Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică

OnlineDroid: sincronizare a dispozitivelor Android cu aplicație web

Absolvent,
Ciobanu Tudor-Adrian

Coordonator,
Lect.Dr. Cosmin-Nicolae Vârlan

Iulie 2015

Universitatea "Alexandru Ioan Cuza" Iași
Facultatea de Informatică

LUCRARE DE LICENȚĂ

OnlineDdroid: sincronizare a dispozitivelor Android cu aplicație web

Tudor-Adrian Ciobanu

Sesiunea: *Iulie, 2015*

Coordonator științific,

Lect.Dr. Cosmin-Nicolae Vârlan

Cuprins

1. Introducere

1.1 Scurt istoric

Internetul a reprezentat un pas extrem de important în evoluţia omenirii. Odată cu răspândirea acestuia şi cu apariţia distribuţiei către publicul larg, acesta a devenit un instrument aproape indispensabil vieţii de zi cu zi. Principala utilitate a acestuia a fost iniţial răspandirea informaţiei. Internetul conţinea la început, o multitudine de site-uri web statice care ofereau informaţii din orice domeniu, şi la un nivel foarte mare de detaliu. Ulterior, prin dezvoltarea acestuia şi a protocoalelor prin care funcţioneaza, Internetul a început să ofere mai mult decât atât. Odată cu apariţia Forumurilor, utilizatorii au început să creeze comunităţi concentrate în jurul unui domeniu sau a unui hobby care îi leagă pe toţi participanţii acestuia. Libertatea de expresie pe Internet nu este îngrădită, acest lucru acţionând în favoarea răspândirii informaţiilor.

Odată cu dezvoltarea protocoalelor de comunicare şi a limbajelor de programare, au început să apară pagini web dinamice, al căror conţinut se schimba în funcţie de nişte evenimente. Acest lucru a făcut ca paginile statice care conţineau doar text, să aibă acum animaţii, porţiuni ale paginii care sunt actualizate în timp real, etc.
În acest punct, realizarea jocurilor single-player într-un browser web a devenit posibilă, şi au apărut o multitudine de instrumente special dezvoltate în acest sens. În ziua de astăzi există jocuri care pot fi jucate de utilizatori din browser, jocul rulând fie pe servere aflate la mii de kilometri distanţă, sau local, beneficiind de toate avantajele plăcii video.

Una dintre cele mai mari comunităţi, în jurul cărora revolva majoritatea utilizatorilor Internetului, o reprezintă platformele de social media. De la începutul anilor 2000, odată cu Myspace, Facebook şi Twitter, acest tip de platforme au fost în continuă creştere a numărului de utilizatori şi a cantităţii de informaţie care circulă pe acestea. În present, pe Facebook există o medie de 1.25 miliarde de utilizatori unici [13], care activează în fiecare lună pe platforma socială, iar pe Twitter, aproape 300 de milioane de utilizatori [14]. Aplicaţiile web ale acestor platforme sunt pline de conţinut dynamic, care se modifică în funcţie de preferinţele utilizatorilor, de comportamentul acestora, de paginile pe care le vizitează mai frecvent utilizatorul, etc. Conţinutul multimedia este integrat în pagini, şi pentru a fi vizualizat,

utilizatorul nu trebuie să mai ajungă pe pagina web unde acesta a fost hostat. Această dezvoltare continuă a tehnologiilor şi uşurinţa utilizării acestuia, a făcut ca Internetul să devină extrem de popular, chiar şi printre utilizatorii care nu au cunoştinţe remarcabile în operarea unui calculator.

Urmatorul pas important în evoluţia Internetului s-a produs odata cu apariţia şi popularizarea smartphone-urilor. Cu ajutorul acestor dispozitive a devenit posibil accesul la internet din orice locaţie. Utilizatorul nu mai trebuie să stea în faţa unui calculator sau să deţină un laptop, pentru a putea avea acces la multitudinea de aplicaţii disponibile pe Internet.

Dispozitivele mobile au cunoscut o evoluţie accelerată odată cu intrarea companiei Apple pe piaţă, cu telefoanele iPhone ce folosesc sistemul de operare iOS, şi ulterior a companiei Google, prin telefoanele care rulează sistemul de operare Android. Aceste dispozitive au propriul ecosistem de aplicaţii dedicate platformei, care adaugă o multitudine de funcţionalităţi în plus telefonului, aşa încât în ziua de astăzi, majoritatea telefoanelor au mai mult rolul de dispozitiv multimedia mobil decât rol de telefon propriu-zis. Apariţia ulterioară a tabletelor nu a făcut decât să întărească această depărtare de la rolul de telefon.

Accesul la internet de pe un dispozitiv mobil nu a reprezentat o noutate, acesta existând deja în pachetele oferite de companiile de telefonie mobilă, însă odată cu dezvoltarea smartphone-urilor, apare accesul la Internet prin Wi-Fi. Acesta se dovedeşte a fi un adevărat competitor pentru conexiunea folosind date, deoarece în majoritatea timpului era gratuit, în ziua de astăzi existând Internet prin Wi-Fi gratuit, peste tot în marile orașe ale lumii.

Aplicaţiile dezvoltate pentru dispozitivele mobile folosesc în principal Internetul, şi oferă o nouă dimensiune acestuia. Platformele de social media au dezvoltat aplicaţii dedicate acestora, care datorită popularităţii, vin deja integrate în sistemele de operare, şi oferă aceeaşi experienţă ca şi aplicaţiile web din browser.

Această evoluţie continuă a dus la dispariţia graniţelor dintre aplicaţiile web şi platforma pe care acestea rulează, lărgind spectrul de utilizare şi dezvoltare al acestui gen de aplicaţii.

1.2 Scopul Aplicaţiei

Aplicaţia OnlineDroid îşi propune să aibă rolul unui punct central pentru notificările dispozitivelor sincronizate. Acesta oferă o interfaţă grafică intuitivă şi atractivă în care utilizatorul este notificat asupra evenimentelor desfaşurate în dispozitivele Android pe care le sincronizează cu serverul.

Pentru a putea utiliza aplicaţia, utilizatorul trebuie să înregistreze un cont, folosind aplicaţia web disponibilă la adresa: http://limitless-savannah-8511.herokuapp.com/.

După autentificarea în aplicaţie, utilizatorului îi este prezentată interfaţa, prin intermediul căreia poate interacţiona cu dispozitivele.

Următorul pas este instalarea aplicaţiei Android pe dispozitivele ce rulează acest sistem de operare, autentificarea în aplicaţie, şi bifarea opţiunilor dorite pentru fiecare dintre acestea. Versiunea minimă necesară pentru ca sistemul de operare să poată rula aplicaţia este: Android "Kit-Kat" 4.4.

Aplicaţia se împarte în două componente principale, care se disting prin platforma pe care rulează:

- **serverul web**, implementat în node.js, prin framework-ul Express, oferă accesul la aplicaţie din orice browser, avantajul principal fiind portabiltiatea şi posibilitatea de acces indiferent de dispozitivul utilizat.

- **aplicatia android**. Aceasta se instalează pe orice tip de dispozitiv (smartphone sau tabletă), care rulează versiunea cerută a sistemului de operare.

Odată autentificat, dispozitivul pe care s-a instalat aplicaţia, se sincronizează cu serverul web. Utilizatorul are acces la un meniu de setări, de unde poate alege ce date doreşte să transmită către server. Opţiunile posibile sunt:

- notificarea serverului în privinţa mesajelor SMS recepţionate de dispozitiv (dacă dispozitivul este smartphone);

- notificarea serverului în privinţa apelurilor recepţionate de dispozitiv (daca acesta este smartphone);

- posibilitatea de a trimite mesaje SMS din interfaţa aplicaţiei web (daca dispozitivul sincronizat este un smartphone);

- notificarea serverului în privinţa tuturor notificărilor primite de la aplicaţiile instalate pe

dispozitiv;

- obţinerea poziţiei geografice a dispozitivului în interfaţa aplicaţiei web.

Toate aceste funcţionalităţi pot fi activate şi dezactivate în mod independent una de cealaltă şi oferă utilizatorului informaţii cât mai diverse despre starea în care se află dispozitivul.

Scopul practic al aplicaţiei este acela de a oferi utilizatorului posibilitatea de a fi la curent cu evenimentele ce se desfaşoară în dispozitivele sale, în special în situaţiile în care utilizatorul nu are dispozitivul la el.

2.Tehnologii folosite

2.1 Android

Android este un sistem de operare pentru dispozitive mobile, bazat pe Linux kernel, fiind dezvoltat de compania Google. Acest sistem este disponibil pentru majoritatea telefoanelor de tip smartphone, tablete, smart-watches şi pentru computerele de bord prin Android Auto. Android este destinat dispozitivelor cu input de tip touch, şi suportă o multitudine de acţiuni prin gesturi, ce oferă o experienţă intuitivă şi plăcută. Sistemul de operare Android este open source, aflat sub licenţă Google, însă, de obicei, vine preinstalat pe dispositive, acompaniat de software proprietar al companiilor care au creat dispozitivul în cauză, dar şi software proprietar al companiei Google. În magazinul virtual al platformei Android există la momentul actual peste un milion de aplicaţii descărcate de peste 50 de miliarde de ori.[7]

2.1.2 Istoric

Sistemul de operare Android a fost dezvoltat iniţial de către compania Android.Inc şi a fost ulterior achiziţionată de către Google, în anul 2005. În anul 2007, Google a dezvăluit prima versiune dezvoltată în cadrul companiei, după care, timp de doi ani, au apărut diverse versiuni beta. În 2008, a fost lansată versiunea 1.0, care s-a remarcat prin prima versiune a magazinului virtual "Android Market Beta". Această versiune a reprezentat un pas important, întrucât în ziua de astăzi, Google Play Store este un punct central în dezvoltarea platformei şi al distribuţiei de aplicaţii la nivel mondial. De asemenea, această primă versiune a sistemului de operare, a fost însoţită de hardware dedicat, prin telefonul HTC Dream.

În Android 1.1 apare prima versiune de Google Voice Search, marcând începutul dezvoltării sistemului de operare şi în partea de cloud computing.

Odată cu lansarea din anul 2009, versiunile sistemului de operare încep să fie însoţite de denumiri de dulciuri, în ordine alfabetică. Android 1.5, intitulat "Cupcake", aduce prima tastatură on-screen, făcând trecerea de la tastaturile fizice, la dispozitivele cu touch-screen. Tot în aceasta versiune, datorită măririi dimensiunilor ecranului, apar şi primele widget-uri şi se pun

bazele interfeţei grafice care există şi în cele mai recente versiuni.

Android 1.6 “Donut” şi 2.0 “Eclair”, aduc o mulţime de îmbunătăţiri şi actualizări ale interfeţei grafice şi aplicaţiilor preinstalate. Ecranele mari încep să devină standard, se renunţă la butoanele fizice pentru primit şi respins apeluri, acestea fiind reprezentate pe ecran în cadrul aplicaţiei telefonice. În 2010, Google lansează primul smartphone creat în producţie proprie, intitulat “Nexus One”. Acesta a fost printre primele telefoane care au fost vândute în magazinele online, nefiind încheiate contracte cu distribuitorii de telefonie mobilă, Google dorind să îndrepte piaţa spre telefoane vândute independent.

În versiunea 2.1 se pune accent foarte mult pe efectele grafice şi animaţii, marea majoritate a componentelor grafice existente deja în sistemul de operare fiind îmbogăţite cu animaţii şi tranziţii. Un mic update la această versiune, a introdus funcţionalitatea multi-touch, patentată de către compania Apple, ducând la o serie de procese şi la o competiţie agresivă între cele doua companii.

Android 2.2 “Froyo”, aduce compilarea JIT (just-in-time), care converteşte bytecodul java în cod nativ la runtime, şi îmbunătăţeşte semnificativ performanţele sistemului de operare. Browserul Chrome, dezvoltat de Google, este îmbunătăţit semnificativ prin integrarea motorului javascript intitulat V8. Tot în această versiune, apar integrate şi primele aplicaţii de social media cum ar fi Twitter şi Facebook, dar şi o nouă funcţionalitate dezvoltată de Google, numită “Voice actions”, ce permite controlul dispozitivului Android prin comenzi vocale. Recunoaşterea vocală nu avea la acel moment o utilitate foarte mare, şi nici nu se află într-un stadiu extrem de dezvoltat, însă cu trecerea timpului a dus, prin perfecţionarea sistemului, la mijloacele principale de interacţiune cu Google Glass sau smart-watch-urile.

În Android 2.3 “Gingerbread”, se modifică masiv interfaţa grafică a sistemului de operare, se perfecţionează bara de notificări şi meniurile. Astfel, apar “Android Market 2.0”, şi alte câteva servicii dezvoltate de Google, cum ar fi Google Books.

Android 3.0 “Honeycomb”, aduce suport nativ pentru tablete, şi marchează intrarea Google pe piaţa tabletelor. Sistemul de operare Android, suportă astfel rezoluţii mult mai mari, spaţiul disponibil pe ecran este mai mare şi permite un layout diferit al aplicaţiilor, spre deosebire de modul portret, în care erau realizate până acum. Din acest motiv, apare “Fragments” API, prin care ecranul unei aplicaţii poate fi divizat în mai multe părţi independente şi poate fi manipulat separat de celelalte. Până la aceasta versiune, telefoanele care rulau

Android nu putea funcţiona fără un buton hardware pentru “back”, “menu” şi “home”, însă acum acestea au fost integrate în interfaţa grafică.

Android 4.0 “Ice Cream Sandwich”, este prima versiune destinată şi telefoanelor, după Gingerbread. Datorită actualizarilor din versiunea 3.0, au fost eliminate toate butoanele fizice, mai puţin cele pentru controlul volumului şi buton pentru blocare. Bara de notificări primeşte actualizări importante şi Android Market este remodelat. Pentru dezvoltatorii de aplicaţii, Google adaugă modul “Developer”, prin care utilizatorul vede o diagnoză în timp real a aplicaţiilor, consumul de memorie, viteza animaţiilor şi altele.

În anul 2012, Google relansează magazinul virtual de aplicaţii sub denumirea de “Google Play Store”, şi prin aceasta, utilizatorii pot cumpăra şi descărca atât cărti, muzică şi filme pentru aplicaţiile dedicate, oferite de Google.

Odata cu apariţia Android 4.1 “Jelly Bean”, Google lansează serviciul de cloud intitulat “Google Now”. Acest serviciu, oferă un motor de căutare predictiv şi o serie de informaţii cum ar fi: starea vremii, trafic, şi multe altele, determinate în mod inteligent, în funcţie de comportamentul utilizatorului.

Datorită noilor modificări făcute asupra Google Play Store şi Google Play Services, actualizările la sistemul de operare încep să fie livrate parţial prin această platformă, fără a mai cere o actualizare a întregului sistem de operare. Tot în această perioadă, Google lansează şi integrează în aplicaţiile sale, propria platformă de social media, numită “Google+” şi aduce modificări semnificative asupra aplicaţiei “Google Play Music”, aceasta devenind un serviciu cu subscripţie lunară. Este lansat şi “Google Play Games”, un serviciu de back-end ce oferă suport dezvoltatorilor de jocuri, pentru integrarea unui sistem de “achievements”, multiplayer, şi tabele de scoruri. Astfel, smartphone-urile intră pe piaţa dispozitivelor de gaming mobile şi concurează direct cu produse ca Sony PS Vita sau Nintendo 3DS.

Cateva luni mai târziu, apare şi primul re-design al aplicaţiei Google Maps, aceasta nemaifiind actualizată de la Android Ice Cream Sandwich.

Android 4.3 “Jelly Bean”, introduce pentru prima oară suport pentru dispozitive purtabile (wearables). Odata cu acest produs nu au apărut şi astfel de dispozitive, însă au fost făcute publice atât intenţia de a intra pe această piaţă cu sistemul de operare Android, cât şi API-urile pentru dezvoltatorii de aplicaţii (au fost puse la dispoziţie devreme, pentru a da un impuls).

Pentru versiunea Android 4.4 “Kit-Kat”, Google a încheiat un parteneriat cu compania Nestle, pentru a folosi acest nume, şi a fost lansat de Halloween, pe 31 Octombrie 2013. Focusul principal în această versiune a sistemului de operare a fost optimizarea utilizarii memoriei RAM, cât şi finisarea interfeţei cu utilizatorul, astfel Android putea să ruleze pe numai 340Mb RAM. Acest lucru a făcut posibilă instalarea celei mai noi versiuni a sistemului de operare pe dispozitive mai vechi, care aveau 512 Mb memorie RAM, mărind procentul de utilizatori care folosesc ultima versiune lansată. Un alt motiv pentru care în această versiune s-a pus accent pe minimizarea memoriei necesare pentru a rula, a fost continuarea integrării suportului pentru dispozitivele wearable, unul din primele astfel de dispozitive fiind Google Glass.

Cea mai recentă versiune a sistemului de operare este Android 5.1.1, intitulată “Lollipop”, lansată pe 12 Noiembrie 2014. Principala modificare pe care o aduce această versiune este re-redisgnul interfeţei grafice cu asa numitul “material design”. Acestă versiune are la origini designul utilizat de Google la “cardurile” din aplicaţia Google Now, şi pune accent pe notificări, animaţii responsive şi tranziţii, şi adâncimea elementelor grafice realizată prin utilizarea umbrelor. Aceasta, a dus şi la refacerea interfeţelor grafice folosite în aplicaţiile native ale sistemului de operare, şi în aplicaţiile dezvoltate de Google. Odată cu acest sistem de operare, apar şi primele dispozitive wearable care suporta Android 5.0, şi ne referim aici la smart-watches. Ele funcţionează prin sincronizarea cu un dispozitiv care rulează Android 4.4 sau o versiune mai nouă, iar această sincronizare se realizează la nivel de aplicaţie.

Astfel, pentru ca o aplicaţie să aibă suport pentru smart-watch, ea trebuie să conţină, separat, un pachet care să fie instalat din telefon, pe ceas. Aplicaţia din telefon va rula în fundal şi va comunica cu smart-watch-ul prin Bluetooth. [5]

La conferinţa Google I/O din Iunie 2015, Google a anunţat următoarea versiune de Android, sub numele de cod Android M. Printre funcţionalităţile importante care vor fi aduse de aceasta se numara: posibilitatea de a seta permisiunile aplicaţiilor în mod independent una de cealaltă, ținând cont că până acum acest lucru nu a fost posibil, utilizatorul putând doar să aleagă, daca doreste sau nu să instaleze aplicaţia. [6]

Pagini din aplicaţia Google Chrome pot fi acum integrate direct în aplicaţii, pentru a nu mai fi necesară deschiderea browserului ca o aplicaţie separată, suport pentru detectoare de amprente pentru blocarea dispozitivelor, un sistem de plăţi electronice numit “Google Pay”, precum şi îmbunătăţiri semnificative în durata de viaţă a bateriei, prin optimizări ale consumului şi

managementul acesteia prin sistemul de operare. Aceasta versiune este anunţată spre lansare în toamna anului 2015.

La ora actuală Google monitorizează distribuţia versiunilor sistemului de operare Android, de pe marea majoritate a dispozitivelor din întreaga lume. Android Lollipop este rulat pe aproximativ 12% din dispozitive, Android Kit-Kat pe 39%, iar Android Jelly Bean pe aproximativ 38% din totalul dispozitivelor monitorizate. Restul de 10-11% sunt dispozitive care rulează Android Ice Cream Sandwich sau versiuni mai vechi.[10]

2.1.3 Android API

2.1.3.1 Structura unei aplicaţii

Aplicaţiile dezvoltate pe platforma Android, sunt aplicaţii scrise în limbajul Java. Android SDK se ocupă de compilarea codului împreună cu fişierele resursă (fişiere XML), şi le împachetează într-un APK (Android package). Odată instalată o aplicaţie pe un dispozitiv Android, ea rulează în mod sandbox, pe câte o maşină virtuală, izolată de celelalte component, care rulează în paralel, din motive de securitate.

Android utiliza maşina virtuală Dalvik, dar odată cu versiunea 4.4 Kit-Kat, acesta a fost treptat înlocuită de ART (Android runtime). Acesta îmbunătăţeşte performanţa aplicaţiilor prin compilare AOT (ahead-of-time compilation).

Sistemul de operare Android este un sistem de operare Linux multi-user, în care fiecare aplicaţie joacă rolul unui utilizator. În mod implicit, sistemul atribuie fiecărei aplicaţii un ID unic, şi păstrează permisiunile aplicaţiei indexate după acesta.

Cu toate că aplicaţiile sunt izolate unele de celelalte în acest mod, nu sunt restricţionate metodele de distribuire a datelor între aplicaţii, sau de acces al aplicaţiilor la resursele sistemului. Permisiunile aplicaţiei constau în accesul aplicaţiei la componente hardware ale dispozitivului (Bluetooth, camera foro, SD card, etc), sau la componente software ale sistemului (accesul la SMS-uri, notificări ale sistemului, etc).

O aplicaţie Android este alcătuită în principal dintr-o serie de componente interconectate. Acestea sunt: Activity, Services, ContentProvider şi BroadcastReceiver. Aceste componente (mai puţin ContentProvider) se activează între ele prin intermediul unor mesaje asincrone,

numite Intents. Acestea leagă componentele individuale la runtime.

Orice aplicaţie Android are asociat un fişier XML în care sunt specificate caracteristicile principale ale acesteia cum ar fi: permisiunile de care are nevoie aplicaţia pentru a rula, componentele aplicaţiei, versiunea minimă a sistemului de operare de care are nevoie dispozitivul pentru a putea rula aplicaţia, etc. Prin acest fişier, aplicaţia îi face clar sistemului modul în care aceasta funcţionează şi care sunt componentele externe de care are nevoie.

Interfaţa grafică într-o aplicaţie Android se realizează prin fişiere XML, care sunt separate de partea de cod Java. Layout-ul ferestrelor aplicaţiei se creează prin intermediul acestor fişiere, însă ele pot fi modificate şi manipulate din cod, prin intermediul fişierului R, care conţine indexări ale tuturor elementelor grafice.

Utilizând fişierele XML este uşor să se creeze design specific pentru dispozitive care diferă ca dimensiune sau utilitate, cum ar fi tablete de 7" sau de 10", sau smartphone-uri de rezoluţie mică (2-3 inci). Android pune la dispoziţie un fişier de resursă numit strings.xml, în care utilizând tag-uri specifice, se pot indexa stringuri. Utilizând acest fişier, se poate realiza internaţionalizarea aplicaţiei cu uşurinţă, nefiind nevoie să se modifice text răspândit în cod.

Un alt aspect important în aplicaţiile Android îl reprezintă ciclul de viaţă al componentei Activity. Utilizatorul navighează între diferite aplicaţii, şi sistemul de operare realizează managementul stărilor acestora, pentru a se asigura că nu sunt pierdute date şi că aplicaţiile nu sunt oprite din cauze nedorite. După cum se observă în ilustraţia 1, aceste metode callback reprezintă stări prin care un Activity poate trece pe parcursul acestui ciclu de viaţă. [4] [2]

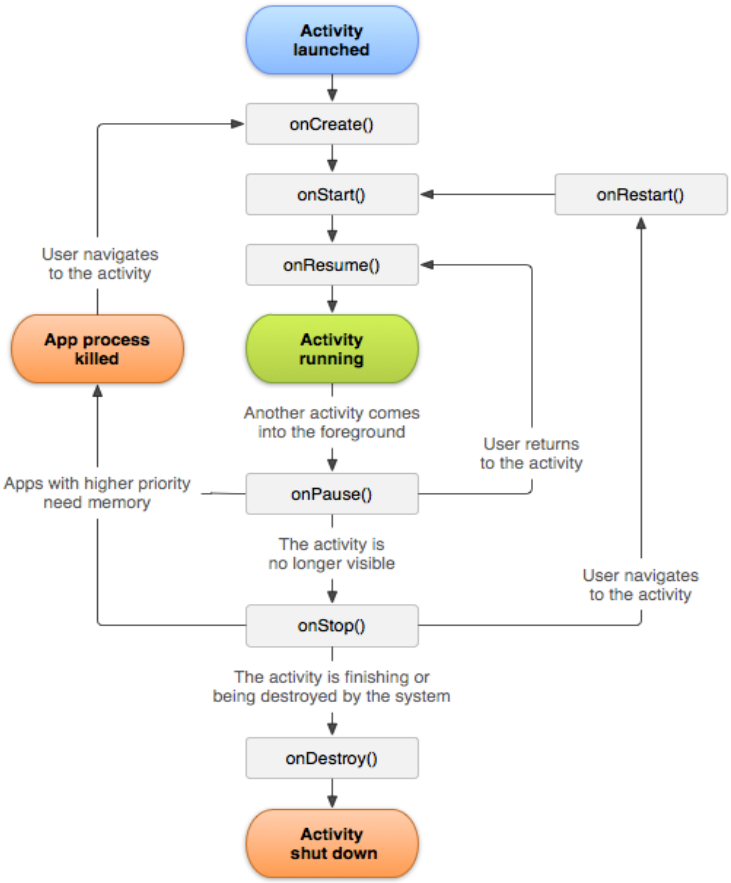


Illustration 1: Activity lifecycle

2.1.3.2 Activity

Principala componentă în majoritatea aplicaţiilor Android sunt ferestrele care compun structura vizuală a aplicaţiei. În dezvoltarea acestora, ele sunt reprezentate prin structura Activity. O aplicaţie poate conţine oricâte Activities, şi fiecare dintre acestea trebuie să implementeze obligatoriu două metode callback:

-”onCreate(Bundle)” - Această metodă este lansată în execuţie la momentul în care un Activity este pornit, şi de obicei aici se asociază layout-ul pentru fereastră prin metoda “setContentView()”, şi se realizează operaţiile necesare a fi făcute înainte ca utilizatorul să poată interacţiona cu aplicaţia. Obiectul “Bundle”, reprezintă principala modalitate de transmitere a

structurilor de date primitive către un Activity.

-”onPause()” - Atunci când utilizatorul părăseşte aplicaţia, sau când aceasta ajunge să fie rulată în fundal, este apelată această metodă pentru a oferi posibilitatea de a salva starea aplicaţiei înainte ca aceasta să fie posibil închisă de către managerul de memorie al sistemului de operare.

Atunci când utilizatorul accesează un nou Activity într-o aplicaţie, se opreşte ultimul Activity în care a fost utilizatorul, şi se salvează într-o stivă numită “back stack”, pentru a putea reveni la acestea, în ordine inversă, prin apăsarea butonului back. Pe lângă cele două metode care sunt obligatoriu de implementat, un activity conţine şi alte metode callback, cum ar fi onDestroy(), onResume(), etc., care permit manipularea unui Activity înainte ca starea acestuia să se modifice conform ciclului sau de viaţă.

După cum se observă în ilustraţia 1, aceste metode callback reprezintă stări prin care un Activity poate trece pe parcursul acestui ciclu de viaţă.

Pentru a putea fi lansat, un Activity trebuie mai întâi declarat în fişierul manifest al aplicaţiei din care face parte. De aici se poate seta un nume pentru acesta sau o temă grafică.

Oricarui Activity, i se asociază un obiect de tip Context, ce oferă acces la resursele unei componente a aplicaţiei. [1]

2.1.3.3 Intent

Un Activity poate fi lansat din altul, prin intermediul unui obiect numit Intent. Acesta este o descriere abstractă a unei operaţii ce urmează a fi făcută. Printr-un Intent se poate porni un Activity, un serviciu, sau pentru a livra un broadcast către un BroadcastReceiver.

Exista două tipuri de Intent:

- **explicit**: este specificat complet numele componentei care urmează a fi pornită. Această metodă este utilizată de obicei atunci când este pornită o componentă din cadrul aceleiaşi aplicaţii, întrucât este cunoscut numele componentei.

- **implicit**: nu specifică numele componentei la care se referă, însă oferă o descriere generală a acţiunii ce trebuie îndeplinită, şi lasă componenta (de obicei aflată în cadrul altei aplicaţii) să facă identificarea. Pentru acest tip de Intent este necesară declararea unui Intent

Filter în cadrul fişierului manifest al aplicaţiei. Prin acesta, se specifică tipul acţiunii care trebuie realizată, pentru a putea fi identificată de către componenta care are nevoie de ea.

Pentru pornirea serviciilor nu este indicat să se folosească Intent implicit, deoarece pune probleme de securitate, astfel începând cu versiunea 5.0, acest lucru duce la aruncarea unei excepţii.

Unui Intent i se pot ataşa date grupate sub forma unui obiect de tip Bundle. Acesta este împachetat şi trimis cu ajutorul Intent-ului, şi este disponibil în callback-ul onCreate().

2.1.3.4 Service

O altă componentă extrem de importantă în aplicaţiile Android, sunt serviciile.

Un serviciu realizează operaţii de lungă durată, în fundal, şi nu oferă o interfaţă pentru utilizator. Serviciile sunt pornite din orice componentă a unei aplicaţii, folosind Intents, şi rulează pe acelaşi thread ca şi acestea (de obicei, acesta se numeşte Main Thread, dacă este vorba de un Activity), astfel încât dacă utilizatorul doreşte să execute o acţiune în fundal, care este costisitoare sau blocantă, este indicat să creeze un thread separat pentru aceasta, în cadrul unui serviciu. Adiţional, orice componentă a unei aplicaţii, se poate lega la un serviciu (face bind), pentru a interacţiona în mod direct cu acesta.

Un serviciu poate lua două forme:

- **pornit**: atunci când o componentă a aplicaţiei îl porneşte prin metoda startService(), acesta porneşte şi execută în fundal o acţiune. El funcţionează independent de componenta care l-a pornit, execută o singură operaţie, şi nu returnează nici un rezultat.

- **legat (bound)**: Un serviciu este legat atunci când o componentă apelează metoda bindService(). Un serviciu pornit astfel, oferă o interfaţă de tip client-server între acesta şi componenta care s-a legat la el.

Serviciile, întocmai ca Activities, trebuie declarate în fişierul manifest al aplicaţiei. Pentru a crea un serviciu, trebuie subclasată clasa Service şi trebuie implementate o serie de metode callback:

- **onStartCommand()**: este apelată de sistem atunci când serviciul este pornit cu metoda startService().

- **onBind()**: este apelat de sistem atunci când o componentă doreşte să facă bind la serviciu.

- **onCreate()** şi **onDestroy()**: metode apelate de sistem atunci când intervin schimbări în ciclul de viaţă al serviciului.

Atunci când sistemul de operare observă că nu este suficientă memorie disponibilă, poate exista riscul ca serviciul să fie oprit, pentru a face loc altor aplicaţii cu importanţă mai mare. Acest comportament poate fi prevenit prin rularea serviciului în foreground. Această metodă presupune crearea unui obiect de tip Notification şi legarea acestuia la serviciu prin metoda startForeground(). În acest mod, se asigură faptul că serviciul nu va fi oprit de către sistemul de operare, și îi este atribuită o notificare pentru a fi vizibil utilizatorului faptul că serviciul este pornit.

Conform ciclului de viață al serviciilor pornite prin metoda startService(), acestea rulează indefinit în fundal, și trebuie oprite, fie apelând metoda stopSelf(), fie dintr-o componentă a aplicației prin metoda stopService(). În cazul serviciilor de tip bound, sistemul va opri automat serviciul în momentul în care toți clienții se deconectează de la serviciu, prin metoda unbindService(). [8]

2.1.3.5 Broadcast receiver

Un BroadcastReceiver este o componentă a aplicației care răspunde la mesaje transmise de către sistemul de operare. Aceste mesaje, de obicei se referă la schimbarea stării sistemului, cum ar fi: întoarcerea dispozitivului la 90 de grade, bateria se apropie de o anumită limită, a fost primit un mesaj SMS, etc. Întocmai ca serviciile, BroadcastReceiverele, nu au o interfață cu utilizatorul, însă pot avea o notificare în bara de stare.

Pentru a porni un BroadcastReceiver, acesta trebuie înregistrat la obiectul Context al componentei căreia dorim să îl asociem, utilizând context.registerReceiver(), sau trebuie declarat în fișierul manifest al aplicației. Trebuie menționat că un BroadcastReceiver este bine să fie oprit în callback-ul onPause() și repornit în callback-ul onResume(), deoarece nu funcționează atunci când componenta care l-a pornit este oprită.

În cazurile în care dezvoltatorul dorește să folosească un BroadcastReceiver creat de el,

doar în aplicația sa, este o practică bună ca acesta să fie declarat utilizând LocalBroadcastManager, dezvoltatorul eliminând riscul ca alte aplicații care rulează concomitent să interacționeze în vreun fel cu acesta.

Un BroadcastReceiver trebuie să implementeze callback-ul onReceive() care este apelat de către sistem atunci când este detectat un eveniment pentru care acesta a fost înregistrat.

2.1.3.6 Content Provider

Un ContentProvider este un manager de acces la date, care sunt disponibile din alt proces din sistem. Pentru a avea acces la aceste date, se folosește un ContentResolver împreună cu Context-ul componentei care dorește să comunice cu ContentProviderul, din rolul de client. ContentResolver-ul comunică cu o instanță a unei clase ce implementează ContentProvider.

Pentru a obține accesul la date din alte aplicații, nu este necesar să se implementeze un ContentProvider, ContentResolver-ul fiind suficient pentru îndeplinirea acestei operații. Implementarea devine necesară doar în cazurile când dezvoltatorul dorește să ofere acces la date din aplicația proprie.[3]

2.1.3.7 Shared Preferences

SharedPreferences reprezintă modul ideal de stocare a datelor în cadrul unei aplicații într-o structură de tip cheie-valoare. Aceste date sunt disponibile doar aplicației care le utilizează, și sunt disponibile oricărei componente din aplicație, însă pot fi utilizate și de aplicații externe, folosind ContentProvider.

Clasa SharedPreferences oferă un framework generic prin care se pot stoca și interoga date persistente în aplicații. Folosind această metodă se pot stoca doar valori primitive.

Framework-ul pune la dispoziție două metode de utilizare:

- **getSharedPreferences()**, utilizat atunci când avem nevoie de mai multe obiecte de tip SharedPreferences în cadrul aceleiași aplicații, ele diferențiindu-se prin nume;

- **getPreferences()**, folosit în cazurile când un singur obiect pentru stocare este suficient.

Pentru modificarea valorilor stocate, după crearea unui obiect SharedPreferences, trebuie instanțiat un obiect de tip SharedPreferences.Editor(). După ce au fost făcute modificările, trebuie apelată metoda commit() a editorului pentru a salva modificările în fișier. [9]

2.2 Aplicația web

2.2.1 Node.JS

Node.js este o platformă de javascript construită peste motorul V8 din browserul Google Chrome, special dezvoltată pentru a susține arhitecturi de tip client-server. Acesta oferă un mediu de lucru asincron, bazat pe evenimente și pe operații input-output non-blocante și este gândit să funcționeze ca un sever de sine stătător.

Node.js a apărut în anul 2009, fiind dezvoltat de câteva persoane. De-a lungul timpului au fost lansate numeroase module sub forma unor framework-uri sau librării pentru a extinde funcționalitățile de bază și pentru a ușura munca dezvoltatorilor. Acestea sunt puse la dispoziție prin intermediul unui manager de pachete denumit *npm* (node package manager).

Diferența dintre Node.js și alte limbaje server-side, este că Node.js execută instrucțiunile în mod non-blocant, adică pentru a executa o instrucțiune, nu este neaparat necesar ca cea precedentă să își fi terminat execuția. Pentru asigurarea ordinii de execuție, sunt folosite preponderent funcții callback, care sunt notificate în momentul în care funcția care o apelează, și-a încheiat execuția. Acesta este principalul motiv pentru care aplicațiile dezvoltate pe platforma Node.js sunt mai rapide, și scalabile la un numar mai mare de utilizatori.

Scalabilitatea este realizată pe alte platforme folosind threaduri, însă Node.js rulează pe un singur thread, și datorită caracterului asincron, cu apeluri I/O nonblocante, asigură suport pentru zeci de mii de conexiuni pe același thread, eliminând costurile impuse de *thread context switching*.

Dezavantajul platformei Node.js este că nu permite scalarea serverului în funcție de numărul de procesoare puse la dispoziție. Acest lucru este totuși posibil folosind module adiționale. [13]

2.2.2 Express.js

Express.js este un modul de Node.js, ce integrează un framework pentru aplicatii web. Express poate fi complementat de nenumărate alte module care au rol de *middleware*. Odată configurat, după ce este pornit, serverul ascultă la portul setat de dezvoltator și prelucrează cererile HTTP, redirectează către paginile aplicației și oferă interacțiune completă cu toate componentele server-side.

Rutarea în cadrul aplicației web, are o importantă ridicată în Express, deoarece acestea determină comportamentul aplicației la cererile HTTP venite de la client. Orice rutare are una sau mai multe funcții handler asociate, care se execută atunci când este primită o cerere de la client care se potrivește cu calea rutării.

Express folosește un *middleware* inclus, numit *static*, care se ocupă de gestionarea resurselor publice, cum ar fi pagini css, imagini, pagini html, etc. Structura unei aplicații Express este în esență o serie de apeluri la diverse *middlewares*. Acest lucru asigură o scalare a funcționalităților serverului pe orizontală, putând fi adăugate oricate astfel de module secundare care să manipuleze modul în care sunt tratate cererile primite de la client.

Un alt aspect important al modulului Express, îl reprezintă posibilitatea de a utiliza orice motor de templating dorit, cum ar fi Jade sau EJS. Acestea permit crearea de template-uri ale paginilor web, oferind posibilitatea de a genera dinamic conținutul acestora în funcție de anumiți factori specifici.

2.2.3 Passport.js

Un modul de Node.js care este *middleware* pentru Express, este Passport.JS. Acesta este un framework care poate fi integrat într-o aplicație Express și gestionează interacțiunea dintre view și baza de date în procesul de autentificare sau de înregistrare a utilizatorilor.

Passport oferă suport atât pentru sistemele de autentificare clasice, de tip user/parolă care sunt stocate în baza de date a serverului, cât și tehnologiile Oauth 2.0 utilizate de Facebook și Twitter sau OpenID utilizat de Google. Acesta oferă posibilitatea utilizatorilor de a se autentifica fie prin cont local, fie utilizânt propriul cont de Google, Facebook, Twitter, etc., contul de utilizator rămânând același, atât timp cât adresa de email folosită la înregistrare este aceeași.

Passport.js dispune de o multitudine de API-uri pe care dezvoltatorul le poate intregra, pentru ca utilizatorii să se autentifice. Acestea sunt implementate sub forma unor strategii, și modulul oferă acces la aproximativ 300 de astfel de strategii, dar și posibilitatea implementării unora customizate după nevoile aplicațiilor.

2.2.4 EJS

EJS este un motor de templating care folosit împreună cu Express oferă posibilitatea de a crea pagini html cu conținut dinamic în funcție de necesități. EJS oferă posibilitatea de a scrie cod javascript în fișierele html pentru a permite manipularea interfeței cu utilizatorul. Codul javascript trebuie scris între tagurile <% și %>. Folosit împreună cu Express, din partea de rutare a paginilor se pot trimite structuri de date din server către paginile html. Valorile variabilelor pot fi afișate în interfață prin intermediul tagului <%= %>.

Un alt avantaj al utilizării motorului de templating EJS, pe lângă dinamism, este acela că paginile HTML au mai puțin cod, și cantitatea de cod repetitiv din aplicație este redus la minimum.

2.2.5 Socket.io

Modulul Socket.io reprezintă o implementare a protocolului de comunicare WebSocket. Acesta oferă o singură conexiune TCP *full-duplex*. WebSocket a fost creat pentru a fi implementat în serverele web și în browsere, pentru a crea o punte de legătură între acestea. Singura legătură a acestui protocol cu protocolul HTTP este *handshake*-ul inițial care este interpretat de serverele HTTP ca un *upgrade-request*. Protocolul WebSocket oferă posibilitatea de a crea o interacțiune mult mai fluidă între serverele web și browsere, fiind tehnologia ideală pentru aplicațiile în timp real.

Socket.io poate fi integrat in serverul Express, pentru a oferi mai multe modalități de conexiune la același server (express realizând conexiunea HTTP și Socket.io realizând conexiunea prin WebSocket) . Integrarea se realizează legarea modulului la serverul Express, dupa care cele două modalități de conexiune rulează în paralel.

Odata realizată conexiunea, socketul creat poate să emită mesaje de forma mSocket.emit('message', data); și ascultă mesaje printr-un listener de forma mSocket.on('another_message', function(received_data)); unde funcția dată ca parametru este o funcție callback ce se execută separat pentru fiecare instanță a mesajului 'another_message' primită de socket.

Pentru Socket.io există și o implementare în limbajul Java, sub forma unei librării. Utilizând această implementare se poate realiza o conexiune prin WebSocket din orice aplicație Java, la orice server web ce permite astfel de conexiuni. Astfel se deschid noi oportunități și moduri de utilizare al aplicațiilor web, spre exemplu împreună cu dispozitive ce rulează sistemul de operare Android. [16]

2.2.6 MongoDB

MongoDB este o bază de date de tip NoSQL, deoarece nu implementează modelul tradițional SQL de baze de date relaționale. Punctul forte al acestui tip de baze de date îl reprezintă rapiditatea execuției operațiilor de tip CRUD (Create/Read/Update/Delete). Tabelele în MongoDB poartă denumirea de *schema* și elementele care le populează se numesc *documente*. Structura unui document este asemănătoare cu structura unui obiect JSON, fapt ce face ca integrarea în aplicațiile scrise în limbajul javascript să se realizeze cu ușurință.

Principalele avantaje ale MongoDB sunt:

-utilizarea interogărilor *Ad-hoc*, care permit căutarea în baza de date folosind expresii regulate și includerea unor funcții javascript

-posibilitatea de a indexa orice câmp dintr-un document

-replicarea bazei de date, pentru a asigura rezistența la eșecuri

-*load balancing* – scalarea pe orizontală folosind conceptul de *sharding*. Acesta presupune distribuirea bazei de date în *shards* în funcție de valoarea unei chei, și posibilitatea de a rula baza de date distribuită între mai multe servere.

-posibilitatea de a folosi baza de date ca un sistem de fișiere. MongoDB implementează o funcție numită GridFS, ce permite stocarea fișierelor de dimensiuni mari în baza de date. Într-un sistem distribuit, fișierele pot fi distribuite între mai multe mașini, ducând la crearea unui load-

balancing mai bun și la o toleranță mai mare la erori.

2.2.7 Mongoose

Mongoose este modulul de Node.js ce asigură conexiunea la baza de date MongoDB. Acesta oferă o implementare ce pune accentul pe manipularea datelor și ușurează lucrul cu acestea. Mongoose permite crearea unui model ce reprezintă *schema* pe care dorim să o utilizăm din baza de date. Operațiile se realizează ulterior pe acest model, dezvoltatorul având la dispoziție metode cum ar fi *findOne(), update(),* și *findOneAndUpdate(),*care le îmbină pe celelalte două.

2.2.8 Heroku si MongoLabs

Heroku este o platformă pentru hostarea aplicațiilor în cloud, și oferă atât servicii de bază în mod gratuit dar restricționat din punctul de vedere al facilităților și traficului, cât și soluții enterprise. Uploadarea aplicației se face relativ ușor, fiind necesar ca aplicația să existe pe un Git repository. Heroku oferă spre descărcare un client local prin intermediul căruia se realizează actualizarea aplicației în cloud sau vizualizarea fișierului log pentru monitorizarea outputului aplicației. O dată ce aplicația este uploadată, îi este creat un subdomeniu ce constă în două cuvinte aleatoare și un cod din 4 cifre, având asociat TLD-ul heroku.com.

Heroku oferă diverse pluginuri printre care și cel al companiei MongoLabs, acesta oferind posibilitatea de a crea și conecta o bază de date MongoDB la aplicația hostată.

3. Proiectarea aplicației

3.1 Structura Generală

Proiectul este divizat în două componente distincte: aplicația web și aplicația pentru dispozitivele Android. Aceasta din urmă este gândită să funcționeze în mod autonom pe parcursul unei perioade îndelungate, timp în care colectează evenimente produse în dispozitiv și le raportează către serverul aplicației web. Comunicarea dintre aplicații se realizează prin protocolul WebSockets. Serverul web primește informații de la diverse dispozitive ale utilizatorului și le afișează într-o interfață grafică intuitivă prin intermediul aplicației web. Utilizatorul poate sincroniza cu serverul web oricâte dispozitive Android pe care rulează aplicația.

3.2 Aplicația Android

3.2.1 Componentele principale și ordinea de execuție

Aplicația pentru dispozitivele Android este alcătuită din 4 mari componente:

-un *login Activity* din care utilizatorul se poate autentifica cu un cont existent.

-un *main screen Activity* ce reprezintă ecranul principal al aplicației, din care utilizatorul poate ajunge în meniul de setări sau poate opri aplicația.

-serviciul *ThreadDispatcher* care se ocupă de conexiunea cu serverul web, obținerea datelor din telefon și filtrarea acestora în funcție de preferințele utilizatorului

-*settings Activity* de unde utilizatorul își poate seta preferințele din cadrul aplicației

La lansarea aplicației, utlizatorului îi este prezentată fereastra de login, prin intermediul login *Activity*. În acest punct, în cazul în care dispozitivul nu are acces la internet, utilizatorul va fi notificat asupra acestui lucru printr-un mesaj de tip *Toast*. La momentul creării *Activity*-ului,

este pornit serviciul *ThreadDispatcher,* și utilizând variabila '*isloggedin'*, stocată cu *SharedPreferences,* se verifică dacă utilizatorul a rămas autentificat în aplicație. Acest caz este valid atunci când sistemul de operare oprește automat login *Activity*-ul, însă serviciul rămâne rulând în fundal. După ce *ThreadDispatcher*-ul este pornit prin metoda *startService(), Activity*-ul se leagă la acesta prin metoda *bind* pentru a putea apela metoda de autentificare din acesta.

Utilizatorul are la dispoziție două câmpuri de tip *EditText* în care poate introduce adresa de email și parola. După ce apasă pe butonul de *login* pentru a se autentifica, se face o verificare dacă au fost introduse date în ambele casete, și se transmit datele introduse către *ThreadDispatcher*. O dată cu datele introduse de utilizator, se adaugă suplimentar și un identificator unic al telefonului, împreună cu modelul acestuia, pentru a fi ușor utilizatorului de asociat datele afișate de către serverul web cu dispozitivul sincronizat. Dacă autentificarea se face cu succes, se creează un intent prin care se lansează *main screen Activity,* în caz contrar utilizatorul fiind notificat printr-o notificare de tip *Toast*, și câmpul dedicat parolei este golit.

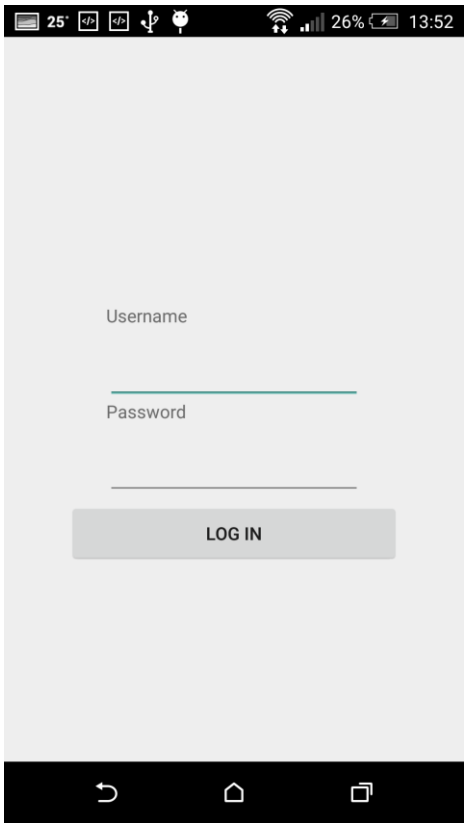


Illustration 2: login Activity

Serviciul *ThreadDispatcher* este pornit o dată cu aplicația, și este punctul central al acesteia. Conexiunea la internet este verificată în prealabil în login *Acitivty*, deci o re-verificare este redundantă. Serviciul inițializează un obiect de tip Socket din librăria Socket.io, realizează conexiunea la serverul web folosind adresa http://limitless-savannah-8511.herokuapp.com și inițializează toate callback-urile necesare pentru cazurile în care conexiunea nu reușește sau dacă socketul este închis. De asemenea sunt inițializate, cu valoarea false, o serie de valori booleene care înregistrează starea serviciilor puse la dispoziție de aplicație pentru serverul web. Aceste valori sunt folosite ca metodă de control a aplicației pentru a ști pe termen lung care servicii au fost pornite și care nu. Tot la pornirea serviciului se inițializează și un obiect de tip *Notification* care este utilizat pentru a menține rularea *ThreadDispatcher*-ului în foreground. În acest mod, serviciul nu va putea fi oprit decât manual de către utilizator prin operația de *logout.*

Serviciul mai execută un ultim pas, trimite un mesaj *local broadcast,* pentru a notifica *login Activity* că serviciul a fost creat, și că poate să facă *bind* la el.

Valorile implicite pentru toate serviciile pe care le gestionează *ThreadDispatcher*-ul sunt inițial setate ca false, însă odată ce utilizatorul face o modificare, *Service*-ul este notificat și se execută operațiile corespunzătoare.

ThreadDispatcherul poartă aceasta denumire deoarece se comportă ca un manager de threaduri, ocupându-se de pornirea și oprirea acestora conform preferințelor setate de utilizator. Astfel există threaduri separate pentru: receptarea mesajelor SMS și trimiterea lor către serverul web, receptarea mesajelor SMS de la serverul web, și trimiterea acestora în rețeaua de telefonie, verificarea statusului bateriei dispozitivului, trimiterea listei de contacte din dispozitiv către serverul web și localizarea dispozitivului în spațiul geografic și transmiterea coordonatelor longitudine și latitudine către serverul web.

Toate aceste threaduri funcționează independent unele de celelalte, însă raportează înapoi la *ThreadDispatcher* și rulează separat de *main thread,* deoarece pe lângă faptul că rulează interfața cu utilizatorul și aceasta nu trebuie să execute operațiuni de lungă durată, dacă sistemul de operare sau utilizatorul ar închide Activity-ul, ar fi oprite toate funcționalitățile aplicației.

Service-ul implementează și un *LocalBinder* pentru a permite componentelor să se lege la el prin metoda *bind*.

Callback-ul webSocket-ului care ascultă evenimentul *loginSucceeded,* este cel mai

important pas deoarece atunci când se execută, resetează variabilele corespunzătoare threadurilor din *SharedPreferences* cu valorile implicite și avansează starea aplicației. Execuția acestui callback reprezintă momentul inițial când dispozitivul este în proces de sincronizare cu serverul web. În acest interval, se transmit către serverul web starea bateriei dispozitivului cât și lista de contacte. Pentru a putea extrage contactele, *ThreadDispatcher*-ul creează un nou thread dedicat, pe care îl rulează.

În threadul de extragere al contactelor, este inițializată o structură de date de tip HashMap<String, String> pentru a memora în perechi cheie-valoare, unde cheia este numărul de telefon (pentru a asigura unicitatea), și valoarea asociată este numele persoanei de contact. Extragerea se realizează folosind un cursor împreună cu un *contentResolver*. Threadul face o cerere la componența *Phone a sistemului* folosind metoda *ContactsContract*, și înregistrează un *ContentResolver* la aceasta, obținând ca rezultat cursorul cu datele. După ce datele sunt extrase în structura HashMap, folosind un iterator, se parcurge această structură și se crează un obiect JSON care conține aceleași date, după care, la finalizare, se emite prin WebSocket un mesaj conrespunzator care conține lista de contacte.

Concomitent este transmis statusul bateriei, operațiune realizată tot în cadrul unui thread separat. Acesta creează un intentFilter și înregistrează un *contentReceiver* la *context*-ul *ThreadDispatcher*-ului. Intentul obținut de la acesta este mai apoi utilizat pentru a extrage datele despre statusul bateriei device-ului, și este emis un mesaj folosind WebSocket-ul care conține nivelul curent al bateriei. Acest proces de interogare al nivelului bateriei se realizează de fiecare dată când *ThreadDispatcher*-ul emite un eveniment folosind WebSocket-ul. Acest comportament nu este consumator din punct de vedere al energiei pentru dispozitiv (comparativ cu o interogare efectuată la intervale de timp regulate), și oferă o perspectivă reală asupra consumului de energie.

La încheierea rulării callback-ului evenimentului *loginSucceeded*, *ThreadDispatcher*-ul transmite un mesaj *local broadcast* destinat activității de autentificare, pentru a îi permite trecerea în următoarea stare a aplicației. Astfel, odată recepționat broadcastul de către *login Activity*, se crează un *Intent*, și se lansează *main screen Activity*.

Odată pornită, fereastra principală a aplicației, prima oară face bind la *ThreadDispatche*r care acum rulează în foreground, cu notificația aferentă în bara de stare a telefonului, după care execută o metodă acestuia *onOptionsChanged()*. Aceasta face ca *ThreadDispatcher*-ul să verifice

setul de valori booleene salvate în *SharedPreferences*, să le compare cu propriile valori locale, și să pornească sau sa oprească threadurile în funcție de necesități.

Din fereastra principală a aplicației, utilizatorul poate alege să se deconecteze de la aplicație, sau să deschidă fereastra de modificare a opțiunilor.

La apăsarea butonului *logout*, începe procesul de deconectare a aplicației de la serverul web, urmat de oprirea *ThreadDispatcher*-ului și a activității principale. ThreadDispatcher-ul este notificat să înceapă operațiunea de deconectare, accesează *SharedPreferences*, și reinițializează toate valorile cu cele implicite, după care va opri toate threadurile care rulau. După acești pași, este emis un semnal de deconectare de la ServerulWeb și WebSocketul este închis, după care serviciul este scos din foreground, și este oprit prin metoda *stopSelf()*. Este creat un obiect *Intent*, și se lansează în execuție *login Activity*, ajungându-se din nou în starea în care se afla aplicația atunci când este deschisă pentru prima oară.

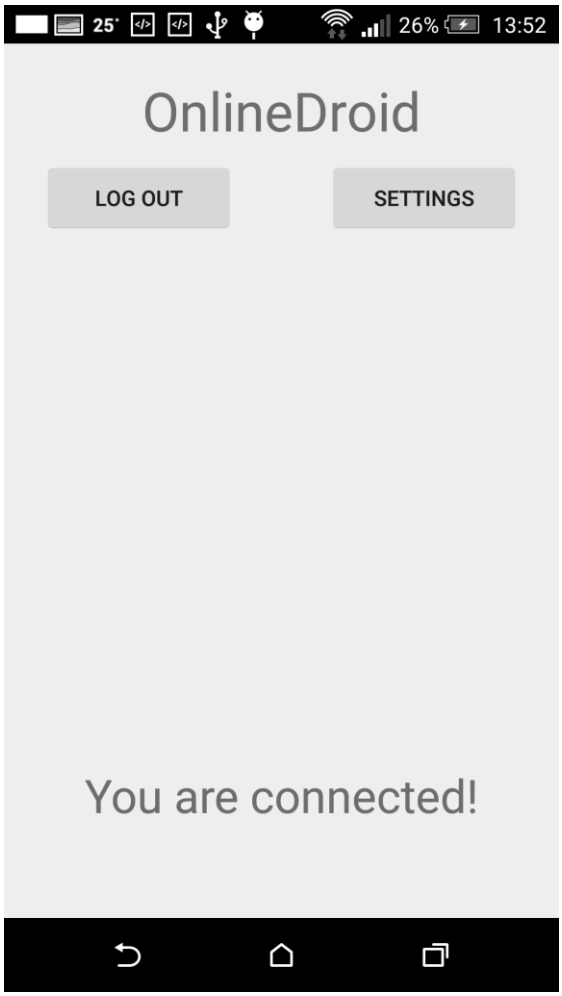


Illustration 3: Main screen Activity

La apăsarea butonului *settings*, se creează un *Intent*, prin care se lansează în execuție *settings Activity.* În această fereastră utilizatorului îi sunt prezentate sub forma unei liste însoțite de un *checkbox*, toate serviciile puse la dispoziție de către aplicație, împreună cu un buton pentru salvarea schimbărilor.

Odată creat acest *Activity*, înainte de a fi afișate opțiunile, se face o verificare pentru a vedea dacă dispozitivul suportă mesaje de tip SMS sau apeluri telefonice. În cazul în care dispozitivul nu suportă acest tip de operații, *checkbox*-urile aferente opțiunilor vor fi blocate, și nu vor putea fi pornite în nici un mod. Acest comportament asigură că nu se pot produce erori din cauza neatenției utilizatorului și că nu sunt pornite niciodată threaduri care nu sunt necesare aplicației. Pentru acest pas, este instanțiat un obiect de tip *TelephonyManager,* și se consulta starea cartelei SIM. Dacă aceasta are valoarea constantei SIM_STATE_ABSENT sau SIM_STATE_UNKNOWN, opțiunile de *send SMS, forward SMS,* și *notify calls,* nu vor fi disponibile în fereastra de setări.

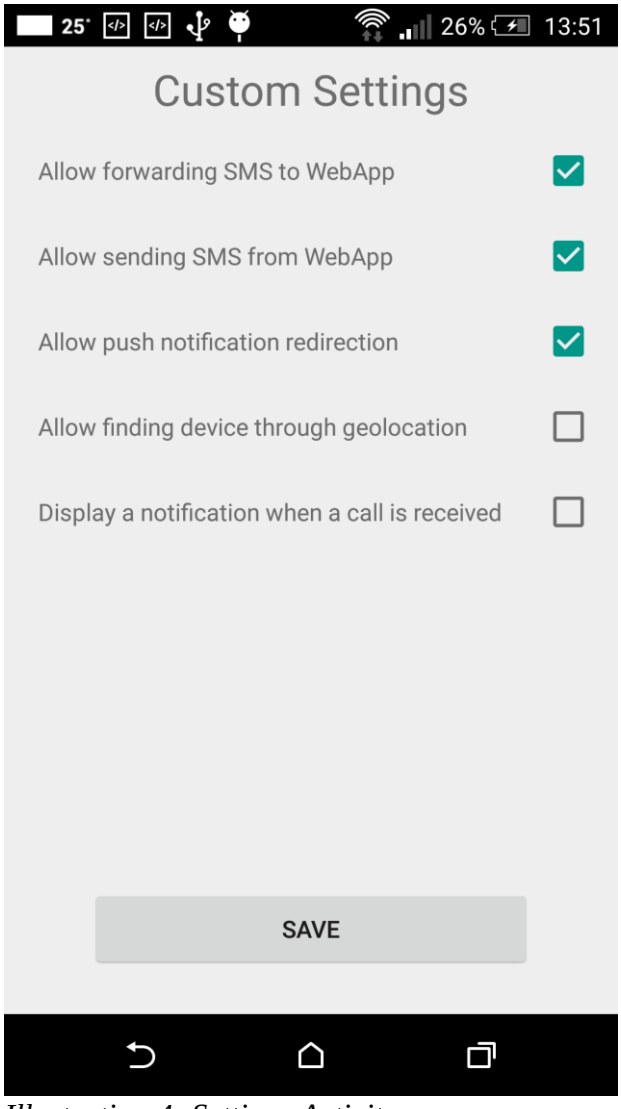


Illustration 4: Settings Activity

După ce utilizatorul modifică setările după preferințele sale, la apăsarea butonului *save*, este lansată în execuție o metoda ce salvează în *SharedPreferences* valorile booleene ale acestora, și creează un *Intent* pentru întoarcerea în activitatea principală a aplicației. Odată re-lansată, este apelată din nou metoda *onOptionsChanged()* din *ThreadDispatcher* și threadurile sunt pornite sau oprite în funcție de opțiunile selectate de utilizator.

3.2.2 Implementarea serviciilor oferite de aplicație

Threadul care se ocupă de transmiterea mesajelor SMS, recepționate de dispozitiv, către serverul web crează un *IntentFilter* către acțiunea sistemului *android.provider.Telephony.SMS_RECEIVED,* și înregistrează un *BroadcastReceiver* cu acesta. În implementarea acestuia, atunci când este recepționat un SMS, este extras din el numărul de telefon împreună cu mesajul propriu-zis, și se emite un eveniment *smsForward* împreună cu datele extrase, către serverul web. Threadul trebuie rulat o singură dată, pentru a înregistra *BroadcastReceiverul* după care până când nu este apelată explicit metoda *unregisterReceiver*, acesta continuă să asculte sistemul pentru acțiuni de tip sms recepționat.

Threadul care trimite mesajele primite de la serverul web, folosind rețeaua de telefonie utilizează un obiect *SmsManager*, cu ajutorul căruia trimite mesajul SMS. Atunci când threadul recepționează pe webSocket mesajul *smsSend,* va prelua datele primite, le va seapara ca mesaj, și număr de telefon, după care verifică lungimea mesajului. Dacă acesta depașește 160 de caractere, mesajul va fi întâi despărțit și apoi împachetat ca mesaj SMS și trimis cu ajutorul *SmsManager*-ului.

Pentru raportarea poziționării geografice a dispozitivului, este pornit un thread care lucrează cu *Google Services API*, pentru a determina locația. Acest API nu interoghează o componenta hardware anume, cum ar fi GPS sau Wi-Fi, ci verifică cu ajutorul unei componente numite *FusedLocation,* care este ultima locație geografică înregistrată în care s-a aflat dispozitivul. În acest fel, se obține o locație precisă, fără a fi nevoie ca dispozitivul să aibă în permanență pornit serviciul GPS care ar putea duce la un consum mărit al bateriei.

Threadul va instanția un obiect de tip *GoogleApiClient* folosind un *Builder*, și API-ul

LocationServices. Odată conectat la API-ul Google Services, se obține ultima locație disponibilă într-un obiect de tip *Location*, din care ulterior se extrag cele doua valori: longitutine și latitudine.

Odata aflate, acestea se transmit către serverul web prin emiterea cu webSocket a mesajului *geoloc*, împreună cu datele extrase. În cazul în care Google API nu reușește să se conecteze, este emis un mesaj către serverul web prin care se specifică faptul că nu s-a putut obține o locație pentru dispozitiv. [12] [11]

În cazul sincronizării notificărilor din dispozitiv cu serverul web, nu este folosit un thread, ci un obiect *NotificationReceiver,* care este o subclasare a unui *BroadcastReceiver,* împreuna cu un *NotificationListener.* Receiverul ascultă mesaje Local broadcast de tip *notification_event*, și emite către serverul web datele primite. Listenerul funcționează ca o componentă *Service*, și este notificată atunci când un *push notification,* este livrat de către o aplicație din dispozitiv. Atunci când un astfel de eveniment este detectat, se extrag numele pachetului aplicației care a lansat notificarea, titlul notificării și textul acesteia. Aceste date sunt împachetate și transmise printr-un intent *BroadcastReceiver*-ului, care o transmite mai departe serverului web.

ThreadDispatcher-ul lansează o notificare în bara de notificări, cu ajutorul căreia rulează *Service*-ul in *foreground.* Utilizatorul poate închide toate activitățile care rulează, și serviciul va ramâne rulând în fundal.

Atunci când serviciul încă rulează (notificarea este prezentă în bara de notificări) , dacă utilizatorul deschide aplicația, îi va fi prezentat direct *main screen Activity*. La lansarea aplicației, se verifică dacă există credențialele utilziatorului în *SharedPreferences*, pentru a evita afișarea ecranului de autentificare și re-pornirea serviciului prin acțiunea de autentificare. Atunci când utilizatorul apasă pe notificare, se lansează *main screen Activity*.

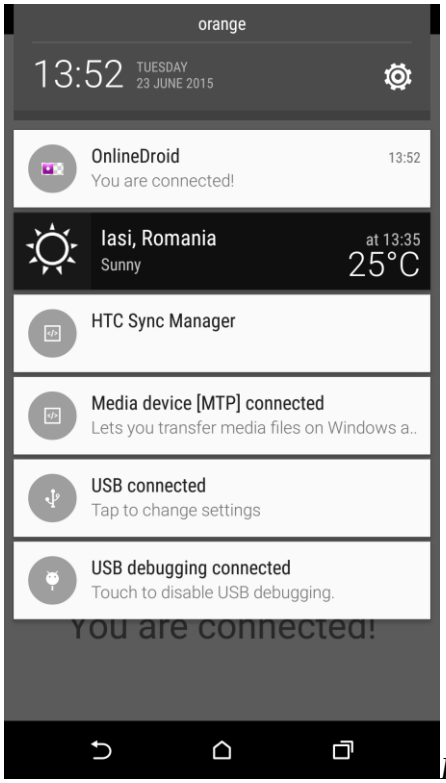


Illustration 5: Notificatia ThreadDispatcher-ului

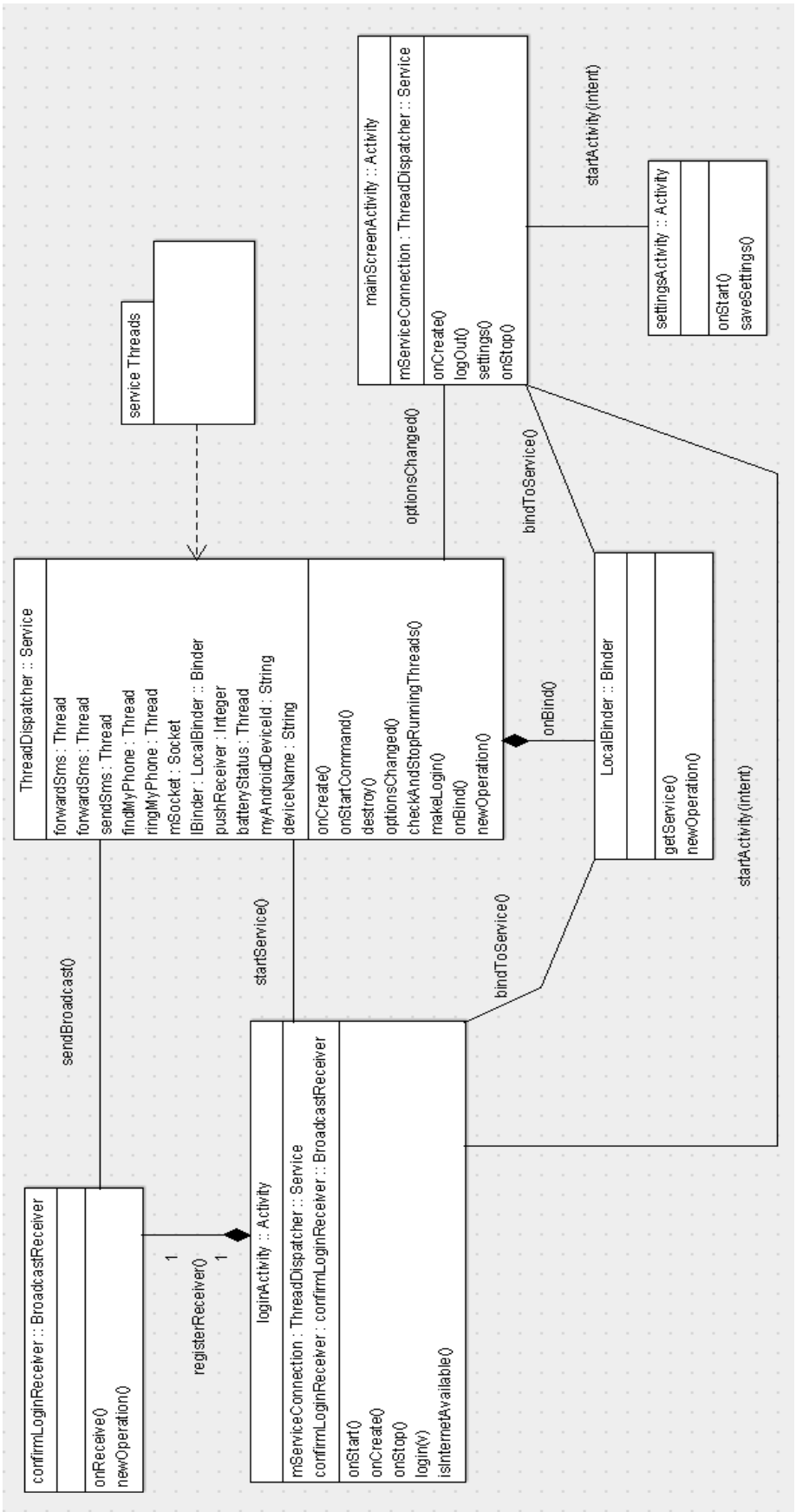
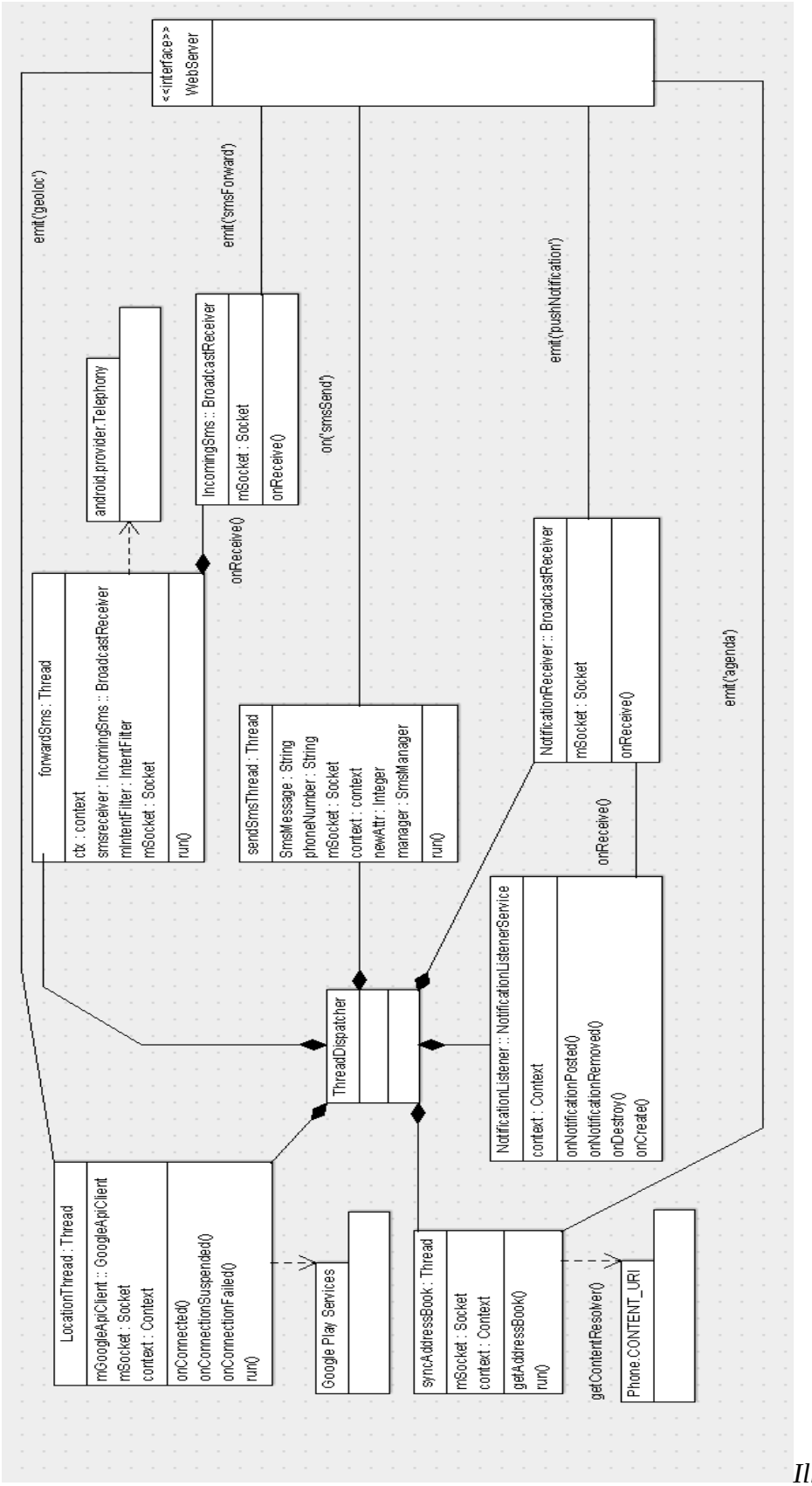


Illustration 6: Diagrama de clase a aplicatiei Android



Ill ustration 7: Diagrama Threadurilor si interactiunea cu serverul web

3.3 Aplicația Web

Utilizatorul interacționează în mod direct cu aplicația web prin intermediul browserului. După ce se autentifică, i se prezintă interfața grafică prin care se poate autentifica și are la dispoziție toate informațiile colectate din dispozitivele sincronizate.

Aplicația rulează pe un server Express, configurat împreună cu frameworkul Passport.js care se ocupă atât de autentificarea și înregistrarea utlizatorilor în sistem cât și de managementul sesiunilor pentru interacțiunea din browser. Fiecare instanță a aplicației Android se conectează la server, și comunică cu acesta în permanență prin intermediul unui web socket. Acesta ascultă inițial pentru mesaje de autentificare, și odată ce aceasta se realizează cu succes, serverul ascultă la acel socket mesaje pentru sincronizarea inițială a agendei, setările din aplicația Android și a nivelului bateriei. Acești pași reprezintă sincronizarea dispozitivului cu serverul web, aceste informații fiind necesare pentru afișarea corectă a informațiilor despre dispozitiv în interfață.

După ce sincronizarea se realizează cu succes, aplicația va asculta toate tipurile de mesaje transmise de către aplicația Android. Serverul nu face verificări suplimentare asupra acestor mesaje. Odată ce serverul recepționează un tip de mesaj, el este tratat corespunzător.

Încărcarea informațiilor primite de la server se realizează în timp real, și se execută prin intermediul modulului socket.io. Când un client se conecteaza la server, după ce trece de etapa de autentificare, din pagina principală a clientului, se deschide un socket către serverul web. Acesta este folosit pentru a sincroniza în timp real mesajele recepționate de pe dispozitivul utilizatorului. Atunci când socketul clientului stabilește o conexiune cu serverul, socketul va deveni participant într-un room care va fi identificat unic prin numele sau de utilizator.

Room-urile au rolul unor canale de comunicare închise, prin care socketii pot emite mesaje doar spre anumite socketuri. Astfel fiecare utilizator are propriul room, iar mesajele venite de la toate dispozitivele sale sincronizate ajung doar în acest room, acesta având rolul de filtru de mesaje.

Atunci când serverul primește un mesaj de la un socket al unei aplicații Android autentificată, utilizând același cont ca și pagina clientului, informațiile oferite de dispozitiv vor fi redirectate către room-ul clientului, realizând o actualizare în timp real a paginilor web. În acest mod, fiecare conexiune a unei pagini client coexista în același room împreuna cu conexiunile tuturor dispozitivelor utilizatorului.

Odată ce un dispozitiv este sincronizat, el va apărea în bara de navigare din partea stângă a interfeței grafice. Aici sunt vizibile informații despre dispozitiv cum ar fi numele acestuia, nivelul la care se află bateria și numărul de notificări noi și de mesaje SMS necitite. În partea dreaptă, fereastra principală afișează în mod implicit la pornire, interfața pentru convorbiri prin mesaje SMS. Această fereastră este divizată în două părți. În partea dreaptă se află agenda telefonică afișată sub forma unei liste, ordonată alfabetic după numele persoanei de contact, iar în partea stângă se află fereastra de convorbire propriu zisă. Atunci când este primit un mesaj, este incrementat numărul de mesaje necitite din coloana din partea stângă a interfeței și mesajul este afișat în fereastra de conversație în partea dreaptă. Utilizatorul poate folosi caseta de input pentru a compune un mesaj și prin apăsarea tastei enter sau prin apăsarea butonului *Send* mesajul este trimis prin intermediul webSocket-ului către smartphone. Atunci când utilizatorul selectează în agenda din partea dreaptă un contact, fereastra de conversație adiacentă se va modifica pentru a arăta ultimele mesaje interschimbate cu acel contact.

În cazul în care un dispozitiv primește un mesaj SMS de la un număr de telefon care nu se află în agendă, contactul acesta va apărea în agendă, ultimul din listă, și în loc să se identifice prin nume, se va identifica prin numărul de telefon. Funcționalitatea rămâne în rest neschimbată.

În partea superioară a interfeței există o bară cu butoane cu rolul de navigare prin aplicație. Fiecare opțiune din bara de navigare modifică doar fereastra adiacentă listei de dispozitive sincronizate cu serverul. Această abordare asigura un layout dinamic al aplicației și scutește utilizatorul de reîncărcări ale paginii. Dispozitivul pentru care sunt afișate aceste ferestre este cel curent, adică cel care a fost selectat ultima oară, sau dacă nu a fost selectat nici unul, implicit va fi primul din lista din partea stângă.

Butonul *Call log,* afișează istoricul apelurilor recepționate de către server. Acestea sunt afișate în funcție de data când au fost recepționate și se identifică prin numele contactului sau numărul de telefon de la care a fost primit apelul, pentru cazul când numele nu este cunoscut.

Butonul *Notifications* modifică aspectul ferestrei centrale, afișând notificările recepționate într-o structură de tip stivă. În această stivă apar în partea de sus cele mai noi notificări, utilizatorul având posibilitatea de a le șterge selectiv prin apăsarea unui buton adiacent fiecăreia. Pentru fiecare notificare sunt afișate: numele pachetului aplicației Android care a lansat notificarea pe dispozitiv, titlul notificării și textul acesteia într-o casetă de text ce poate fi expandată sau restrânsă atunci când este selectată de utilizator. În momentul în care notificarea

este transmisă din dispozitiv către serverul web, aceasta este automat eliminată din interfața dispozitivului, pentru a scuti utilizatorul de nevoia de a elimina a doua oară notificările în momentul în care utilizează dispozitivul.

Butonul *Location* afișează în fereastra centrală o hartă, prin pluginul Google Maps, pe care se va afla poziționat un pin la poziția unde se află dispozitivul. Această fereastră are mai mult efect informativ și nu oferă o interacțiune cu utilizatorul de nici un fel. Pentru reactualizarea locației este necesar ca utilizatorul să apese pe butonul Location.

Meniul de setări este disponibil pentru utilizator prin intermediul butonului Settings. De aici pot fi modificate setările care au fost realizate inițial pe dispozitivul mobil. Utilizatorul poate bifa sau debifa opțiunile, după care, prin apăsarea butonului *save settings*, noile setări sunt trimise către dispozitivul mobil.

Ultima opțiune pusă la dispoziție în bara de navigare este butonul de deconectare de la aplicație, prin care socket-ul clientului este oprit și utilizatorul revine la ecranul de autentificare.

În partea dreaptă a barei de navigare, utilizatorul va fi notificat de apelurile pe care le primește, dacă opțiunea aceasta a fost selectată. Notificarea apare sub forma unei casete plutitoare, care va conține numele contactului care apelează, sau numărul de telefon, dacă este un contact necunoscut, împreună cu numele dispozitivului pentru a asocia apelul recepționat cu dispozitivul. Utilizatorul are la dispoziție un buton prin care poate respinge apelul. În cazul în care apelul se încheie înainte ca utilizatorul să ia o acțiune, notificarea dispare, și este trecută în istoricul apelurilor din fereastra corespunzatoare.

4. Concluzii

4.1 Obiective îndeplinite

Obiectivele îndeplinite cu ajutorul aplicației sunt în principal ținerea la curent a utilizatorului în legatură cu starea dispozitivelor sale sincronizate. Toate notificările care sunt afișate în mod normal pe dispozitiv, sunt transmise către serverul web, împreună cu starea bateriei dispozitivului. Posibilitatea de a modifica setările fiecărui dispozitiv în parte, în mod independent și de a opri la cerere serviciul care rulează pe acesta.

Utilizatorul este notificat în timp real despre toate evenimentele, nefiind nevoie de reîmprospatarea paginii pentru a obține cele mai recente notificări. Acest lucru este posibil datorită protocolului webSocket ce permite serverului să notifice clienții despre modificările apărute, eliminând nevoia de a face polling.

4.2 Limitări ale aplicației

Una din limitările care se observă imediat în utilizarea aplicației este în stiva de notificări. Atunci când apare o notificare în stivă, îi sunt afișate numele pachetului, titlul notificării și textul acesteia. Forma numelui pachetului este greoaie și reprezintă succesiunea numelor pachetelor care formează aplicația care a lansat notificarea. Denumirea aplicației așa cum apare ea utilizatorului nu se poate obține, deoarece sistemul de operare atribuie denumirea aplicației automat numelui pachetului. Pentru ca acest lucru să fie posibil în aplicația web, ar trebui să existe o indexare de forma cheie-valoare unde cheia să fie numele pachetului (care este unic în dispozitiv) și valoarea să fie numele aplicației așa cum o vede utilizatorul pe dispozitiv. Pentru a obține această indexare ar trebui parcurse toate aplicațiile disponibile în Google Play Store, și asocierea numelui real cu numele pachetului, lucru care nu se poate face programatic, deci implică muncă manuală care nu este justificabilă pentru rezultatul obținut.

4.3 Cercetare și direcții ulterioare de dezvoltare

Ca direcție de cercetare asupra aplicației, am încercat să ajung la o posibilitate de a utiliza calculatorul pentru a putea purta conversații prin intermediul unui smartphone. Acest lucru s-a dovedit a fi de ne-realizat după mai multe încercări în acest sens. Purtarea conversațiilor la distanță utilizând aplicația web ar fi oferit mai mult control asupra smartphone-ului la distanță, și ar fi reprezentat o funcționalitate foarte importantă în cadrul aplicației.

Inițial am încercat să realizez o conexiune între smartphone și laptop utilizând serviciul DNS-SD (DNS - service discovery). Această tehnologie este un protocol de comunicare utilizând servicii publicate pe Local Area Network și este dezvoltată de compania Apple, prin intermediul serviciului Bonjour. O aplicație aflată pe un dispozitiv conectat în rețea poate publica un nume al serviciului oferit de aceasta, împreună cu tipul de protocol de comunicare pe care îl folosește. Aceste nume ale protocoalelor nu sunt standard, ci pot fi și implementate independent de către dezvoltator. Un alt serviciu aflat pe un calculator sau un alt dispozitiv conectat la aceeași rețea, poate interoga rețeaua pentru a obține o listă de servicii ce utilizează un anumit protocol. În acest mod se obține portul și adresa aplicației care a publicat numele serviciului, și conexiunea se realizează fără a avea hardcodat nici o adresă de conexiune în nici una din cele două aplicații.

După realizarea conexiunii, scopul era să manipulez conexiunea în așa fel încât smartphone-ul să identifice conexiunea cu aplicația din laptop ca fiind o conexiune la o cască bluetooth. Această tentativă a eșuat deoarece sistemul de operare Android, nu permite manipularea componentelor interne în așa fel încât conexiunea să pară altceva decât este de fapt.

A doua tentativă a fost de a încerca să obțin datele de voce în timpul unei conversații. După o perioadă de cercetare pe această temă, am concluzionat că aceasta abordare nu are posibilități de succes deoarece implică accesul la datele de voce direct din driver-ul audio al sistemului de operare. Pentru a putea realiza o implementare la nivel de driver, este nevoie de un telefon rootat. Odată obținut codul sursă al sistemului de operare, ar trebui realizată o injecție a streamului audio venit din baseband, prin rețea, în aplicație, și o injecție în sens invers a semnalului audio venit din aplicație în baseband. Chiar și considerând toate aceste operații ca fiind fezabile, ar trebui realizate aceleași operațiuni de extragere și injecție a streamului audio primit din rețea, în driver-ul audio aflat în calculator. Un alt motiv pentru care această operațiune nu s-ar realiza cu succes îl reprezintă diversitatea driverilor audio disponibili pe piață, și nevoia

de a avea la dispoziție o interfață generică de lucru cu aceștia, pentru a putea permite o singură implementare, pentru toate tipurile de driveri.

Operațiunea de root-are a telefonului presupune acordarea permisiunilor de administrator (root) pe telefon. Această modificare poate avea efecte nedorite la nivel de securitate a dispozitivului și nu este indicată de obicei. Așadar la nivel de driver, atunci când se inițiază un apel telefonic, streamul audio output este direcționat din microfon către un codec, după care în baseband, iar streamul input circulă în sens invers, din baseband, în codec după care în difuzor. Baseband-ul este cipul care realizează procesarea semnalului audio și transmisia radio a acestuia în timp real. Faptul că streamurile trec direct în baseband, și nu trec prin procesor face ca accesul la aceste date să nu fie disponibile pentru dezvoltatori prin intermediul unui API.

Această abordare s-a dovedit a fi neviabilă și în afara scopurilor aplicației, deoarece ar implica în primul rând rootarea smartphone-ului, operație ce trebuie realizată manual și nu este indicat spre a fi realizată de persoane care nu au cunoștințe în acest domeniu, în al doilea rând ar presupune rescrierea unei părți a sistemului de operare Android, fapt ce implică instalarea unui ROM Android ne-licențiat ce poate impune riscuri de securitate și instabilitate a sistemului de operare, și nu se dovedește a fi în favoarea utilizatorilor în nici un fel.

Bibliografie și referinte

[1]Android Activity. (n.d.). Retrieved from http://developer.android.com/guide/components/activities.html

[2]Android activity lifecycle. (n.d.). Retrieved from http://developer.android.com/training/basics/activity-lifecycle/index.html

[3]Android Content Provider. (n.d.). Retrieved from http://developer.android.com/guide/topics/providers/content-providers.html

[4]Android development guide. (n.d.). Retrieved from https://developer.android.com/guide/index.html

[5]Android history. (n.d.). Retrieved from http://arstechnica.com/gadgets/2014/06/building-android-a-40000-word-history-of-googles-mobile-os/

[6]Android M features. (n.d.). Retrieved from https://www.androidpit.com/android-m-release-date-news-features-name

[7]Android OS Wiki. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Android_%28operating_system%29

[8]Android services. (n.d.). Retrieved from http://developer.android.com/guide/components/services.html

[9]Android Storage. (n.d.). Retrieved from http://developer.android.com/guide/topics/data/data-storage.html

[10]Android version distribution. (n.d.). Retrieved from https://developer.android.com/about/dashboards/index.html?utm_source=suzunone

[11]Git Socket.io Java nkzawa. (n.d.). Retrieved from https://github.com/nkzawa/socket.io-client.java

[12]Google Services. (n.d.). Retrieved from https://developers.google.com/android/guides/setup

[13]Node.JS. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Node.js

[14]Statista Facebook. (n.d.). Retrieved from http://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/

[15]Statista Twitter. (n.d.). Retrieved from http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[16]WebSocket Wiki. (n.d.). Retrieved from https://en.wikipedia.org/wiki/WebSocket