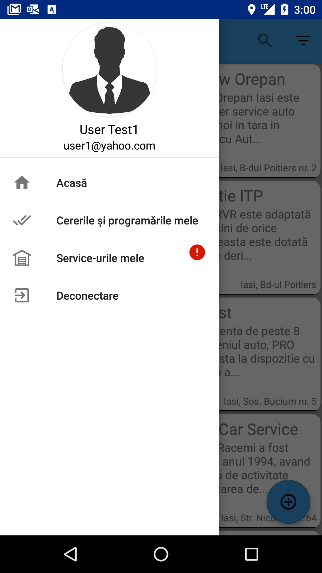


Fig. 9 - Imagini din aplicație cu lista de service-uri și meniul principal

## 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[[7]](#footnote-7).

|  |  |
| --- | --- |
| 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 opțiuni, 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. | Fig. 10 - 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 într-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 răspunsul 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.

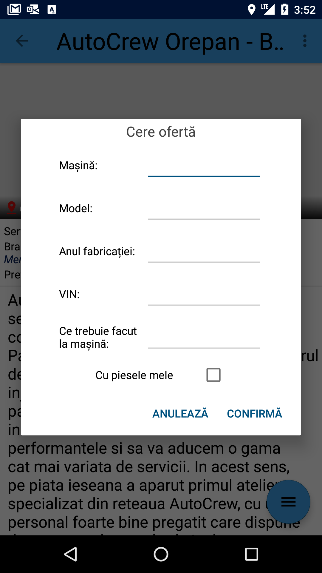
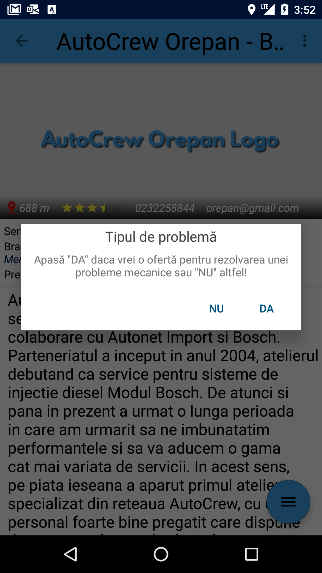
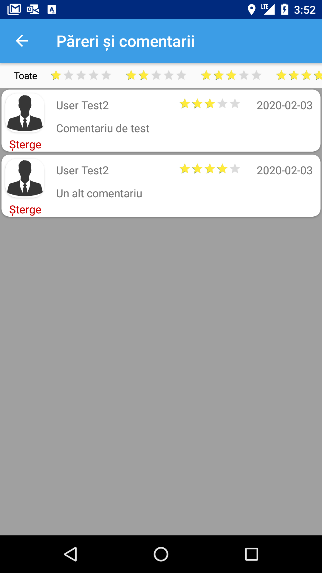
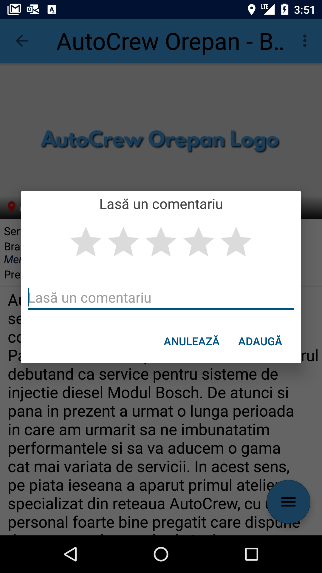


Fig. 11 - Adăugarea unui comentariu în primele două imagini și cererea unei oferte în ultimele două imagini

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 progrmarea 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 listă cu orele disponibile în acea zi. După alegere orei datele vor fi trimise spre server pentru adăugarea programării. Pentru realizare unei programări, am creat în baza de date două tabele. Tabela LockedDay ce reține pentru fiecare service toate zilele care au cel puțin o programare, iar pentru fiecare zi se va reține și câte programări există în acea zi, maximul fiind 8. A doua tabelă este LockedHour ce reține toate orele programate, pentru fiecare service. Astfel când vor ajunge pe server date pentru o nouă programare, serverul va verifica dacă pentru ziua dorită de utilizator există deja măcar o programare, iar dacă există va incrementa cu 1 câmpul lockedHours din tabela LockedDay. Dacă nu există va crea o nouă intrare setând campul lockedHours pe 1, însemnând că este prima programare pe acea zi pentru acel service. După modificarea tabelei LockedDay se va trece la tabela LockedHour unde se va adăuga efectiv programarea, de asemenea aici se vor adăuga și referințele către service, utilizator și către ziua aflată în tabela LockedDay. Un utilizator are voie să se programeze maxim o dată pe zi la același service.

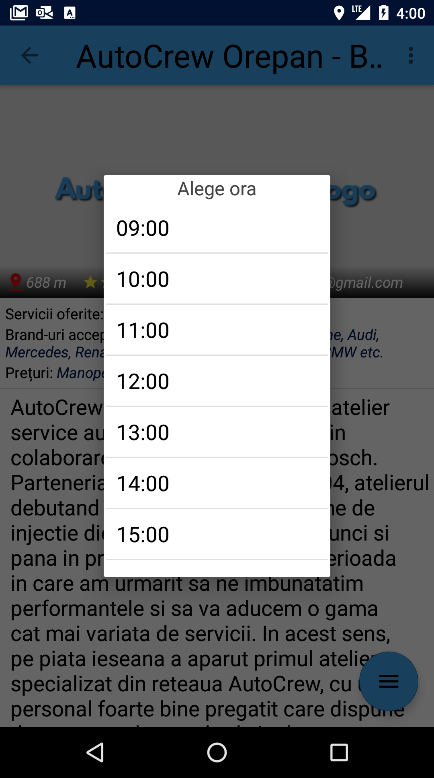
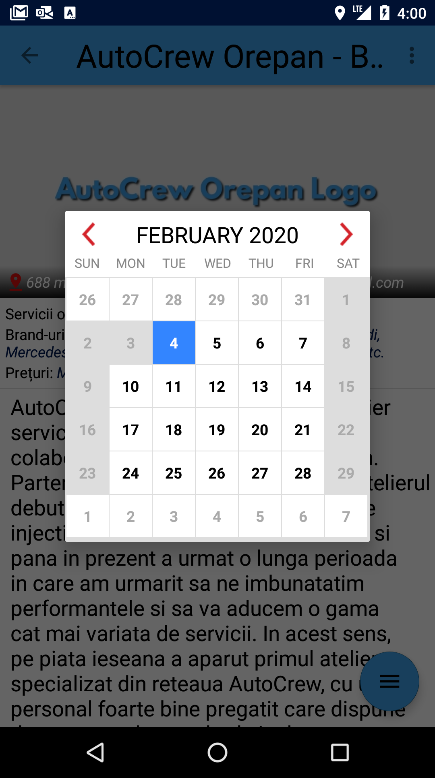
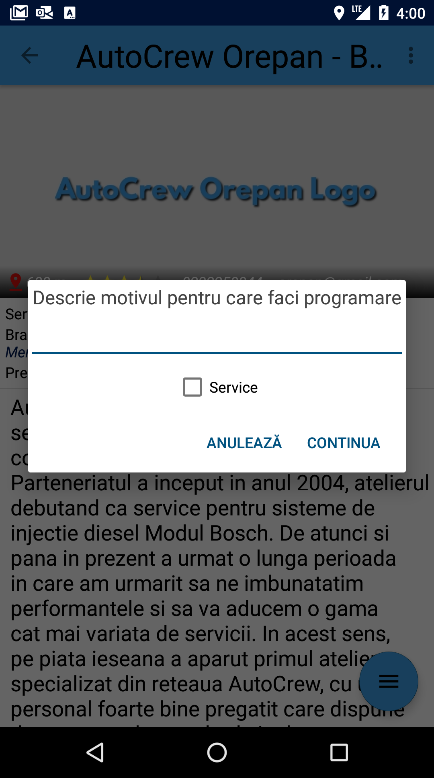


Fig. 12 - Realizarea unei programări

## Vizualizarea cererilor și a programărilor pentru un service

Am creat aceste funcționalități pentru proprietarii de service-uri, de a vizualiza cererile de ofertă și programările pe ziua curentă, pentru a putea răspunde la aceste cereri și pentru a permite service-urilor să-și organizeze activitățile în funcție de aceste programări. Astfel accesând meniul service-ului și apăsând butonul „Cereri de ofertă”, aplicația va deschide activitatea OfferRequestsActivity, cu ajutorul unui Intent, și îi va seta ca mesaj ce trebuie să ajungă la activitate, id-ul service-ului pentru care trebuie să afișeze cererile de ofertă. În această nouă activitate, cererile de ofertă vor fi afișate sub forma unei liste. Bineînțeles că și pentru această listă am folosit tot un RecyclerView pentru afișare, un ArrayList pentru memorarea elementelor și adaptorul RequestOfferAdapter pentru legarea elementelor din listă de *view-urile* din RecyclerView. După afișarea listei de cereri, proprietarul poate alege să accepte acea cerere și să trimită un răspuns celui care a făcut cererea, sau să nu accepte. Abia după ce a acceptat cererea sau a refuzat-o va putea să o șteargă din listă. Dacă service-ul dorește să facă o ofertă, aplicația prin intermediul unui *pop-up* îl va pune pe utilizator să ofere o dată de intrare a mașinii în service, o dată de ieșire și prețul lucrării. Aceste date vor fi trimise spre server, care le va introduce în baza de date, imediat după asta, utilizatorul ce a făcut acea cerere putând să vadă raspunsul dat de service. În cazul în care service-ul nu dorește sa dea o ofertă, procesul este similar, singura diferență fiind conținutul *pop-up-ului* unde trebuie completat cu motivul pentru care refuză să facă o ofertă.

Ca și la vizualizarea cererilor, pentru a putea vedea programările service-ului pe ziua curentă, proprietarul, va trebui să acceseze butonul din meniu ce deschide această pagină. Astfel la apăsarea acestei pagini se va crea activitatea TodayAppointmentsActivity ce oferă o listă cu toate programările pe ziua de astăzi. Desigur implementarea fiind similară, se cere de la server toate programările pe ziua curentă, apoi se introduc într-un ArrayList, iar cu ajutorul adaptorului AppointmentsAdapter se face legătura cu *view-urile* oferite de un RecyclerView. De pe această pagină nu se pot realiza alte acțiuni, scopul ei fiind doar de a afișa utilizatorului programările pe ziua curentă.

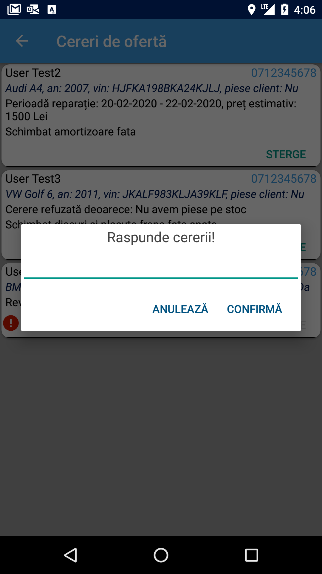
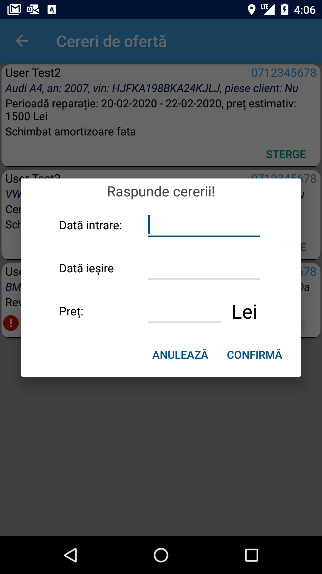
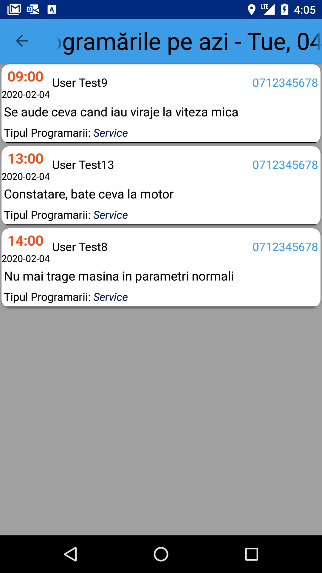


Fig. 13 - Figură 9 – Prima imagine reprezintă pagina cu programările pe ziua curentă, a doua este pagina cu cererile de ofertă primite de un service iar ultimele două reprezintă pop-up-urile afișate pentru răspunderea la cerere în mod pozitiv, respectiv negativ

## Vizualizare cereri de ofertă și programările unui utilizator

Similar punctului anterior, și clienții pot vedea la ce service-uri s-au programat și la ce service-uri au făcut o cerere de ofertă. Pagina ce afișează aceste detalii se poate accesa din meniul oferit de activitatea principală, HomeActivity. După apăsarea butonului din meniu, fragmentul pe care se afla utilizatorul în momentul apăsării (cel mai probabil HomeFragment) va fi înlocuit de un alt fragment, ce oferă la rândul său un container pentru fragmente, navigarea între ele făcându-se prin intermediul unor taburi situate în partea de sus a paginii sub bara de instrumente. Primul tab fiind destinat cererilor de ofertă, de unde utilizatorul va putea vedea o listă cu toate cererile făcute de el. Tot aici, pentru cererile sale care au primit un răspuns de la service, va putea să accepte oferta sau să o refuze. Pe această pagina utilizatorul va putea șterge orice element din listă făra a fi nevoit să facă vreo acțiune înainte, cum am văzut în cazul proprietarului de service. La ștergerea unei cereri la care service-ul nu a oferit un răspuns, acea cerere va fi ștearsă și pentru client și pentru service. Dar dacă clientul vrea să șteargă o cerere la care a primit un răspuns, atunci acea cerere va fi implicit refuzată din partea clientului, iar în baza de date doar va fi invalidată pentru a nu mai fi afișată clientului, dar service-ul încă va vedea acea cerere, cu statusul de refuzată, până va alege el să o șteargă. Abia apoi va fi ștearsă de tot din baza de date.

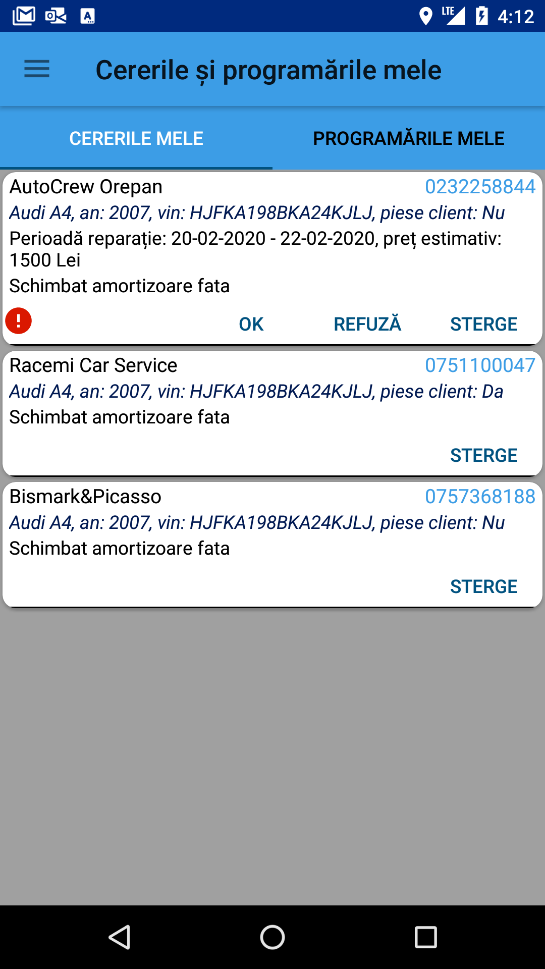


Fig. 14 - Prima imagine reprezintă lista de cereri făcută de un utilizator la un service iar a doua reprezintă programările făcute

## Adăugarea, editarea și ștergerea unui service

Cum am menționat la prima funcționalitate, pentru a adăuga un service am folosit un FloatingActionButton, care în momentul apăsării pe el va deschide un *pop-up*. Aici utilizatorul va trebui să completeze toate câmpurile pentru a putea adăuga noul service. În partea de sus a *pop-up-ului* se află o hartă oferită de Google. Pentru a putea utiliza această hartă am fost nevoit să intru pe Google Cloud Platform Console[[8]](#footnote-8) pentru a genera o cheie, iar apoi am introdus acea cheie ca meta dată în fișierul manifest din android sub eticheta *<application>*.

|  |  |
| --- | --- |
| Fig. 15 - Exemplu cum se adaugă cheia generată de Google | Apoi am descărcat două fișiere .aar oferite de Google[[9]](#footnote-9) și le-am copiat în proiectul meu la calea app/libs. |

Ultimul lucru a fost să creez un element de tipul MapView pentru a afișa harta. Această hartă este responsabilă de furnizarea poziției noului service. Prin urmare utilizatorul va fi nevoit să pună un *marker* pe hartă, altfel se va folosi *markerul* pus la crearea *pop-up-ului* (va fi setat pe hartp la locația în care se află utilizatorul). Sub elementul de tip MapView, am pus un ScrollView unde se găsesc datele ce trebuiesc completate despre service, acestea fiind: numele service-ului, adresa, orașul, numărul de telefon, email-ul, descrierea, mărcile de mașini acceptate și tipul de servicii oferite împreună cu un preț (de exemplu Service sau Vulcanizare). Ultimul element din acest ScrollView este o imagine care în momentul apăsării pe ea va deschide galeria pentru a permite utilizatorului să adauge o siglă noului service. Dacă utilizatorul nu alege nicio imagine, service-ul va avea o imagine prestabilită de aplicație. După introducerea tuturor datelor și verificarea lor de către aplicație atunci când utilizatorul apasă butonul de confirmare, aplicația va începe procesul de stocare a acestor date. Prima dată se va încărca imaginea cu sigla pe platforma Firebase, iar aceasta va oferi înapoi un *link* de unde se poate accesa în orice moment acea imagine, apoi acel *link* va fi trimis către server-ul scris în python, împreună cu celelalte date preluate din *pop-up*. Serverul va avea rolul de a lua toate aceste informații și de a le introduce în baza de date. Astfel cand se va cere aplicației afișarea unui service, se vor lua aceste date de la server și se vor afișa, mai puțin imaginea care se va cere de la Firebase prin biblioteca Glide și cu ajutorul *link-ului* primit de la server.

Editarea unui service se va realiza similar cu adăugarea. O diferență ar fi locul de unde se accesează butonul de editare, acesta fiind găsit în meniul de pe pagina cu detalii despre service, doar dacă utilizatorul ce accesează pagina este și proprietarul service-ului. *Pop-up-ul* afișat la accesarea butonului de editare este aproape identic, diferența fiind făcută de câmpurile cu detaliile service-ului, acestea fiind deja completate cu actualele informații despre service, utilizatorului rămânându-i doar să modifice acele informații și apăsarea butonului de confirmare. Apoi restul procesului de editare fiind făcut de aplicațe, adică modificare siglei pe platforma Firebase și a datelor din baza de date.

Pentru a șterge un service pașii ce trebuie să-i facă utilizatorul sunt următorii: navigarea pe pagina cu toate service-urile lui (acest pas este realizat prin apăsarea butonului corespondent din meniul principal al aplicației), apăsarea butonului de ștergere aflat pe view-ul care afișează service-ul ce îl vrea șters și confirmarea că este sigur că vrea să șteargă acel service.

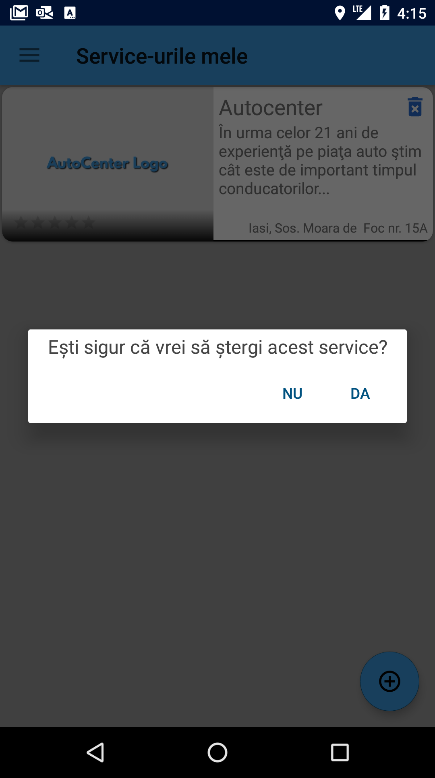
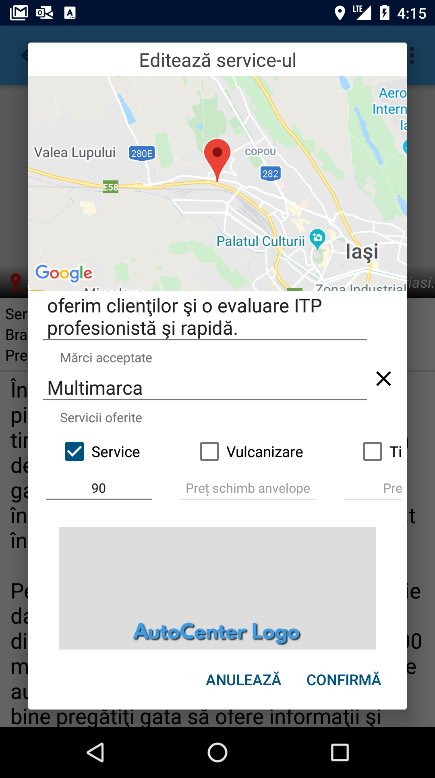
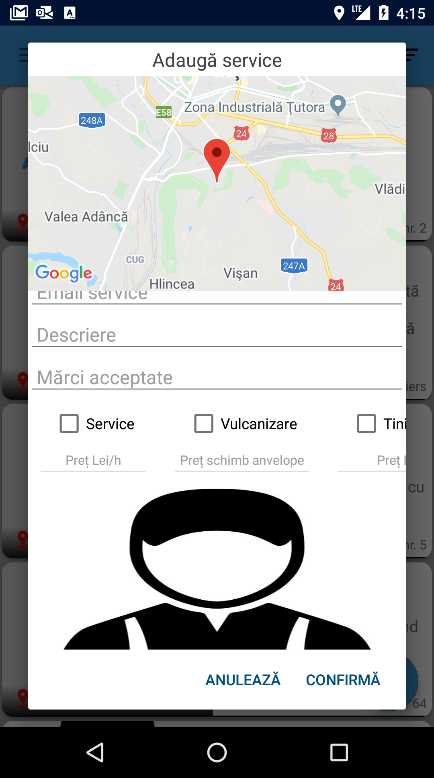


Fig. 16 - Adăugare, editare și ștergere service

## Autentificarea și schimbarea datelor de profil

Pentru acestă funcționalitate, am creat activitatea AuthenticationActivity, o interfață *Authentication.java* și două clase, *Login.java* și *Register.java* ce implementează această interfață. Activitatea se ocupă de afișarea a două butoane, 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 afișa *pop-upul* care 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 utilizatorului, se va deschide galeria. La apăsarea butonului de confirmare oferit de *pop-up*, fie pentru conectare sau înregistrare, mai întâi se va verifica dacă câmpurile obligatorii nu sunt goale, în caz contrar se va seta un mesaj de eroare, abia după această verificare se vor 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 imaginii de profil, numele utilizatorului și alte date întoarse de server, folosite de aplicație când va fi deschisă din nou. Răspunsul pozitiv al serverului poate apărea, în cazul conectării, atunci când utilizatorul a dat o adresă de email validă și parola corespunde acelei adrese de email, iar în cazul autentificării, atunci când utilizatorul a dat o adresă de email ce nu a mai fost înregistrată înainte în aplicație.

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 pune utilizatorului să-și modifice datele introduse la înregistrare. După apăsarea butonului de confirmare, procesul de verificare a datelor este similar ca la înregistrare.

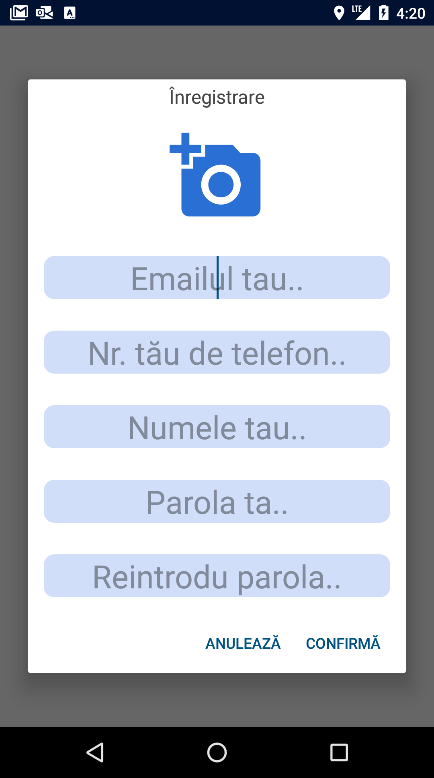


Fig. 17 - Înregistrarea și Conectarea la aplicație

După cum am văzut mai sus, aplicația oferă diferite servicii, atât pentru utilizatorii ce doresc să-și promoveze service-ul, dar și pentru utilizatorii ce caută un service pentru a-și repara mașina. Astfel aplicația poate fi privită din două părți: vizitator, utilizator conectat.

Din punct de vedere al utilizatorului neconectat, adică vizitator, aplicația oferă următoarele funcționalități din cele prezentate mai sus: afișarea unei liste de service-uri, vizualizarea detaliilor despre service-uri și apelarea, trimiterea unui email și calcularea unei rute către un service. Spre deosebire de un simplu vizitator, utilizatorii ce au un cont și își caută un service, dar si utilizatorii ce au un service adăugat în aplicație au posibilitatea de a explora toate funcționalitățile aplicației.

# 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 acestora cu succes.

## Adăugarea unui service

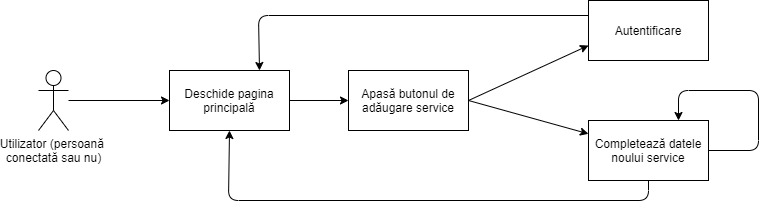


Fig. 18 - Diagramă use case de adăugare a unui nou service

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 afișându-i 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

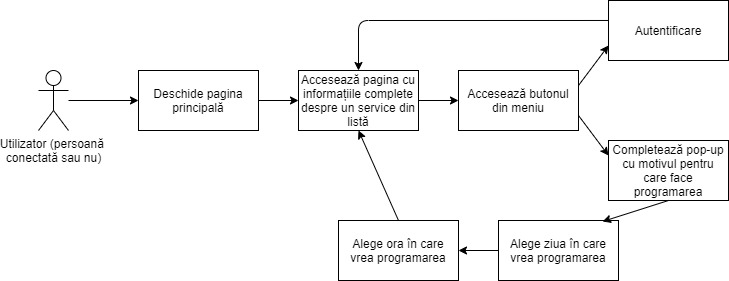


Fig. 19 - Diagramă use case de programare la un service

În cel de-al doilea scenariu prezint modul 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 utilizatorul 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 reprezintă 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

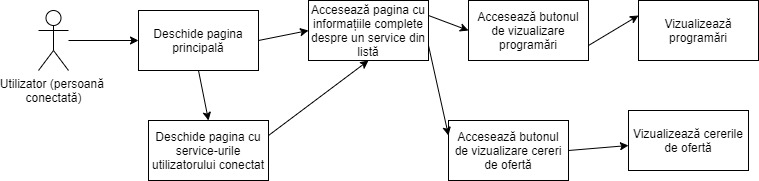


Fig. 20 - Vizualizarea cererilor și a programărilor unui service

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 acceseze 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  Fig. 21 - 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 tânăr ajuns la vârsta de 18 ani dorește să aibă o mașina, sunt familii ce dețin câte două sau chiar trei 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 că 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, de asemenea 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).

Pe lângă faptul că această aplicație are potențialul să ajute foarte mulți utilizatori, consider că dezvoltarea ei m-a ajutat și pe mine să înțeleg ce reprezintă construirea unei aplicații *end-to-end*, de asemenea cred că m-a făcut să înțeleg mult mai bine interacțiunea dintre un client și un server web. Realizarea acestei aplicații mi-a demonstrat că este foarte important crearea unei componente *front-end* bine definită și structurată în dezvoltarea ulterioară a componentei back-end și a întregii aplicații.

## 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 utilizatorilor 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 cereri 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://firebase.google.com/docs/android/setup>

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

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

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

- <https://c4model.com/>

- <https://www.svgrepo.com/svg/98979/car-shed>

- <https://www.svgrepo.com/svg/1930/mechanic-user>

- <https://www.youtube.com/watch?v=J5bIPtEbS0Q>

- <https://developer.android.com/guide/topics/ui/menus>

- <https://www.tutlane.com/tutorial/android/android-intent-filters-with-examples>

- <https://developer.android.com/training/data-storage/shared-preferences>

- <https://www.jetbrains.com/pycharm/>

- <https://developer.android.com/reference/android/support/v4/app/FragmentPagerAdapter>

- <https://developer.android.com/reference/android/app/Activity>

- <https://developer.android.com/guide/components/fragments>

- <https://developer.android.com/training/basics/fragments/fragment-ui>

- <https://developer.android.com/guide/topics/ui/layout/recyclerview>

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

- <https://picfont.com/faq.php?id=0>

# Anexe

Pentru realizarea acestei aplicații au fost folosite mai multe tehnologii, atât învățate în perioada facultății cum ar fi Python, Android, dar și tehnologii ce nu au fost predate, în 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 interfața dintre user și funcționalitățile aplicației. Deasemenea aplicația folosește diferite *framework-uri*/biblioteci adiționale. Dezvoltarea aplicației am realizat-o cu ajutorul IDE-ului[[10]](#footnote-10) Android Studio. Acesta este dezvoltat de Google și este IDE-ul oficial pentru Android. Cu ajutorul său se pot crea aplicații pentru toate dispozitivele ce folosesc ca sistem de operare Android. Printre cele mai importante caracteristici ale acestuia se afle posibilitatea de a integra cu GitHub, aplicarea modificărilor de cod instant, este un editor de cod inteligent.

|  |  |
| --- | --- |
| **Firebase** este o platformă oferită de Google pentru dezvoltarea aplicațiilor mobile, dar și web ce oferă o arhitectură care decuplează serverul de client, comunicarea dintre cei doi făcându-se prin intermediul unei bazei de date în timp real. | a51.png (1024×506)  Fig. 22 - Arhitectura unui sistem decuplat cu **Firebase** (source: <https://www.todaysoftmag.ro/article/2642/sisteme-decuplate-folosind-platforma-firebase-de-la-google>) |

**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[[11]](#footnote-11)** este de asemenea o bibliotecă pentru android, de tipul Fluent API[[12]](#footnote-12) ce permite trimiterea de cereri prin intermediul protocolului HTTP către un server proiectat pe arhitectura REST. Biblioteca permite atât cereri asincrone, dar și sincrone. Cu ajutorul acestei biblioteci este realizată comunicarea dintre serverul făcut in Python și interfața Android.

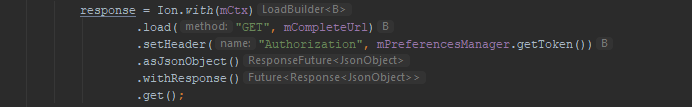


Fig. 23 - Exemplu de cod ce folosește biblioteca **Ion**

**Glide[[13]](#footnote-13)** este o bibliotecă implementată pentru android cu ajutorul căreia se pot încărca imagini în *view-urile* oferite de android (de exemplu ImageView). Deasemenea Glide oferă diferite beneficii cum ar fi posibilitatea de a folosi un cache ce crește viteza de încărcare a unei imagini a doua oară. Fiind o librărie de tipul Fluent API, cererile se pot face într-o singură linie. Această bibliotecă este folosită în ServiceFinder pentru a aduce imaginile din Firebase Storage și pentru a le încărca în *view-ul* destinat imaginii.

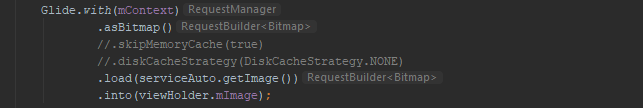


Fig. 24 - Exemplu de cod ce folosește biblioteca **Glide**

**Caldroid[[14]](#footnote-14)** este o bibliotecă similară cu clasa DatePickerDialog[[15]](#footnote-15) oferită de android, dar spre deosebire de acea clasă, biblioteca Caldroid oferă posibilitatea de a bloca diferite zile sau intervale de zile. Astfel, prin biblioteca de mai sus, aplicația ServiceFinder poate oferi utilizatorului un calendar ce afișează doar zilele disponibile în care se pot programa.

**Google Maps Platform[[16]](#footnote-16),** biblioteca oferită de Google propune o hartă ce are diferite funcționalități cum ar fi adăugarea de *markere*, preluarea coordonatelor a unui anumit *marker* sau chiar adăugarea de poligoane pe hartă. Am integrată aceast API în aplicație cu scopul de a pune la dispoziție posibilitatea de a adăuga unui service și locația sa, astfel fiind mult mai ușor clienților să găsească un service.

## Python

Python este unul dintre cele mai flexibile și populare limbaje de programare, cu o sintaxă simplă și o semantică dinamică, ce pune accentul pe curățenia și simplitatea codului. Pe lângă posibilitatea de a scrie cod orientat pe obiecte, acest limbaj oferă suport pentru programarea procedurală și programarea funcțională. Datorită lucrurilor de mai sus am ales ca server-ul aplicației sa fie implementat în acest limbaj.

Pentru dezvoltarea serverului în acest limbaj a trebuit să aleg un IDE în care să fac acest lucru. Astfel am ales PyCharm, un IDE foarte popular, ce oferă suport pentru diferite *framework-uri* web, printre care și Flask, deasemenea este un editor de cod inteligent.

|  |  |  |
| --- | --- | --- |
| Pe lângă limbajul pur, pentru a realiza serverul am folosit **Flask[[17]](#footnote-17)**, *framework* prin care se pot crea servere web in Python și **SQLAlchemy[[18]](#footnote-18)**, un *framework* de tipul ORM ce permite limbajului Python interactionarea cu baze de date. | Fig. 25 - Exemplu de cod ce folosește **Flask** și **SQLAlchemy** | |
|  | |  |

Biblioteca **PyJWT[[19]](#footnote-19)** permite generarea de *token-uri* pe baza unor date și decodarea lor. Scopul acestei biblioteci este de a genera un *token* de fiecare dată când un utilizator se conectează la aplicație, sau când un utilizator se înregistrează în aplicație. Apoi când un utilizator dorește modificarea, ștergere sau executarea oricărei operații ce ar duce la modificarea unor informații din baza de date, serverul cere un astfel de token pentru a restricționa acțiunile neautorizate asupra informațiilor din baza de date.

1. Descriere preluată de pe site-ul: <https://www.cylex.ro/> [↑](#footnote-ref-1)
2. <http://service.auto.ro/> [↑](#footnote-ref-2)
3. Android Nougat – versiunea de android lansată oficial pe data de 22 august 2016 [↑](#footnote-ref-3)
4. Dacă acesta a permis utilizarea locației la intrarea în aplicație [↑](#footnote-ref-4)
5. Harta oferită de Google, vezi anexe [↑](#footnote-ref-5)
6. RecyclerView este un layout de tip listă ce afișează într-un mod eficient elementele listei, cu acest layout se creează view doar pentru elementele afișate plus un număr de elemente aflate înainte și după elementele afișate, spre deosebire de ListView ce crează *view-uri* pentru întreaga listă [↑](#footnote-ref-6)
7. Intent-ul este un obiect cu ajutorul căruia se poate cere lansa o nouă activitate componentă a aplicației sau o nouă aplicație, de asemenea poate conține și un mesaj ce îl va transmite activității ce o va lansa [↑](#footnote-ref-7)
8. <https://console.cloud.google.com/google/maps-apis/overview?project=nimble-cortex-231817&folder=&organizationId=> [↑](#footnote-ref-8)
9. <https://developers.google.com/maps/documentation/android-sdk/v3-client-migration> [↑](#footnote-ref-9)
10. Un IDE este un program ce oferă un o mulțime de instrumente (editarea codului sursă, compilarea codului etc.) cu ajutorul căruia un programator poate dezvolta o aplicație [↑](#footnote-ref-10)
11. <https://github.com/koush/ion> [↑](#footnote-ref-11)
12. Este o metodă de proiectare a API-urilor orientate pe obiecte, ce presupune înlănțuirea a mai multor metode membre. Are ca scop creșterea lizibilități codului. [↑](#footnote-ref-12)
13. <https://bumptech.github.io/glide/> [↑](#footnote-ref-13)
14. <https://github.com/roomorama/Caldroid> [↑](#footnote-ref-14)
15. Android oferă clasa DatePickerDialog pentru a selecta o dată dintr-un calendar [↑](#footnote-ref-15)
16. <https://developers.google.com/maps/documentation/android-sdk/intro> [↑](#footnote-ref-16)
17. <http://flask.palletsprojects.com/en/1.1.x/> [↑](#footnote-ref-17)
18. <https://www.sqlalchemy.org/> [↑](#footnote-ref-18)
19. <https://pyjwt.readthedocs.io/en/latest/> [↑](#footnote-ref-19)