UNIVERSITATEA “TITU MAIORESCU” DIN BUCUREŞTI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

COORDONATOR ŞTIINŢIFIC:

Conf. univ. dr. Ana Cristina Dăscălescu

ABSOLVENT:

Sabin Drăgan

SESIUNEA IUNIE

2021

UNIVERSITATEA “TITU MAIORESCU” DIN BUCUREŞTI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENŢĂ

Dezvoltarea unei aplicații pentru managementul proiectelor pe platforma Android

COORDONATOR ŞTIINŢIFIC:

Conf. univ. dr. Ana Cristina Dăscălescu

ABSOLVENT:

Sabin Drăgan

SESIUNEA IUNIE

2021

**UNIVERSITATEA TITU MAIORESCU**

**FACULTATEA DE INFORMATICĂ**

**DEPARTAMENTUL DE INFORMATICĂ**

***REFERAT***

***DE APRECIERE A LUCRĂRII DE LICENȚĂ/~~DISERTAȚIE~~***

***TITLU:­­­­­­­­­­­­­****\_\_\_\_\_\_\_\_* *Dezvoltarea unei aplicații pentru managementul proiectelor pe platforma Android\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***ABSOLVENT/ ~~MASTERAND~~:****\_\_\_\_\_* *Sabin Drăgan \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***PROFESOR COORDONATOR:****\_\_\_ Conf. univ. dr. Ana Cristina Dăscălescu\_\_\_\_\_\_\_\_*

Referitor la conținutul lucrării, fac următoarele aprecieri:

|  |  |  |
| --- | --- | --- |
| A. | Conţinutul ştiinţific al lucrării | 1 2 3 4 5 6 7 8 9 10 |
| B. | Documentarea din literatura de specialitate | 1 2 3 4 5 6 7 8 9 10 |
| C. | Contribuția proprie | 1 2 3 4 5 6 7 8 9 10 |
| D. | Calitatea exprimării scrise și a redactării lucrării | 1 2 3 4 5 6 7 8 9 10 |
| E. | Conlucrarea cu coordonatorul științific | 1 2 3 4 5 6 7 8 9 10 |
| F. | Realizarea aplicației practice | 1 2 3 4 5 6 7 8 9 10 |
|  | **Punctaj total** = (A+B+C+D+E+2F)/7 |  |

În concluzie, consider că lucrarea de licență/disertație întrunește/ nu întrunește condițiile pentru a fi susținută în fața comisiei pentru examenul de licență/disertație din sesiunea \_\_\_\_\_\_IUNIE\_2021\_\_\_\_\_\_\_\_\_\_ și o apreciez cu nota\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

***CONDUCĂTOR ȘTIINȚIFIC,***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Cuprins

[2 Introducere 5](#_Toc74556414)

[3 Capitolul I – Tehnologii utilizate 5](#_Toc74556415)

[3.1 Arhitectura Android 5](#_Toc74556416)

[3.1.1 Kernel Linux 5](#_Toc74556417)

[3.1.2 Strat de abstractizare hardware 6](#_Toc74556418)

[3.1.3 Bibliotecile platformei Android 7](#_Toc74556419)

[3.1.4 Runtime Android 8](#_Toc74556420)

[3.1.5 Framework aplicații 9](#_Toc74556421)

[3.1.6 Aplicații 10](#_Toc74556422)

[3.2 Java in contextul Android 14](#_Toc74556423)

[3.3 SQLite 15](#_Toc74556424)

[3.4 RecyclerView si ViewPager2 17](#_Toc74556425)

[3.5 Fragmente 17](#_Toc74556426)

[4 Capitolul II - Etapele de dezvoltare a unei aplicații Android 17](#_Toc74556427)

[4.1 Cercetare 17](#_Toc74556428)

[4.2 Schițare 17](#_Toc74556429)

[4.3 Studiu tehnic de fezabilitate 17](#_Toc74556430)

[4.4 Prototipare 17](#_Toc74556431)

[4.5 Proiectare 17](#_Toc74556432)

[4.6 Dezvoltare 17](#_Toc74556433)

[4.7 Lansarea aplicatiei 17](#_Toc74556434)

[5 Capitolul III – Aplicații utilizate 17](#_Toc74556435)

[5.1 Git 18](#_Toc74556436)

[5.2 GitHub 18](#_Toc74556437)

[5.3 Android Studio 18](#_Toc74556438)

[5.4 Gradle 18](#_Toc74556439)

[5.5 Android emulator - qemu 18](#_Toc74556440)

[5.6 Trello 18](#_Toc74556441)

[6 Capitolul IV – Arhitectura aplicației 18](#_Toc74556442)

[6.1 Clasele model 18](#_Toc74556443)

[6.2 Structura bazei de date 18](#_Toc74556444)

[6.3 Autentificare – Înregistrare 18](#_Toc74556445)

[6.4 ViewPager2 18](#_Toc74556446)

[6.5 Fragmente cu RecyclerView 19](#_Toc74556447)

[6.6 Fragmente de editare 19](#_Toc74556448)

[7 Capitolul V – Funcționalitatea aplicației 19](#_Toc74556449)

[7.1 Activitatea de autentificare 19](#_Toc74556450)

[7.2 Activitatea de înregistrare 19](#_Toc74556451)

[7.3 Activitatea principala 19](#_Toc74556452)

[7.4 Navigarea 19](#_Toc74556453)

[7.5 Operații cu Task-uri 19](#_Toc74556454)

[7.6 Operații cu liste de Task-uri 19](#_Toc74556455)

[8 Testarea aplicației 19](#_Toc74556456)

[9 Concluzii 20](#_Toc74556457)

[10 Bibliografie 21](#_Toc74556458)

# Introducere

# Capitolul I – Tehnologii utilizate

## Arhitectura Android

Arhitectura Android conține diverse componente folosite pentru a suporta nevoile oricărui  dispozitiv Android. Aceasta conține un sistem de operare open source bazat pe kernel-ul Linux și o colecție de biblioteci C și C++ care sunt expuse prin servicii de framework de aplicație.

Printre componentele arhitecturii găsim kernel-ul Linux care furnizează funcționalitatea principală a sistemului de operare și mașina virtuală Dalvik(DVM) care furnizează o platformă pentru rularea aplicațiilor Android, mai nou înlocuită de ART(Android RunTime)



Figura 3.1 - Stiva de software Android

<https://developer.android.com/guide/platform#native-libs>

Componentele principale ale arhitecturii Android

* Aplicațiile
* Framework-ul de aplicații
* Runtime-ul Android
* Bibliotecile platformei
* Kernel-ul Linux

### Kernel Linux

Kernel-ul Linux este inima arhitecturii Android. El gestionează driverele disponibile precum driverul pentru ecran, cameră, bluetooth, audio și memorie, toate necesare la runtime.

Kernel-ul Linux furnizează un strat de abstractizare între dispozitivele fizice și alte componente ale arhitecturii Android. El este responsabil pentru gestionarea resurselor de memorie, energie, dispozitive etc.

Funcțiile kernel-ului Linux sunt:

* Securitatea: Kernel-ul linux gestionează aspectele de securitate dintre aplicații și sistem.
* Gestionarea memoriei: gestionează memoria eficient pentru a oferi libertate în dezvoltarea aplicațiilor.
* Gestionarea proceselor: alocă resurse proceselor când acestea au nevoie de ele.
* Network Stack: gestionează eficient comunicarea în rețea.
* Modelul driverelor: Asigură că aplicațiile funcționează corect pe dispozitiv și că producătorii de hardware sunt responsabili pentru crearea driverelor și adăugarea lor în kernel-ul Linux.

<https://www.geeksforgeeks.org/android-architecture/>

### Strat de abstractizare hardware

Dezvoltarea driverelor pentru dispozitive Android este similară cu dezvoltarea unui driver pentru dispozitiv pentru Linux. Android folosește o versiune a kernel-ui Linux cu câteva adăugiri speciale ca Low Memory Killer - un sistem de gestionare a memoriei mult mai agresiv în conservarea memoriei, wake locks - un serviciu de sistem pentru gestionarea consumului de energie, driverul IPC Binder și alte funcții importante pentru o platformă mobilă încorporată. Aceste adaugiri sunt folosite în general pentru îmbunătățirea funcționalității sistemului și nu afectează dezvoltarea driverelor. Orice versiune a kernel-ului poate fi folosită cat timp suporta caracteristicile necesare, deși este recomandat să folosim cea mai nouă versiune a kernel-ului Android.

https://source.android.com/devices/architecture

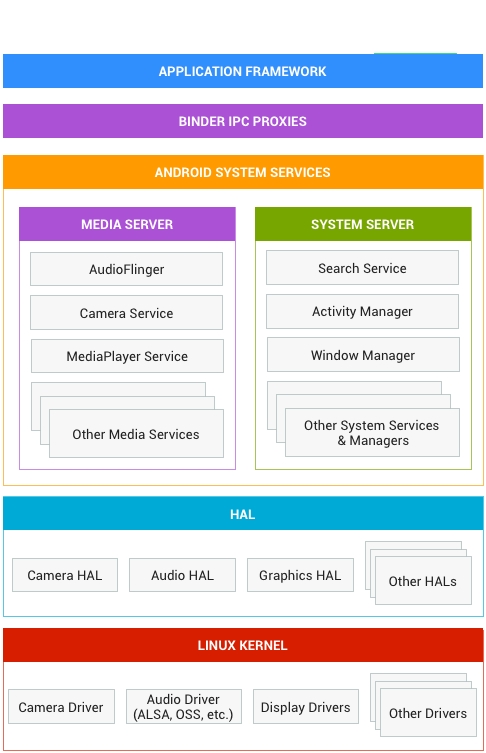


Figura 3.1.2 – Arhitectura sistemului Android

Stratul de abstractizare hardware HAL (Hardware abstraction layer) definește o interfață standard pe care producătorii de hardware să o poată implementa, care permite sistemului Android să nu aibă implementări specifice pentru drivere de nivel inferior. Folosirea HAL permite implementarea funcționalității fără afectarea sau modificarea nivelului de sistem înalt.

Implementările Hal sunt împachetate în module și încărcate de sistemul Android la momentul potrivit.

Limbajul definirii interfeței HAL - HIDL(HAL interface definition language) descrie interfața dintre o HAL și utilizatorii ei, permițând framework-ului Android să fie inlocuit fără refacerea HAL-urilor. Astfel de la restructurarea framework-ului sistemului de operare android de la versiunea Android 8.0 a devenit mai rapid, mai ușor și mult mai puțin costisitor pentru producători să actualizeze dispozitivele la o nouă versiune de Android. Pentru acestea HIDL separă implementarea de nivel inferior specifică fiecărui dispozitiv, scrisă de producătorii de siliciu, a furnizorilor de sistemul de operare Android prin o nouă interfață furnizor. Furnizorii și producătorii de cipuri construiesc HAL-uri o singură dată și le plasează în partiția „/vendor” de pe dispozitiv. Framework-ul la randul lui în propria partiție poate fi apoi înlocuit prin actualizare OTA(over-the-air).

Diferența dintre vechea arhitectura Android și cea curentă, bazată pe HIDL, constă în utilizarea acestei interfețe. În versiunile Android 7.x sau mai vechi, interfața neexistand, producătorii dispozitivelor trebuia sa inlocuiasca porțiuni extinse din codul Android pentru a își putea tranziționa dispozitivele la o versiune mai nouă de Android. Începând cu versiunea 8.0 o interfață nouă de furnizor permită accesul la părțile din Android specifice hardware astfel ca producătorii dispozitivelor să poate oferi versiuni mai noi ale framework-ului sistemului de operare fără să necesite muncă suplimentară din partea producătorilor de silicon.

Pentru a asigura compatibilitatea cu implementările furnizorilor, interfața furnizorului este validată de suita de testare a vendorilor VTS(Vendor Test Suite) analogă CTS(Compatibility Test Suite). VTS poate fi folosit pentru automatizarea testării HAL și a kernel-ului sistemului de operare în arhitecturile Android precedente și curente.

<https://source.android.com/devices/architecture>

### Bibliotecile platformei Android

Bibliotecile platformei includ diverse biblioteci C/C++, biblioteci bazate pe Java ca Media, Graphics, Surface Manager, OpenGL etc. pentru a oferi suport pentru dezvoltarea pe Android.

* Biblioteca Media oferă suport pentru redarea și înregistrarea diverselor formate video și audio.
* Surface manager este responsabil pentru managementul accesului către subsistemul de afișare.
* SGL și OpenGL sunt ambele API-uri(application programming interface) multiplatformă și multilimbaj folosite pentru grafică computerizată 2D și 3D.
* SQLite oferă suport pentru baze de date.
* FreeType oferă suport pentru fonturi.
* Web-Kit este un motor de browser care furnizează toată funcționalitatea pentru afișarea conținutului web și pentru simplificarea încărcării paginilor.
* SSL (Secure Sockets Layer) este o tehnologie pentru securitate care este folosită pentru realizarea unei legături criptate între un server web și un browser web.

Bibliotecile native C, C++

Multe componente de nucleu de sistem Android și servicii ca ART și HAL sunt construite din cod nativ care necesită biblioteci native scrise în C și C++ . Platforma Android furnizează API-uri din framework Java pentru a expune funcționalitatea a câtorva din aceste biblioteci native către aplicații. De exemplu se poate accesa biblioteca OpenGL ES prin API-ul OpenGL Java al framework-ului Android pentru a adăuga suport pentru desenarea și manipularea graficii 2D și 3D în aplicații. Pentru dezvoltarea unei aplicații care necesită cod nativ C sau C++ se poate folosi Android NDK(Native Development Kit) pentru a accesa câteva din aceste biblioteci native ale platformei direct din codul nativ al aplicației.

### Runtime Android

Mediul de rulare Android Runtime este una din cele mai importante părți ale Android deoarece conține componente importante a bibliotecii nucleu din mașina virtuală Dalvik. În principal el furnizează baza pentru framework-ul de aplicații și rulează aplicația cu ajutorul bibliotecilor de bază.

Runtime-ul oferă o implementare a bibliotecii standard java și un mecanism pentru transformarea bytecode-ului dex, în care aplicațiile sunt instalate pe dispozitiv, și convertirea lui în cod mașină care rulează direct pe procesor. De la Android Nougat implementarea standard a fost schimbată din Apache Harmony în openJDK, motiv pentru care programatorii Android au avut acces la noi caracteristici ale limbajului ca API-ul streams, lambda, referințe ale metodelor.

Runtime-ul Android și mașina virtuală Dalvik rulează pe bytecode Dalvik având diferențe fundamentale care afectează performanța și stocarea în mod special.

#### Mașina virtuală Dalvik

Cel mai vechi dintre cele două runtime-uri, mașina virtuală Dalvik este similară cu mașina virtuală standard java dar cu câteva diferențe arhitecturale care o fac mai potrivită pentru rularea cu instanțe multiple pe un dispozitiv cu resurse limitate. Ea folosește un compilator just-in-time pentru a transforma bytecode dex în cod mașină în timp ce este rulat. Acesta poate părea a fi un moment mai puțin optim pentru a compila bytecode în cod mașină, dar compilatorul just-in-time are avantajul de a folosi caracteristici specifice runtime-ului pentru a obține optimizări la utilizările ulterioare. Aceasta este denumită compilare just-in-time ghidată de profil și este motivul pentru care ART conține și un compilator just-in-time în ciuda faptului că face majoritatea compilării anterior.

#### Android Runtime(ART)

Runtime-ul Android ART este disponibil începând cu versiunea KitKat a sistemului de operare și a rămas singurul runtime disponibil începând cu versiunea Android Lollipop, când s-a renunțat la Dalvik. Unul din motivele pentru stocarea aplicațiilor în bytecode este faptul că bytecode-ul este mai expresiv, așadar ocupă mai puțin spațiu pe disk spre deosebire de codul mașină. Dispozitivele mobile nu mai sunt la fel de limitate de spațiu de stocare așa că stocarea codului mașina este acceptata. Astfel este îmbunătățit timpul de pornire al aplicațiilor și eliminat timpul de compilare la rulare.

ART compilează bytecode dex în cod mașină sub forma unui format executabil și linkabil ELF fișiere obiect partajat. De la Android Nougat, ART conține un compilator JIT cu scopul de a optimiza performanța la runtime spre deosebire de profilarea menționată anterior.

<https://theiconic.tech/android-java-fdbd55aadc51>

### Framework aplicații

Framework-ul de aplicații furnizează mai multe clase importante care sunt folosite pentru crearea unei aplicații Android. Astfel furnizează o abstractizare generică a accesului hardware și de asemenea ajută în managementul interfeței utilizatorilor împreuna cu resursele aplicației. În general el furnizează serviciile cu ajutorul căruia putem crea o clasă specifică și face acea clasă funcțională pentru crearea aplicației.

El este inclus de diverse tipuri de servicii pentru managementul activităților, managementul notificărilor în sistem de view-uri, managementul pachetelor, etc. care sunt de ajutor în dezvoltarea aplicației noastre în concordanță cu cerințele clientului.

Framework-ul Java API

Întregul set de caracteristici al sistemului de operare Android este disponibil prin API-uri scrise în limbajul Java. Aceste API-uri formează bazele de care este nevoie pentru a crea aplicația Android, simplificând reutilizarea componentelor modulare de sistem și a serviciilor de bază care includ următoarele:

Sistem de View-uri bogat și extensibil care poate fi folosit pentru a construi interfața cu utilizatorul unei aplicații incluzând liste, grile, cutii de text, butoane și un browser web incorporabil

Manager de resurse care furnizează access resurselor non cod ca variabile string localizate, grafică și fișiere de amplasare (layout).

Un manager de notificări care permite tuturor aplicațiilor să afișeze alerte personalizate în bara de status.

Manager de activități care gestionează ciclul de viață al aplicațiilor și furnizează o stivă de navigare comună întregului sistem.

Furnizor de conținut care permite aplicațiilor să acceseze date din alte aplicații, precum aplicația de contacte, și să partajeze propriile date.

Dezvoltatorii au acces deplin la aceleași API-uri de framework la care au acces și aplicațiile de sistem Android.

### Aplicații

Aplicațiile sunt stratul cel mai de sus al arhitecturii Android. Aplicațiile preinstalate precum home, contact, camera, gallery dar și aplicațiile descărcate din Play Store ca aplicații de chat, jocuri și altele vor fi instalate numai pe acest strat. Acestea rulează în Runtime-ul Android cu ajutorul claselor și serviciilor furnizate de framework-ul de aplicații.

#### Aplicațiile de sistem

Android vine cu aplicații preinstalate pentru email, mesagerie SMS, calendare, internet browsing, contacte și altele. Aplicațiile incluse cu platforma nu au statut special față de alte aplicații pe care utilizatorul alege să le instaleze, așadar o aplicație terță(third party) poate deveni browserul web implicit al utilizatorului, aplicația de mesagerie sms implicită sau chiar și tastatura implicită cu câteva excepții, de exemplu aplicația de setări de sistem.

Aplicațiile de sistem funcționează și ca aplicații pentru utilizatori și să furnizeze capabilități cheie care dezvoltatorii le pot accesa din aplicația lor. De exemplu dacă aplicația noastră ar vrea să trimită un mesaj SMS nu este nevoie să construim această funcționalitate în aplicație, putând invoca aplicația implicită de SMS care este instalată în dispozitiv pentru a trimite un mesaj destinatarului specificat.

<https://developer.android.com/guide/platform#native-libs>

#### Componentele unei aplicații Android

Toate aplicațiile Android vor fi construite din cele patru componente de bază care sunt definite de Arhitectura Android :

Activitățile sunt comparabile utilităților de sine stătătoare pe sistemele desktop, de exemplu aplicațiile office. Activitățile sunt bucăți de cod executabil care sunt pornite și oprite, instantiate fie de utilizator fie de sistemul de operare și rulează cât timp sunt necesare. Ele interacționează cu utilizatorul și cer date sau servicii de la alte activități sau servicii prin query sau intent. Majoritatea codului executabil scris pentru Android va fi executat în contextul unei activități. Activitățile de obicei corespund ecranelor de afișare, așadar fiecare activitate arată un ecran utilizatorului. Când o activitate nu rulează activ, ea poate fi oprită de sistemul de operare pentru a economisi memorie.

Serviciile sunt analogice serviciilor sau daemonilor din sistemele de operare desktop sau de server și sunt bucăți executabile de cod care de obicei rulează în fundal de la timpul instalării lor până când dispozitivul mobil este închis. În general ele nu expun o interfață vizibilă utilizatorului.

Un exemplu clasic de un serviciu Android este un player mp3 care trebuie să continuie redarea fișierelor din listă chiar dacă utilizatorul a început să folosească alte aplicații. Aplicația noastră va necesita implementarea serviciilor pentru a rula sarcini de fundal care să persiste fără o interfață cu utilizatorul activă.

#### Receptoare de difuzare și de intenție(Broadcast and Intent Receivers)

Există receptoare care răspund cererilor pentru servicii făcute de alte aplicații,  un receptor răspunde anunțului la nivel de sistem al unui eveniment. Aceste anunțuri vin de la sistemul de operare în sine, de exemplu “Baterie descărcată”, sau de la orice aplicație care rulează pe sistem.

O aplicație sau un serviciu oferă altor aplicații acces la funcționalitatea sa executând un receptor de intenție: o bucată mică de cod executabil care răspunde cerințelor pentru date sau servicii de la alte activități. Activitatea client, pentru a cere acces la un serviciu, trimite o intenție lăsând la latitudinea framework-ului Android decizia de la care aplicație ar trebui să primească răspuns și să acționeze asupra ei.

Intențiile sunt una din elementele arhitecturale cheie din Android care facilitează crearea noilor aplicații din aplicații preexistente (mobile mashup). Putem folosi intenții în aplicații pentru a interacționa cu alte aplicații și servicii care oferă informații necesare aplicației noastre.

Furnizorii de conținut sunt creați pentru a partaja datele cu alte activități sau servicii. Un furnizor de conținut folosește o interfață standard în forma unui URI Pentru a îndeplini și cererile de date de la alte aplicații care pot să nici nu știe care furnizor de conținut este folosit. De exemplu când o aplicație emite cerere pentru date de contact, aceasta adresează cererea în URI de forma:

content://contacts/people

Sistemul de operare verifică apoi care aplicații s-au înregistrat ca furnizor de conținut pentru URI-ul furnizat și trimite cererea aplicației potrivite, pornind aplicația dacă aceasta nu rulează deja. Dacă sunt mai mult de un furnizor de conținut înregistrați pentru URI-ul cerut sistemul de operare întreabă utilizatorul pe care dintre ele vrea să îl folosească.

Aplicația nu este obligată să folosească toate componentele Android dar o aplicație bine scrisă va utiliza toate mecanismele furnizate în loc să reinventeze funcționalitate sau să hard-codeze referințe către alte aplicații. URI-urile și intențiile împreună permit sistemului Android să furnizeze un mediu foarte flexibil pentru utilizator. Aplicațiile pot fi de asemenea cu ușurință adăugate, șterse sau înlocuite iar cuplarea slabă a intențiilor cu URI-urile face ca totul să funcționeze împreună.

#### Ciclul de viață al activităților Android

Android este proiectat în jurul cerințelor unice ale aplicațiilor mobile. În special Android recunoaște faptul că resursele sunt limitate și furnizează mecanisme pentru a economisi acele resurse. Mecanismele sunt cel mai evidente în ciclul de viață al activităților Android, care definește stările sau evenimentele prin care o activitate va trece de la timpul în care a fost creată până când termină rularea.

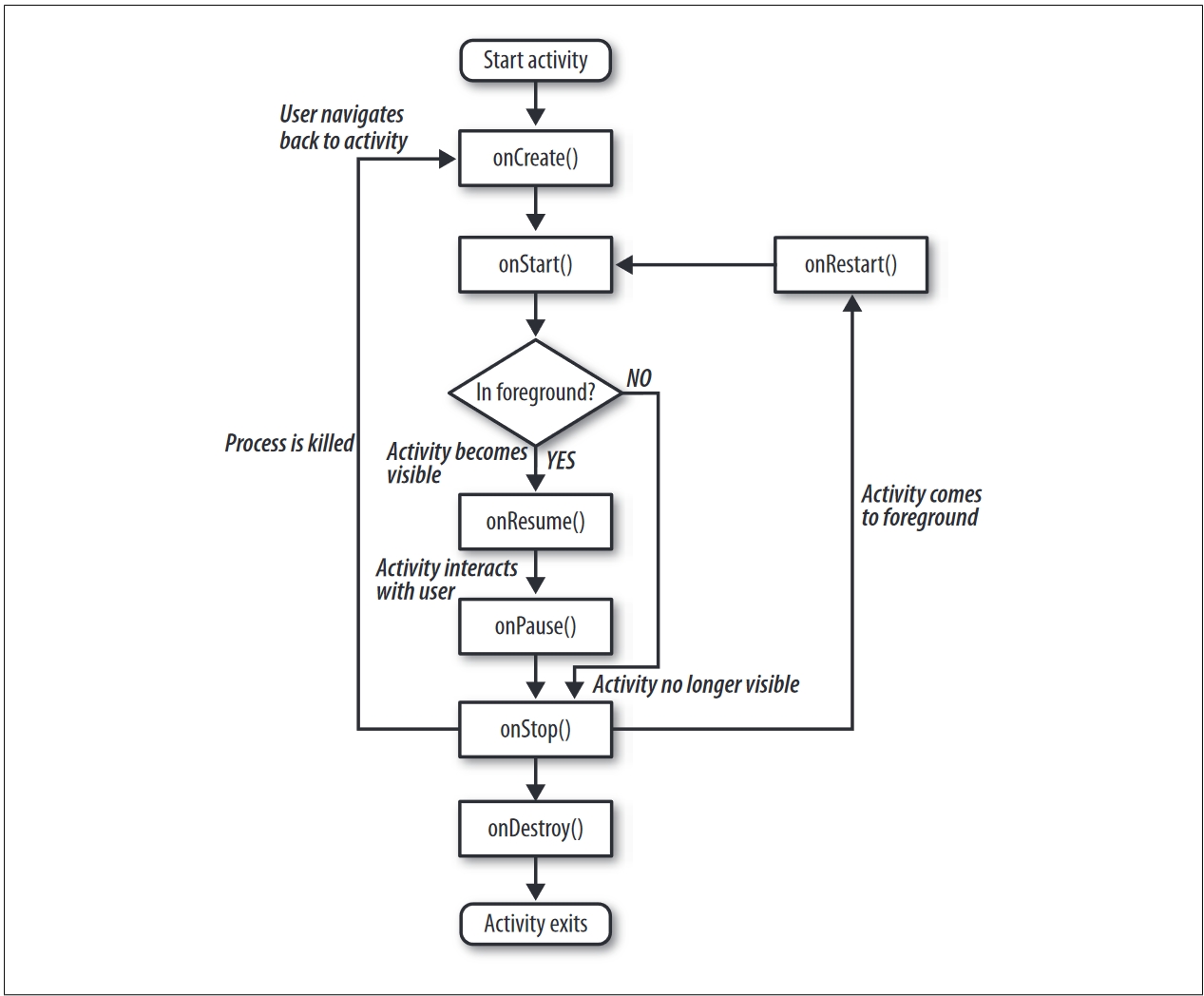


Figura 3.1.6.4 - Ciclul de viață al unei activitați Android

OReilly Android Application Development (2009) Rick Rogers, John Lombardo, Zigurd Mednieks & Blake Meike

Activitatea monitorizează și reacționează la aceste evenimente instanțând metode care suprascriu metode ale clasei activitate pentru fiecare eveniment.

* onCreate Este apelat atunci când activitatea este creată pentru prima dată. Acesta este locul în care de obicei sunt create View-urile, fișierele de date persistente de care are nevoie aplicația, sunt deschise și în general activitatea este inițializată. Când apelăm onCreate framework-ul Android primește un obiect Bundle care conține orice stare de activitate salvată din momentul în care activitatea a rulat anterior.
* onStart este apelat chiar înainte ca activitatea să devină vizibilă pe ecran. Dacă activitatea poate deveni activitatea din prim plan pe ecran, odată completat onStart controlul va fi transferat către onResume. Dacă activitatea nu poate deveni activitatea din prim plan din orice motiv controlul este transferat către metoda onStop.
* onResume este apelat imediat după onStart dacă activitatea este cea din prim plan. În acest punct activitatea rulează și interacționează cu utilizatorul, primește intrări de la tastatură sau atingeri de la ecran, iar ecranul afișează interfața cu utilizatorul. onResume este de asemenea apelat dacă activitatea pierde prim planul în defavoarea altor activități și acea activitate eventual se închide readucând activitatea inițială înapoi în prim plan. În acest moment aplicația ar reîncepe lucrurile necesare pentru reîmprospătarea interfeței cu utilizatorul și pentru a primi actualizări de locație sau pentru a rula o animație.
* onPause este apelat când Android este pe cale să repornească o activitate diferită dând acelei activități prim planul. În acest punct activitatea noastră nu va mai avea acces la ecran așa că va trebui să oprim procesele care consumă baterie și cicluri de procesor inutil. Dacă rulăm o animație nimeni nu o va putea vedea așa că o vom putea suspenda până când primim acces din nou la ecran. Aplicația noastră va trebuie din nou să utilizeze această metodă pentru a stoca orice stare de care vom avea nevoie în caz că activitatea noastră primește prim planul din nou, deoarece nu este garantat că activitatea noastră va fi reapelată. Dacă dispozitivul mobil rămâne fără memorie nu există memorie virtuală pe disc care să o putem folosi pentru expansiune așa că activitatea va trebui să facă loc pentru procese de sistem care necesită memorie. Odată ieșit din această metodă Android ne poate opri activitatea în orice moment fără returnarea controlului către activitate.
* onStop este apelat atunci când activitatea nu mai este vizibilă fie pentru că o altă activitate a luat prim planul fie pentru că activitatea este în proces de a fi distrusă.
* onDestroy este ultima șansă pentru ca activitatea noastră să ruleze procese înainte de a fi distrusă. În mod normal s-a ajuns în acest punct pentru că activitatea este terminată și framework-ul a chemat metoda ei finish(). Dar metoda poate fi apelata deoarece Android a decis că are nevoie de resurse care momentan sunt folosite de activitatea noastră.

Este important să utilizăm aceste metode pentru a furniza utilizatorului cea mai bună experiență posibilă.

#### Ciclul de viață al serviciilor Android

Serviciile pot fi pornite când un client apelează metoda Context.startService(Intent). Dacă serviciul nu rulează deja Androidul pornește pe lângă el metoda onCreate urmată de metoda onStart. Dacă serviciul deja rulează metoda lui onStart este invocată din nou cu noua intenție. E posibil și chiar normal ca metoda onStart a unui serviciu să fie apelată în mod repetat într-o singură rulare a serviciului.

onResume, onStart și onStop nu sunt necesare deoarece un serviciu nu are o interfață cu utilizatorul în general așadar nu este nevoie de aceste metode. Dacă un serviciu rulează el este întotdeauna în fundal.

onBind este apelat când un client necesită o conexiune persistentă la un serviciu prin metoda Context.bind Service. Aceasta creează serviciul dacă nu rulează deja și apelează onCreate dar nu și onStart. În schimb metoda onBind este apelată în intenția clientului și returnează un obiect IBind pe care clientul îl poate folosi să facă apelari ulterioare către serviciu. Este un lucru obișnuit pentru un serviciu să fie apelat de clienți care îl pornesc și clienți legați de el în același timp.

onDestroy este metoda apelată când serviciul este pe cale să fie închis. Android va închide un serviciu când nu mai sunt clienți care îl pornesc sau legați de el. La fel ca la activități, Android poate să închidă un serviciu când memoria este pe cale de a se umple. Dacă acest lucru se întâmplă Android va încerca să repornească serviciul când memoria este eliberată așa că dacă serviciul are nevoie să stocheze informație persistentă pentru această repornire este mai bine să o facem în metoda onStart.

OReilly Android Application Development (2009) Rick Rogers, John Lombardo, Zigurd Mednieks & Blake Meike

## Java in contextul Android

Faptul că aplicațiile Android sunt programate în Java este destul de cunoscut. Acest lucru este de multe ori, în mare, incorrect interpretat. Aplicațiile Android rulează pe o mașină virtuală Java Standard JVM(Java Virtual Machine) folosind bytecode Java.

Pentru a afla cât de mult se bazează Android cu adevărat pe Java va trebui să definim clar la ce ne referim când spunem Java. De obicei prin aceasta ne referim la limbajul de programare, în cazul tipic mediul de rulare este incorect presupus a fi mașina virtuală Java. Pentru astfel de referire Java este de fapt doar o specificație a limbajului de programare. Orice compilator care are un front end pentru specificație poate lua acel cod sursă și va produce o aplicație suportată care nu trebuie să fie bytecode JVM. Acesta va fi doar una din opțiuni. Drept exemplu există un compilator lansat ca parte din colecția de compilatoare GNU denumite GCC-GCJ. Acesta poate compila cod sursă Java direct în cod mașină așa dar nu necesită și o mașină virtuală pentru a executa. Din nefericire s-a renunțat la acest proiect între timp.

Sistemul de Build Android

După ce aplicațiile sunt scrise în Java, codul sursă este compilat și împachetat într-un APK (Android Application Package) împreună cu toate resursele, obiectele și dependențele lor.

Compilatorul a fost implementat în două metode. Metoda inițială a fost lansată ca parte a SDK-ului Android. În această metodă codul sursă Java este luat și convertit în bytecode Java (fișiere .class) prin compilatorul Java javac ce include sursele generate din procesorul de adnotări Java (dagger sau lombok). Transformările de bytecode au apoi oportunitatea de a manipula bytecode Java pentru a face lucruri ca ofuscarea și minimizarea, înainte să transforme codul în bytecode dex folosind compilatorul dx. Acesta este apoi împachetat în APK folosind diverse alte resurse. Separarea celor două compilatoare implică faptul că tot ce are compatibilitate cu Java poate fi folosit transparent cu dx. Acest lucru înseamnă că procesul de Build durează mai mult decât dacă ar fi fost compilat direct în bytecode dex.

A doua metodă denumită JACK și JILL încearcă să simplifice acest proces furnizând un compilator care poate compila din Codul sursă Java direct în bytecode dex peste compilator. Acest compilator este numit Java Android Compiler Kit, sau JACK pe scurt. Jack nu mai suportă bytecode Java deși multe biblioteci Android sunt distribuite în formă arhivelor Java (jar) sau arhivelor Android (aar) care conțin bytecode Java. Lanțul de unelte încă trebuie să îl suporte. Acest suport este furnizat de JILL (Jack intermediate library linker) care preia codul Java și fișierele asociate și le transformă într-un nou format intermediar numit JAYCE. Aceasta înseamnă că fișierul .jack conține bytecode dex și nu mai trebuie să fie compilat din nou. Suportul pentru bytecode Java sau alte limbaje compatibile JVM (Kotlin, Scala) este încapsulat în JILL.

## SQLite

SQLite este o bibliotecă în proces care implementează un motor de baze de date SQL tranzacțional autonom, fără server, cu configurare zero. Codul pentru SQLite se află în domeniul public și, prin urmare, este gratuit pentru utilizare în orice scop, comercial sau privat. SQLite este cea mai răspândită bază de date din lume cu mai multe aplicații decât putem număra, inclusiv mai multe proiecte de profil înalt. SQLite este un motor de baze de date SQL încorporat. Spre deosebire de majoritatea celorlalte baze de date SQL, SQLite nu are un process server separat.

Fiind bază de date SQL completă cu mai multe tabele, indici, declanșatoare și vizualizări, SQLite citește și scrie direct pe disc fișiere obișnuite. Formatul fișierului bazei de date este multiplatformă - puteți copia în mod liber o bază de date între sistemele pe 32 de biți și pe 64 de biți sau între arhitecturi big-endian și little-endian. Aceste caracteristici fac din SQLite cea mai populară alegere. Fișierele bazei de date SQLite sunt un format de stocare recomandat de Biblioteca Congresului din SUA. SQLite nu este un înlocuitor pentru Oracle, ci un înlocuitor pentru fopen().

SQLite este o bibliotecă compactă. Cu toate caracteristicile active, dimensiunea bibliotecii poate fi mai mică de 600 KB, în funcție de platforma țintă și setările de optimizare a compilatorului, cum ar fi integrarea agresivă a funcțiilor și derularea buclelor, pot face codul obiectului mai mare. Există un compromis între utilizarea memoriei și viteză. SQLite rulează în general mai repede cu cât îi oferiți mai multă memorie. Cu toate acestea, performanța este de obicei destul de bună chiar și în mediile cu memorie redusă. În funcție de modul în care este utilizat, SQLite poate fi mai rapid decât procedura directă I/O a sistemului de fișiere.

Majoritatea codului sursă SQLite este dedicat doar testării și verificării. O suită de teste automate rulează milioane și milioane de cazuri de testare care implică sute de milioane de instrucțiuni SQL individuale și realizează 100% acoperire de testare a ramurilor. SQLite răspunde cu grație la eșecurile de alocare a memoriei și la erorile I/O ale discului. Tranzacțiile sunt ACID(atomice, consistente, izolate și durabile) chiar dacă sunt întrerupte de blocări ale sistemului sau de întreruperi de alimentare. Toate acestea sunt verificate de testele automate folosind hamuri de testare speciale care simulează defecțiunile sistemului. Desigur, chiar și cu toate aceste teste, există încă bug-uri. Dar SQLite este deschis cu privire la toate erorile și oferă liste de erori și cronologii ale modificărilor de cod.

<https://www.sqlite.org/about.html>

## RecyclerView si ViewPager2

## Fragmente

# Capitolul II - Etapele de dezvoltare a unei aplicații Android

Cu peste 1.5 milioane de aplicații în Google Play Store și Apple App Store este important să știm procesul dezvoltării unei aplicații mobile și cum aceasta se va potrivi în piață și cu obiectivele de marketing. Ciclul de viață al dezvoltării aplicațiilor mobile este o reprezentare a ciclului de viață de dezvoltare software convențional din perspectiva unui dispozitiv mobil.

În ziua de azi realizarea unei aplicații mobile nu este foarte complicată deși realizarea unei aplicații mobile de succes necesită planificare în prealabil considerabilă. Construirea propriei aplicații android se poate rezuma la deschiderea IDE-ului, incropirea câtorva lucruri, o testare rapidă și publicarea în App Store, totul terminat într-o singură zi. De asemenea realizarea aplicației poate fi un proces foarte intensiv, implicand proiectare prealabilă riguroasă, testare pentru asigurarea calității pe un set extins de dispozitive, testarea utilizabilității, un ciclu beta complet și publicarea aplicației în mai multe moduri.

## Cercetare

Toate aplicațiile încep doar cu o idee, chiar dacă ideea este de a avea o aplicație mobilă vor trebui noțiuni de bază solide pentru a crea o aplicație. Analiza inițială trebuie să includă demografica, motivația, modelele de comportament și scopurile cumpărătorului. În timpul fiecărei etape ale procesului utilizatorul final trebuie considerat. Ne gândim la ciclul de viață al consumatorului, al cărui caracteristici sunt definite. După ce atragem atenția consumatorilor, ei trebuie să fie captivați, convertiți, reținuți și fidelizați. În sfârșit va trebui să înțelegem că clientul va folosi produsul digital. Dacă faceți acest lucru chiar de la început, vă veți stabili o bază fermă, iar claritatea obtinuta vă va oferi dumneavoastră și investitorilor dumneavoastră încrederea atât de necesară.

Faza aceasta este necesara deoarece în timpul ei fundația muncii care va urma este stabilită. Înainte de a avansa la următoarea etapă este necesară o cantitate considerabilă de cercetare și brainstorming. O altă parte esențială a acestei faze este analizarea competiției. Un studiu detaliat al aplicațiilor competiției va oferi o idee foarte bună despre caracteristicile care lipsesc din aplicația lor și ce putem face pentru a le include în a noastră pentru a o face remarcabilă.

## Schițare

Următorul pas este documentarea și schițarea aplicației pentru a înțelege funcționalitățile viitoare. Deși timpul poate să nu fie de partea noastră în acest moment, desenarea schițelor detaliate ale produsului final imaginat ajută la descoperirea problemelor de utilizabilitate. Schițarea înseamnă mult mai mult decât urmărirea pașilor și poate fi o unealtă valoroasă pentru comunicare și colaborare. După schițarea inițială, ideile se pot rafina iar componentele design-ului pot fi rearanjate în moduri optime. În etapa aceasta inițială se pot depăși orice limitari tehnice descoperite în procesul dezvoltării backend-ului. Următorul pas este crearea unei înțelegeri funcționale a felului în care toate caracteristicile aplicației vor colabora, întrunind o aplicație funcțională. În această etapă se poate de asemenea crea o harta a aplicației pentru a specifica relațiile dintre fiecare fereastră și modul în care utilizatorii vor naviga prin aplicație. Putem exploata toate oportunitățile de a ne incorpora marca, ne putem concentra pe experiența utilizatorului și putem lua în considerare diferențele din felurile în care utilizatori diferiți ne vor folosi aplicația în moduri diferite.

## Studiu tehnic de fezabilitate

Putem avea o înțelegere fundamentală a elementelor vizuale ale aplicației dar trebuie de asemenea să luăm în considerare dacă sistemele backend vor fi capabile să suporte funcționalitățile aplicației noastre. Pentru a afla dacă ideea noastră de aplicație este fezabilă tehnic va trebui să accesăm date publice din API-uri publice. În funcție de format(telefon, tabletă, accesorii, TV, etc.) și de platformă (iOS, Android, etc.) o aplicație va avea cerințe diferite. La sfârșitul acestei etape echipa poate avea idei diferite pentru aplicație ori poate să fi decis că unele din funcționalitățile inițiale nu sunt fezabile, apoi ne vom putea pune întrebări referitoare la aplicație și putem reanaliza starea aplicației.

## Prototipare

Este important să construim un prototip rapid, cu accent pe rapid. Nu putem înțelege experiența oferită de aplicație până nu putem interacționa fizic cu ea și simți cum funcționează. Așadar va trebui să realizăm un prototip care transpune conceptul aplicației într-o variantă fizică în mâinile utilizatorului cât mai rapid posibil pentru a putea vedea cum funcționează în cazurile de utilizare cele mai comune. Putem folosi modele mai puțin riguroase pentru această fază deoarece scopul ei este de a ne ajuta să înțelegem dacă dezvoltarea aplicației continuă în direcția corectă. Va trebui să includem și investitorii în acest proces pentru a le permite să interacționeze cu prototipul și să putem implementa sugestiile lor. Mai mult, prototipul va da fiecărui investitor o primă interacțiune cu aplicația și va valida sau invalida informațiile adunate până acum.

## Proiectare

Imediat după ce acest pas este terminat se poate începe dezvoltarea aplicației. Designerul experienței utilizatorului UX arhitectează interacțiunea dintre elementele de design, în timp ce designerul de interfață de utilizator (UI) construiește aspectul aplicației. Acesta este un proces în mai multe etape în urma căruia se creează planuri și o direcție vizuală care vor informa inginerii software despre cum va trebui să arate și să funcționeze aplicația. În funcție de scopul proiectului și bugetul aplicației, această fază de design poate fi completată între o singură după amiază sau poate dura mai multe zile cu implicarea întregii echipe. Ar trebui create mai multe variante ale fiecărui ecran mutând elementele de navigare, butoanele sau alte elemente vizuale. Cu cât produsul variază mai mult cu atât este mai probabil ca experiența utilizatorului să fie originală. Rezultatul proiectării aplicației ar trebui să fie o direcție vizuală clară care să furnizeze abstractizarea produsului final.

## Dezvoltare

Fază dezvoltării începe în general foarte devreme, de fapt odată ce ideea este cât de cât matură în faza de concepție, un prototip funcțional este dezvoltat care va fi folosit pentru a valida funcționalitatea și presupunerile echipei. Acesta ajută să formeze o imagine asupra volumului de muncă necesar.

În timp ce dezvoltarea progresează, aplicația trece printr-un set de etape.  În etapa inițială funcționalitatea de bază deși prezentă încă nu este testată, aplicația find încă plină de defecte software(Bugs). Toată funcționalitatea care nu ține de nucleu nu există în momentul acesta. În a doua etapă o mare parte din funcționalitatea propusă este încorporată în aplicație, aceasta trecând deja prin o fază ușoară de testare și reparare a bug-urilor, însă tot va avea câteva probleme. În această fază aplicația va fi trimisă câtorva utilizatori externi pentru mai multă testare. După ce aceste bug-uri sunt reparate, aplicația se va muta într-o fază a dezvoltării în care va fi pregătită pentru lansare.

Dacă aplicația este un proiect complex unde cerințele se schimbă des poate folosi metodologia agile. Aceasta va ajuta cu planificări flexibile, dezvoltarea progresivă, dezvoltarea inițială și îmbunătățiri constante. O aplicație mare poate fi împărțită în module mai mici iar metodologia agile poate fi aplicată asupra fiecărei bucăți.

## Testare

În domeniul dezvoltării aplicațiilor mobile este în general o idee bună să testăm începând devreme și foarte des. Acest lucru va păstra costurile finale scăzute deoarece într-un stadiu avansat al ciclului de dezvoltare poate devine mai scump să reparăm buguri.  Va trebui să folosim documentele inițiale de proiectare și planificare ca referință în timp ce construim diverse cazuri de testare.

Testare aplicației este vastă așa că va trebui să ne asigurăm că echipa acoperă toate fațetele necesare ale ei. Aplicația va trebui testată pentru utilizabilitate, compatibilitate, securitate, interfață, stres și performanță. Cu testarea de acceptanță a utilizatorului vom descoperi dacă aplicația mobilă funcționează pentru utilizatorii țintă. Pentru a realiza acest test va trebui să dăm  aplicația utilizatorilor în grupul țintă și să punem întrebări pertinente pentru a afla dacă soluția noastră funcționează pentru ei. După aceea va trebui să facem aplicația disponibilă în faza de Beta trial a grupurilor identificate anterior printr-o solicitare deschisă pentru participanți. Răspunsurile primite de la utilizatori în Beta ne vor ajuta să înțelegem dacă funcțiile aplicației sunt utilizabile în cazuri din lumea reală.

## Lansarea aplicatiei

Odată ce aplicație este gata de lansat va trebui să alegem o zi ca să ne pregătim pentru o lansare formală. Pentru magazine aplicații politicile lansării unei aplicații sunt diferite. Va trebui să ținem cont că acesta nu este sfârșitul: dezvoltarea aplicațiilor nu se sfârșește la lansare. Odată ce aplicație va ajunge în mâinile utilizatorilor finali, feedback-ul va ajunge în cantități mari și va trebui încorporat în versiunile viitoare ale aplicației. Orice aplicație va avea nevoie de noi funcții și actualizări. De obicei odată cu lansarea primei versiuni a aplicației ciclul de dezvoltare începe din nou. Va trebui să ne asigurăm că avem resursele pentru a întreține produsul. Pe lângă investiția financiară într-un produs digital trebuie să ținem cont că este și un angajament pe termen lung.

<https://medium.com/swlh/what-are-the-various-phases-of-mobile-app-development-4f0a1748e619>

# Capitolul III – Aplicații utilizate

## Git

## GitHub

## Android Studio

## Gradle

## Android emulator - qemu

## Trello

# Capitolul IV – Arhitectura aplicației

## Clasele model

## Structura bazei de date

## Autentificare – Înregistrare

## ViewPager2

## Fragmente cu RecyclerView

## Fragmente de editare

# Capitolul V – Funcționalitatea aplicației

## Activitatea de autentificare

## Activitatea de înregistrare

## Activitatea principala

## Navigarea

## Operații cu Task-uri

## Operații cu liste de Task-uri

# Testarea aplicației

# Concluzii

# Bibliografie

1. **Overflow, Stack.** *Android™ Notes for Professionals.* s.l. : Creative Commons BY-SA.

RecyclerView este un widget mai flexibil si mai avansat decat ListView și GridView. Este un container folosit pentru afișarea seturilor de date mari care pot fi derulate eficient prin menținerea unui număr limitat de view-uri. Putem folosi RecyclerView pentru colectii mari de date ale căror elemente se pot schimba în timpul rulării aplicației în funcție de evenimente din rețea sau de o acțiune a utilizatorului. Platforma Android folosește clasele View și ViewGroup pentru a desemna obiecte pe ecran. Aceste clase sunt abstracte și sunt extinse în diferite implementări pentru a se potrivi cazurilor de utilizare. EditText se extinde din aceeași clasă View și adaugă mai multă funcționalitate pentru a permite utilizatorului să introducă date.

RecyclerView este un ViewGroup care randeaza orice view bazat pe adaptoare în mod similar și este gândit ca fiind succesorul