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 Băscălescu

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.	Conținutul științific 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,

_____

1 Cuprins

2 Introducere

Viața din ziua de azi devine din ce în ce mai plină de mici evenimente care ne distrag atenția de la munca la care încercăm sa ne concentrăm, fie în scop personal sau hobby, fie în scop de serviciu. Aceste evenimente pot fi notificări din aplicațiile de social media sau de mesagerie, articole de presă noi apărute, anunțuri oficiale, email-uri sau discuții neprogramate între colegi la serviciu. Oricare din aceste întreruperi ne pot face sa pierdem șirul gândurilor și concentrarea asupra obiectului la care munceam.

Pentru atenuarea acestei probleme am decis realizarea unei aplicații Android care are ca obiectiv să ajute la urmărirea fluxului de munca pentru a ne putea aminti mai rapid ultima etapa abordată și pentru a putea reveni mai ușor unde am rămas. Aplicația va putea de asemenea ordona activitățile curente sau task-urile pe liste de categorii de tipul De făcut, În curs de realizare, Terminat. Aceste categorii pot ajuta la o ușoare prioritizare a sarcinilor de lucru și eventual împărțirea lor pe echipă.

Ideea pentru această aplicație a apărut în urma neajunsurilor în urmărirea și terminarea la timp a unor proiecte mai mici, câteodată de importanță mai scăzută dar nu întotdeauna, de la locul de munca. Aceste proiecte puteau reajunge în prim-plan în momente neașteptate, când ar fi fost nevoie rapid de rezultatul lor, dar acesta nu există sau când rezultatul lor ar fi putut preveni diverse probleme care încep să apară. Finalizarea lor în prealabil ar putea economisi timp și bani la apariția unei situații urgente care le necesita. Când un membru al echipei ar fi ocupat cu alte sarcini de muncă, astfel nereușind continuarea muncii la unul din aceste proiecte, alt coleg cu mai puține sarcini ar putea prelua munca pe proiect dacă ar avea o idee despre etapa în care a rămas înainte de ultima întrerupere.

Din posibilitatea coordonării pe echipă a proiectelor a apărut a doua idee din timpul schițării și proiectării acestei aplicații și anume obiectivul extins de a permite sincronizarea datelor din aplicația de pe dispozitivul mobil cu un server central la care ar avea acces toți membrii echipei pentru a putea reactualiza starea în care se află fiecare proiect. Fiecare utilizator s-ar fi autentificat cu acest server cu datele personale și ar fi primit drepturi de acces diferite pentru fiecare proiect la care participă, drepturile fiind alocate de liderul de echipă astfel încât fiecare să aibă acces doar unde are nevoie și nu la proiecte pentru care nu are autorizație de vizualizare sau editare în cazul etapelor proiectelor pentru care nu are pregătirea necesara de a le aborda. Dreptul de ștergere a proiectelor urmând a fi rezervat doar administratorilor și

folosit doar în cazurile în care proiectele au fost abandonate permanent cu prea puțină muncă efectuată în contul lor pentru a putea păstra un rezultat pertinent.

În dezvoltarea practică a aplicației acest obiectiv nu a fost atins momentan datorită lipsei resurselor de timp, a complexității aplicației și a programării anevoioase din pricina lipsei de experiență în programarea aplicațiilor de o asemenea anvergura în limbajul de programare java pe platforma Android. După realizarea backend-ului de pe serverul de centralizare și agregare a datelor proiectelor, a utilizatorilor și a drepturilor lor, realizarea modulului de sincronizare a bazei de date SQLite de pe dispozitiv cu cea de pe server și după finalizarea modulului de gestionare a utilizatorilor și a drepturilor lor, aplicația ar fi putut fi pregătită pentru a trece în faza de lansare putând fi publicată pe Google Play Store cu scopul monetizării.

Piața pentru aplicații mobile este una din cele mai mari cu aproape 5.3 miliarde de utilizatori la nivel global și cu numărul global de dispozitive mobile fiind 14 miliarde în 2020 și estimat că va ajunge până la 17 miliarde în 2024[1].

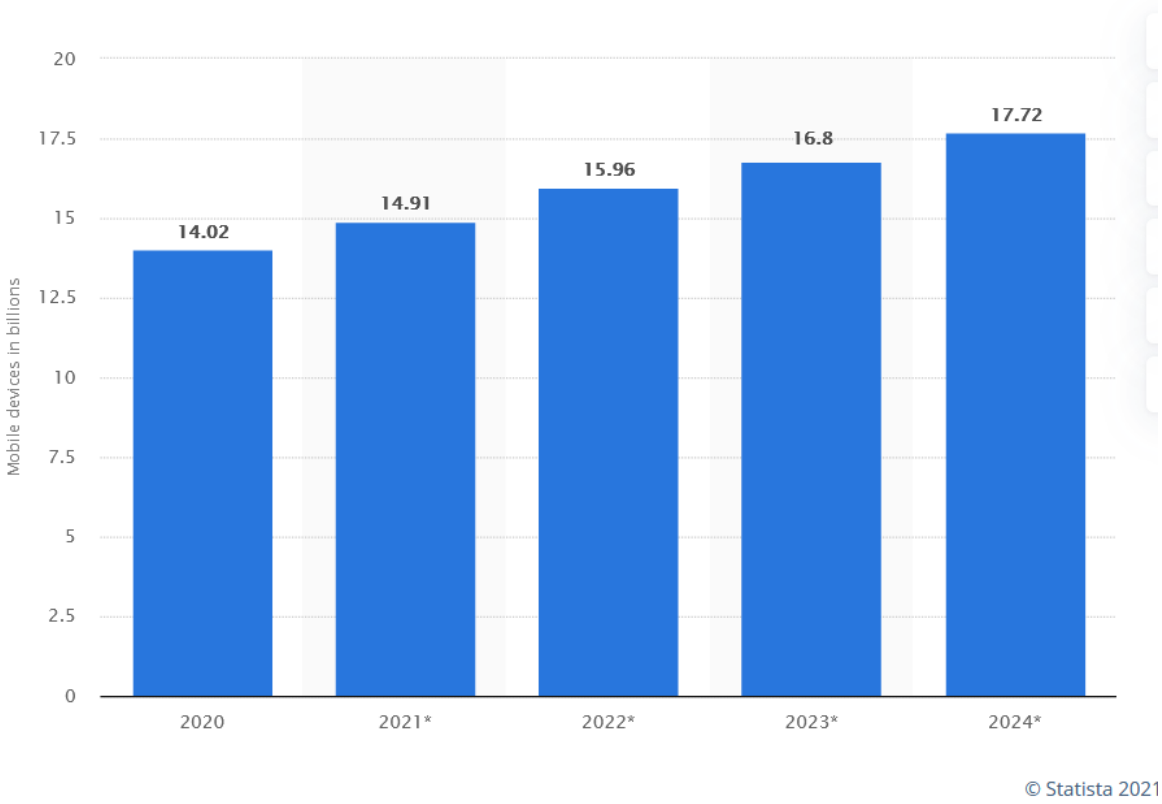


Figura 2.1 – Estimarea creșterii numărului de dispozitive mobile

[1] https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/ - accesat pe 14.06.2021

Aplicația realizată include în stagiul din momentul realizării documentului 14 clase Java care asigură funcționalitatea și 7 fișiere .xml care descriu modul în care aspectul grafic al aplicației va fi desenat.

Printre clase se regăsesc:

- LoginActivity.java - activitate care se ocupă cu autentificarea utilizatorului și trimiterea la activitatea principală sau la activitatea de înregistrare
- RegisterActivity.java - activitate care înregistrează utilizatorul în baza de date și trimite la activitatea principală
- MainActivity.java - activitatea principală care gestionează majoritatea elementelor aplicației
- SQLiteDatabaseHelper.java - clasa ajutătoare pentru citirea și scrierea în baza de date integrată a dispozitivului
- Role.java - clasa model pentru drepturile de acces ale utilizatorilor
- User.java - clasa model pentru utilizator și datele personale
- Project.java - clasa model pentru proiectele înscrise în aplicație și accesul asupra lor
- TaskList.java - clasa model pentru categoriile/listele de task-uri
- Task.java - clasa model pentru task sau o activitate individuală
- TaskAdapter.java - clasa pentru adaptarea datelor din task în RecyclerView
- TaskListAdapter.java - clasa pentru adaptarea fragmentelor cu listele de task-uri în ViewPager2
- TaskFragmentDetail.java - Fragment pentru editarea datelor unui task
- TaskListFragment.java - Fragment care conține un RecyclerView cu task-uri
- TaskListDetailFragment.java - Fragment pentru editarea datelor unei liste de task-uri

Fișierele pentru generarea aspectului corespund activității view-ului sau fragmentului pe care îl descriu.

- activity_login.xml - LoginActivity.java
- activity_main.xml - MainActivity.java
- activity_register.xml - RegisterActivity.java
- fragment_task_detail.xml - TaskFragmentDetail.java
- fragment_task_list.xml - TaskListFragment.java
- fragment_task_list_detail.xml - TaskListDetailFragment.java
- layout_tasklistitem.xml - TaskAdapter.java

Fiecare fișier va fi descris în detaliu la capitolele Capitolul IV - Arhitectura aplicației și Capitolul V – Funcționalitatea aplicației. Implementările din toate acestea sunt contribuția personală, la fel și aspectele descrise în fișierele .xml.

3 Capitolul I – Tehnologii utilizate

3.1 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)

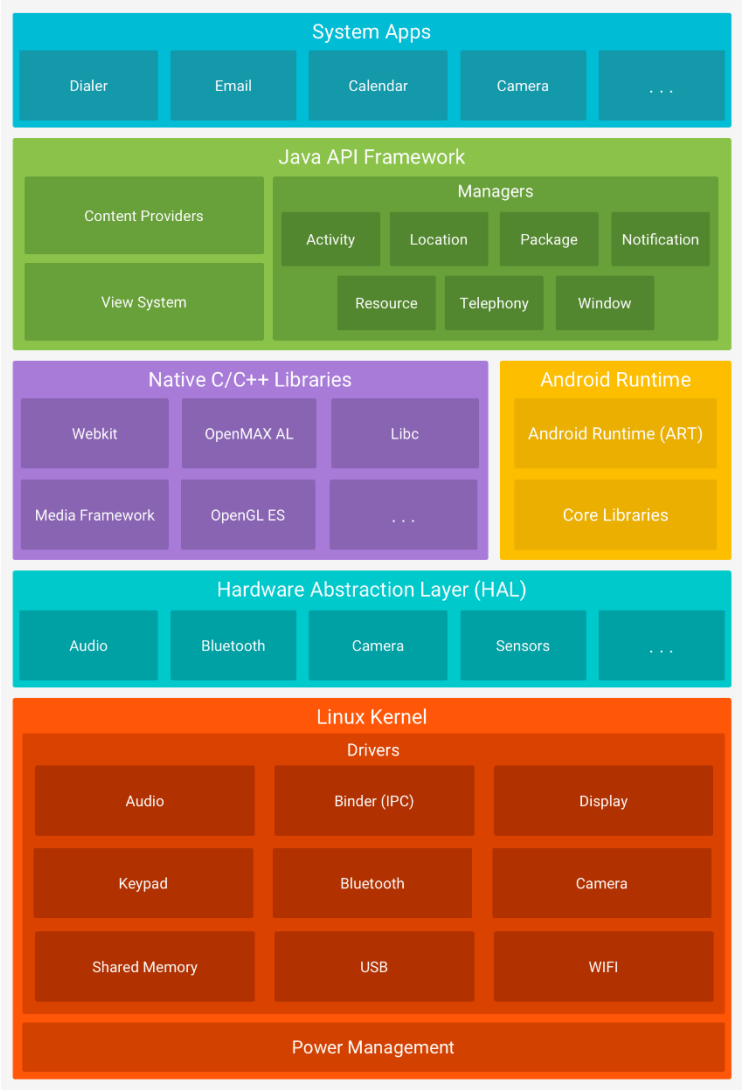


Figura 3.1 - Stiva de software Android[2]

[2] https://developer.android.com/guide/platform#native-libs - accesat pe 14.06.2021

Componentele principale ale arhitecturii Android

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

3.1.1 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.[3]

3.1.2 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 adăugiri sunt folosite în general pentru îmbunătățirea funcționalității sistemului și nu

[3] https://www.geeksforgeeks.org/android-architecture/ - accesat pe 14.06.2021

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.[4]

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.

[4] https://source.android.com/devices/architecture - accesat pe 14.06.2021

Limbajul definirii interfeței HAL - HIDL(HAL interface definition language) descrie interfața dintre o HAL și utilizatorii ei, permițând framework-ului Android să fie înlocuit 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 rândul 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 neexistând, producătorii dispozitivelor trebuia sa înlocuiască 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.[5]

3.1.3 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.

[5] https://source.android.com/devices/architecture - accesat pe 14.06.2021

- 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.

3.1.4 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.

3.1.4.1 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.

3.1.4.2 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.[6]

3.1.5 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.

[6] https://theiconic.tech/android-java-fdbd55aadc51 - accesat pe 14.06.2021

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 acces 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.

3.1.6 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.

3.1.6.1 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[7].

3.1.6.2 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, instanțiate 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.

3.1.6.3 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

[7] https://developer.android.com/guide/platform#native-libs - accesat pe 14.06.2021

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.

3.1.6.4 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.

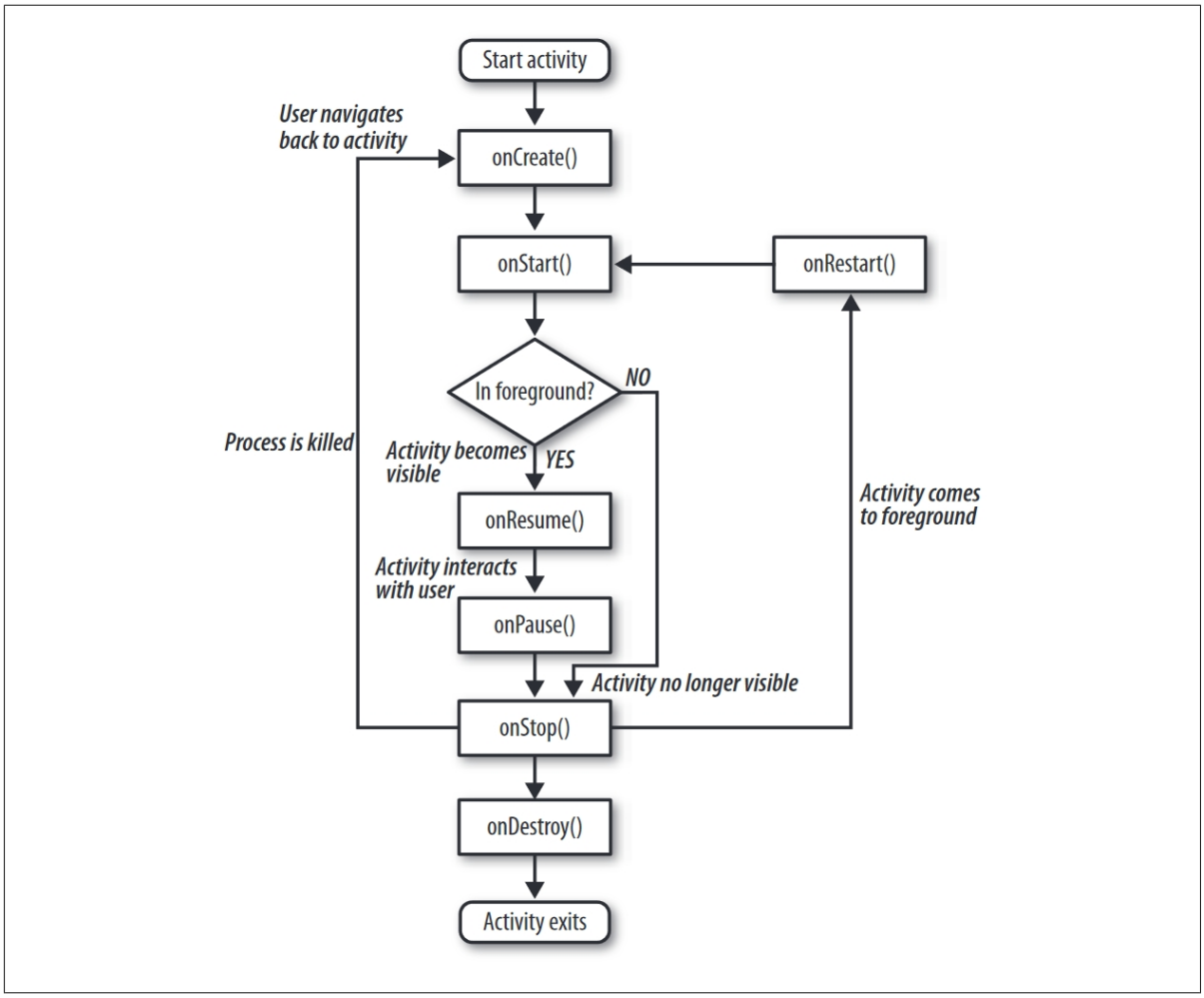


Figura 3.1.6.4 - Ciclul de viață al unei activități Android[8]

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.

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

- 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 va putea fi suspendată până când primește acces din nou la ecran. Aplicația va trebui din nou să utilizeze această metodă pentru a stoca orice stare de care va avea nevoie în caz că activitatea ei 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.

3.1.6.5 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 apelări 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.[9]

3.2 Java in contextul Android

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

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

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.

3.3 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[10].

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.

[10] https://www.sqlite.org/about.html - accesat pe 14.06.2021

3.4 RecyclerView si ViewPager2

RecyclerView este un widget mai flexibil și mai avansat decât 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 și refolosirea lor în loc de distrugerea lor astfel Îmbunătățind performanța dramatic și reducând consumul de energie. Putem folosi RecyclerView pentru colecții 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[11].

- RecyclerView este un ViewGroup care randează orice view bazat pe adaptoare în mod similar și este gândit ca fiind succesorul ListView și GridView. Unul din motivele pentru care RecyclerView are un framework mai extensibil este ca oferă posibilitatea implementării aranjărilor verticale dar și a celor orizontale sau de tip grile eșalonate.
- RecyclerView diferă față de predecesorul lui ListView prin următoarele caracteristici:
- adaptoarele ListView nu necesită folosirea modelelor ViewHolder pentru a îmbunătăți performanța față de RecyclerView care necesită un model ViewHolder pentru care folosește RecyclerView.Viewholder.
- Listview poate aranja termenii listei doar în mod vertical și acest lucru nu poate fi personalizat. În schimb RecyclerView are RecyclerView.LayoutManager care permite orice aranjament inclusiv cel orizontal sau grile eșalonate.
- În Listview nu se regăsesc mod special animații se pot Anima ștergerea sau adăugarea de obiecte Față de RecyclerView care conține clasa RecyclerView.ItemAnimator pentru gestionarea animațiilor.
- RecyclerView necesită o implementare manuală a surselor de date față de Listview care avea adaptoare pentru surse diferite ca ArrayAdapter pentru vectori sau CursorAdapter pentru rezultate din baza de date.

[11] https://www.guru99.com/recyclerview-android-list.html - accesat pe 14.06.2021

- ListView folosește proprietatea android:divider pentru adăugarea ușoară de separatoare între obiectele listei, în contrast cu RecyclerView care necesita folosirea RecyclerView.ItemDecoration pentru configurarea manuală a decorațiilor pentru separatoare.
- ListView are interfața AdapterView.OnItemClickListener pentru detectarea automată a evenimentelor tip click pe fiecare obiect din listă, în contrast cu RecyclerView care suportă doar RecyclerView.OnItemTouchListener pentru gestionarea evenimentelor tip touch individuale dar fără gestionare a click-urilor predefinită.

3.4.1 Componentele RecyclerView

RecyclerView necesită un LayoutManager și un adaptor pentru a fi instanțiat. LayoutManager determină pozițiile view-urilor în interiorul RecyclerView și determină când să reutilizeze obiectele care nu mai sunt vizibile utilizatorului. RecyclerView include următoarele componente:

- LinearLayoutManager generează liste cu derulare orizontală sau verticală, GridLayoutManager arată obiectele în aranjament tip grilă, iar StaggeredGridLayoutManager le aranjează în grilă eșalonată.
- RecyclerView.Adapter Este noul tip de adaptor asemănător cu cele folosite precedent dar cu câteva particularități, de exemplu un ViewHolder obligatoriu. Vor trebui suprascrise două metode principale una pentru a umfla(inflate) ViewHolder-ul și încă una pentru a lega datele de View fără nevoia verificării dacă este reciclat.
- RecyclerView.ItemAnimator va anima modificările ViewGroup-urilor ca adăugare, ștergere și selectare care sunt notificate către adaptor. DefaultItemAnimator poate fi folosit pentru animații de bază și funcționează foarte bine[12].

ViewPager2 moștenește ViewGroup așa că nu este compatibil cu ViewPager. Nucleul lui este format din RecyclerView și LinearLayoutManager care este de fapt un component al RecyclerView[13].

[12] https://guides.codepath.com/android/using-the-recyclerview - accesat pe 14.06.2021

[13] https://www.programmersought.com/article/95884244584/ - accesat pe 14.06.2021

Acest View permite dezvoltatorilor să afișeze View-uri sau fragmente utilizatorului într-un format de tip swipe astfel devenind o caracteristică comună în interfața grafică a aplicațiilor moderne.

ViewPager2 este adeseori integrat cu TabLayout pentru a indica pagina curentă și permite utilizatorului să navigheze prin pagini. Deși este succesorul bibliotecii ViewPager, ViewPager2 are o diferență majoră față de această bibliotecă deoarece folosește adaptorul RecyclerView în care View-urile sunt acum reciclate, astfel experiența utilizatorului este îmbunătățită cu tranziții line(smooth transitions) și minimizând utilizarea memoriei[14].

3.5 Fragmente

Înainte de introducerea fragmentelor exista o limitare care nu ne permitea să afișăm mai mult de o activitate pe ecran, așadar nu puteam diviza ecranul și controla părțile diferite separat. Odată cu introducerea fragmentelor am căpătat mai multă flexibilitate și am scăpat de limitarea de a avea o singură activitate pe ecran, fiecare activitate putând fi compusă din mai multe fragmente cu propriul aranjament, evenimente și ciclu de viață complet[15].

Fragmentele sunt porțiuni reutilizabile din interfața cu utilizatorul a aplicației. Un fragment își definește și își gestionează propriul aranjament, are propriul ciclu de viață poate, gestiona propriile evenimente de intrare. Fragmentele nu pot exista de sine stătător ci trebuie să fie găzduite de o altă activitate sau fragment, astfel ierarhia View-urilor a fragmentului devine parte din ierarhia View-urilor a gazdei.

Fragmentele introduc modularitate și reutilizabilitate în interfața grafică a activității permițând divizarea interfeței în bucăți discrete mai potrivite pentru definirea și gestionarea perfectă și a unui singur ecran sau a unei porțiuni din ecran față de activități care sunt locul ideal de a pune elementele globale ale aplicației ca un sertar de navigare.

Considerând o aplicație care răspunde dimensiunilor variate ale ecranelor, pe ecrane mai mari aplicația va trebui să afișeze un sertar static de navigație și o listă în aranjament tip grilă care pe ecrane mai mici vor deveni o bară de navigație la baza ecranului și o listă cu aranjament liniar. Gestionarea tuturor acestor variații în activitate poate fi dificilă, așadar

[14] https://www.section.io/engineering-education/android-viewpager2/ - accesat pe 14.06.2021

[15] https://www.tutorialspoint.com/android/android_fragments.htm - accesat pe 14.06.2021

separarea elementelor de navigare de conținut poate face acest proces mai facil. Activitatea este apoi responsabilă pentru afișarea elementului corect de navigare și a îl lista în aranjamentul potrivit.

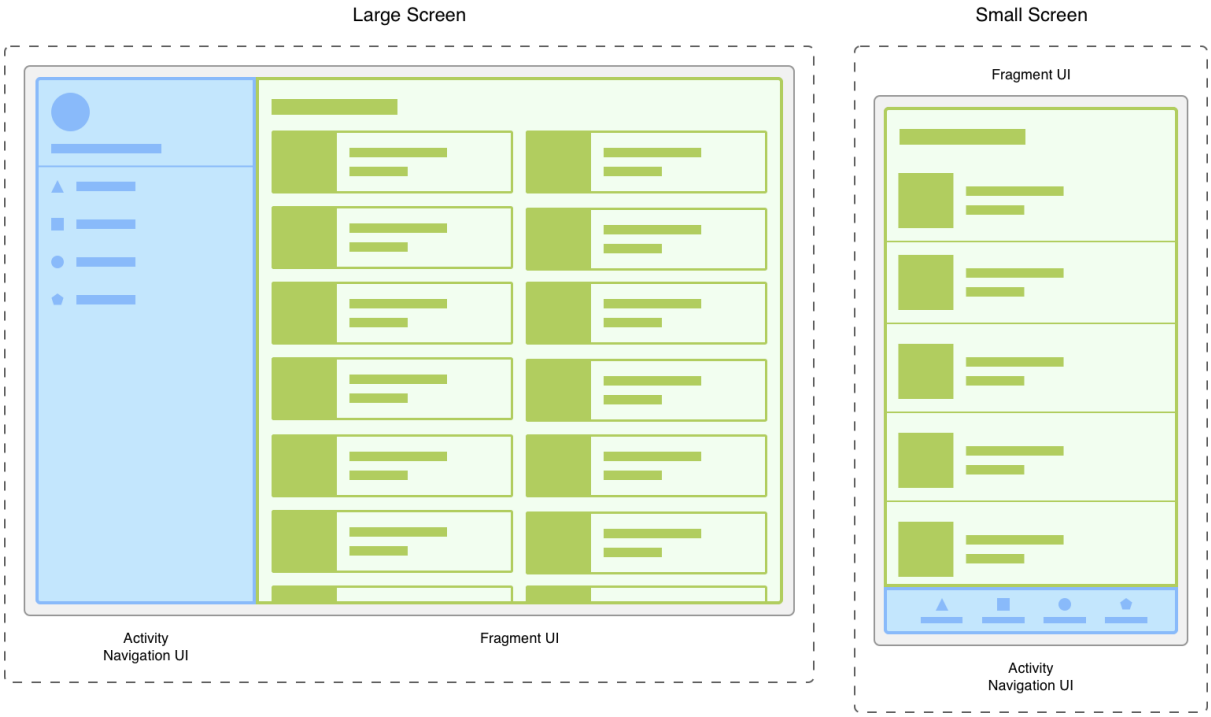


Figura 3.5.1 - Doua versiuni ale aceleiași activități pe ecrane diferite

Separarea interfeței cu utilizatorul în fragmente face mai ușoară modificarea aparenței aplicației în timpul rulării. Cât timp activitatea este în starea ciclului de viață STARTED sau mai sus, fragmentele pot fi adăugate, înlocuite sau scoase. Se poate păstra o înregistrare a schimbărilor în stiva funcției back care este gestionată de activitate permițând revenirea asupra modificărilor.

În aceeași activitate pot fi folosite multiple instanțe ale aceleiași clase de fragment. Ele pot fi folosite în multiple activități sau în alte fragmente, așadar ar trebui să furnizăm fragmentului doar logica necesară pentru a-și gestiona propria interfață ar trebui să evităm manipularea unui fragment din altul[16].

[16] https://developer.android.com/guide/fragments - accesat pe 14.06.2021

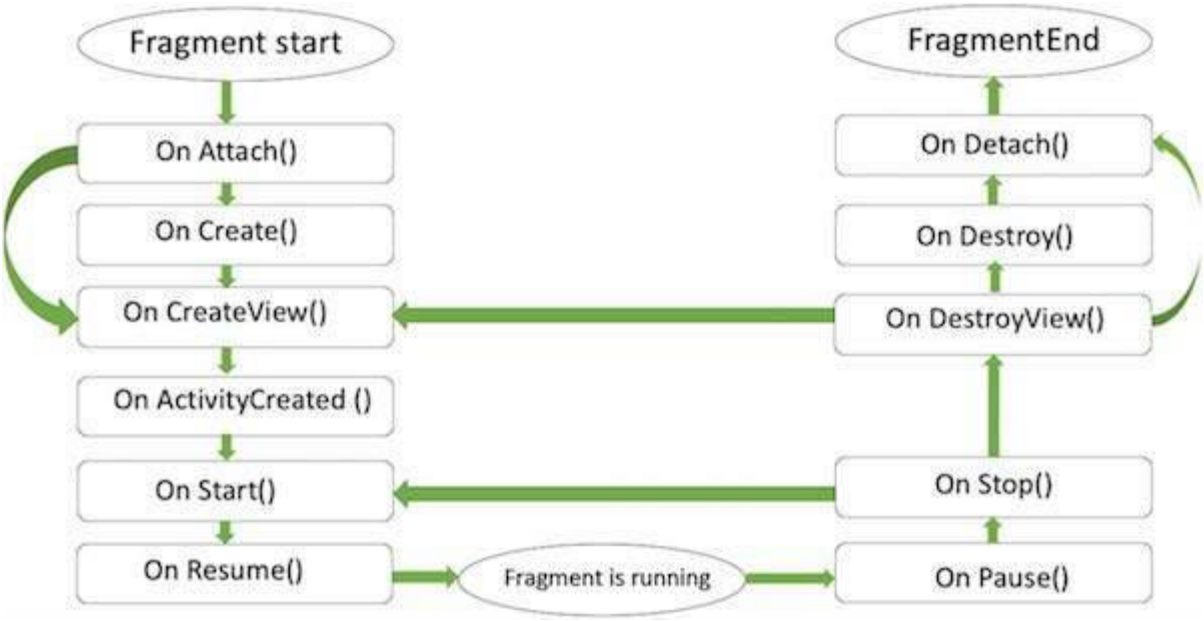


Figura 3.5.2 – Ciclul de viață al fragmentelor

4 Capitolul II - Etapele de dezvoltare a unei aplicații Android

Cu peste 2.5 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, încropirea 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, implicând 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.

4.1 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 obținută 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.

4.2 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 limitări 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.

4.3 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.

4.4 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.

4.5 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.

4.6 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 fiind 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.

4.7 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.

4.8 Lansarea aplicației

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[17].

5 Capitolul III – Aplicații utilizate

5.1 Git

Git este un sistem de control al versiunilor gratuit și cu sursă deschisă proiectat să gestioneze orice de la proiecte mici până la proiecte foarte mari rapid și eficient. Git este ușor de folosit și are o dimensiune foarte mică cu performanță ridicată. Ceea ce diferențiază Git de alte sisteme de versionare este modelul lui de ramificare care permite mai multe ramuri locale complet independente între ele. Crearea, reunirea sau ștergerea acestor linii de dezvoltare durează doar câteva secunde.

Indexul, sau zona de organizare, a lui Git este o zonă intermediară care permite formatarea și revizionarea schimbărilor înainte de a le comite.

Una din cele mai bune caracteristici ale Git este că este distribuit astfel că putem clona întregul repertoriu în loc să vedem doar cea mai nouă variantă a codului. Astfel avem acces la multiple copii de rezervă cu toate versiunile. Fiind parte dintr-un flux de lucru centralizat, fiecare utilizator are o copie de rezervă întreagă a serverului central. Astfel fiecare dintre aceste copii poate fi împinsă să înlocuiască copia de pe serverul central în cazul defectării sau coruperii acestuia. Nu există un singur punct de eșec pentru Git decât dacă există o singură copie a depozitului[18].

[17] https://medium.com/swlh/what-are-the-various-phases-of-mobile-app-development-4f0a1748e619 - accesat pe 14.06.2021

[18] http://git-scm.com/about - accesat pe 14.06.2021

5.2 GitHub

GitHub este un furnizor de găzduire pentru dezvoltare software și controlul versiunilor folosind Git. El oferă caracteristicile Git plus câteva caracteristici proprii cum ar fi controlul accesului, urmărirea bug-urilor, cereri de noi funcții, gestionarea task-urilor, integrare continuă pentru fiecare proiect. GitHub își oferă serviciile în general gratuit dar serviciile mai avansate, profesionale sau enterprise sunt cu plată. Conturile gratuite sunt mai des folosite pentru găzduirea proiectelor open-source. Din ianuarie 2019 GitHub oferă un număr nelimitat de depozite private pentru toate conturile dar permite doar până la 3 colaboratori pe depozit în mod gratuit. În aprilie 2020 conturile gratuite permit colaboratori nelimitați dar limitează numărul de acțiuni GitHub la 2.000 de minute pe lună[19].

5.3 Android Studio

Android Studio este mediul integrat de dezvoltare (IDE - Integrated Development Environment) oficial pentru dezvoltarea aplicațiilor Android. Acesta este bazat pe editorul de cod și unelte de dezvoltare IntelliJ IDEA, și oferă în plus alte caracteristici care cresc productivitatea în timpul construirii aplicațiilor Android, ca de exemplu:

- Un sistem flexibil de Build bazat pe gradle.
- Un emulator rapid și bogat în funcții.
- Un mediu unificat în care se poate dezvolta pentru toate dispozitivele Android.
- Poate aplica schimbări și împinge aceste schimbări de cod sau resurse către aplicația care rulează fără a fi necesară repornirea acesteia.
- Conține modele de cod și integrare cu GitHub pentru a ajuta la construirea aplicațiilor obișnuite și pentru a importa exemple de cod rapid.
- Variate instrumente de testare și framework-uri.
- Unelte pentru a detecta probleme de performanță, utilizabilitate și compatibilitate între versiuni.
- Suport pentru C++ și NDK(Native Development Kit).
- Suport integrat pentru platforma Cloud Google care face ușoară integrarea mesageriei Google Cloud și a motorului de aplicații.

[19] https://en.wikipedia.org/wiki/GitHub - accesat pe 14.06.2021

Fiecare proiect din Android Studio conține unul sau mai multe module cu fișiere cu cod sursă și fișiere de resurse[20].

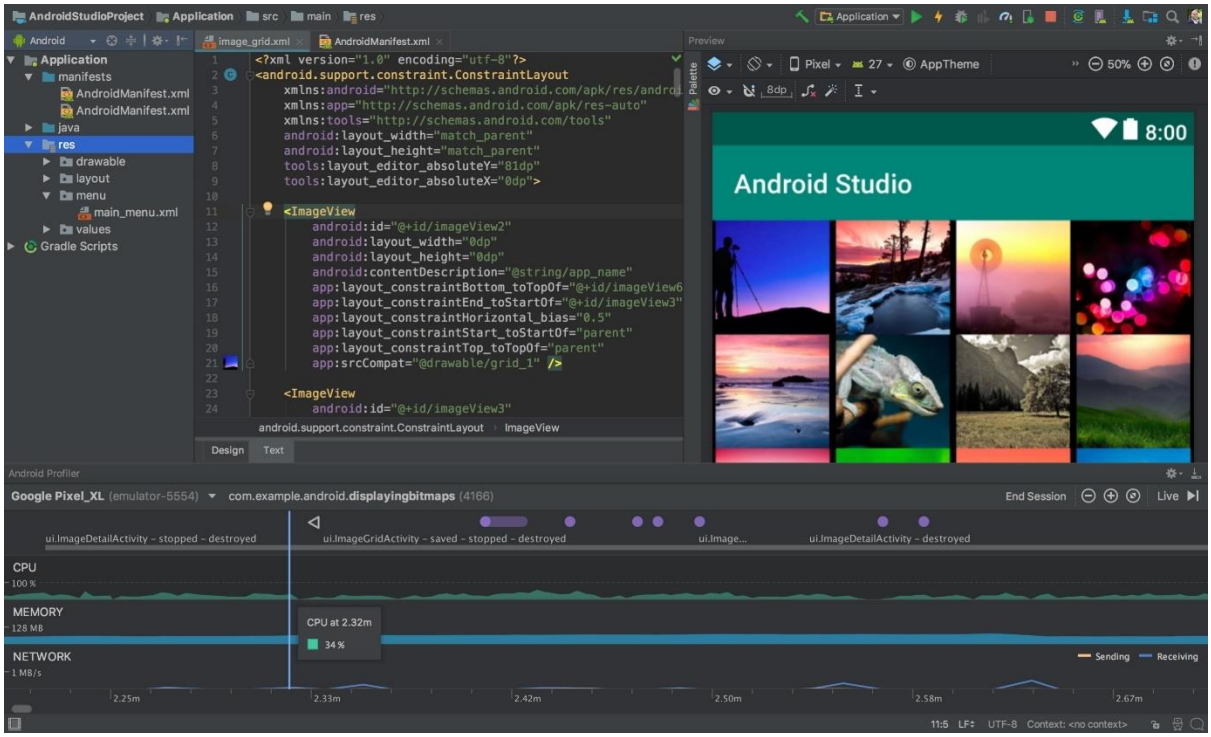


Figura 5.3.1 – Un proiect deschis în Android Studio[21]

5.4 Gradle

Gradle este o unealtă Open source pentru automatizarea construirii softurilor, concentrat pe flexibilitate și performanță. Scripturile de construcție gradle sunt scrise folosind un limbaj specific de domeniu(DSL - Domain-Specific-Language) Groovy sau Kotlin DSL. Gradle este construit în așa fel încât poate fi foarte personalizabil. El folosește rezultate din execuții anterioare procesând doar intrările care s-au schimbat și executând task-uri în paralel, iar fiind unealta oficială de Build pentru Android oferă suport pentru cele mai populare limbaje și tehnologii[22].

Gradle a fost proiectat pentru builduri multiproiect care pot fi foarte mari. El este bazat pe o serie de task-uri de Build care pot rula serial sau în paralel. Builduri incrementale sunt suportate prin determinarea bucăților din arborele buildului care sunt deja la zi. Orice task bazat

[20] https://developer.android.com/studio/intro - accesat pe 14.06.2021

[21] https://developer.android.com/studio/ - accesat pe 14.06.2021

[22] https://docs.gradle.org/current/userguide/userguide.html - accesat pe 14.06.2021

numai pe aceste bucăți nu necesită a fi executat din nou. De asemenea suportă stocarea în cloud a componentelor lor de construcție peste rețea utilizând Gradle Build Cache[23].

5.5 Android emulator - qemu

Emulatoarele de Android sunt softuri proiectate să simuleze dispozitivele hardware și software Android pentru scopurile de dezvoltare și testare. Emulatoare de Android pot rula și pe Mac și pe PC, astfel încât aplicațiile Android să poată fi dezvoltate în întregime la calculator fără a fi necesar un dispozitiv mobil. Alte motive populare pentru folosirea emulatoarelor Android sunt jucarea sau testarea jocurilor mobile dar pot fi folosite și pentru rularea oricărei aplicații din Google Play Store.

Ca unealtă Open source QEMU(Quick Emulator) rulează pe multe tipuri de sisteme gazdă cu sisteme de operare și procesoare diverse pentru a emula alte procesoare și sisteme de operare. QEMU oferă emulatoarelor Android puterea de a simula hardware-ul unui dispozitiv diferit. Această translație este complexă și durează destul de mult făcând emularea foarte lentă dar dacă arhitecturile dispozitivului gazdă și al celui emulat sunt similare translația devine ușoară și rapidă[24].

5.6 Trello

Trello este o unealtă vizuală pentru organizarea proiectelor de la muncă sau de acasă. Este o aplicație web care implementează un șablon de tipul Kanban Board în care se pot crea panouri de lucru, liste și task-uri care se pot muta între panourile de lucru. De obicei aceste coloane includ statusuri ca: De făcut, În derulare, Terminat(To Do, In Progress, Done). Aceasta poate fi folosită în scopuri personale sau de afaceri incluzând management imobiliar, dezvoltarea software, managementul proiectelor, aviziere școlare, planificarea lecțiilor, contabilitate, dezvoltarea paginilor web, dezvoltarea jocurilor și managementul cazurilor la birouri de avocatura[25].

[23] https://en.wikipedia.org/wiki/Gradle - accesat pe 14.06.2021

[24] https://theqalead.com/topics/what-is-an-android-emulator/ - accesat pe 14.06.2021

[25] https://en.wikipedia.org/wiki/Trello - accesat pe 14.06.2021

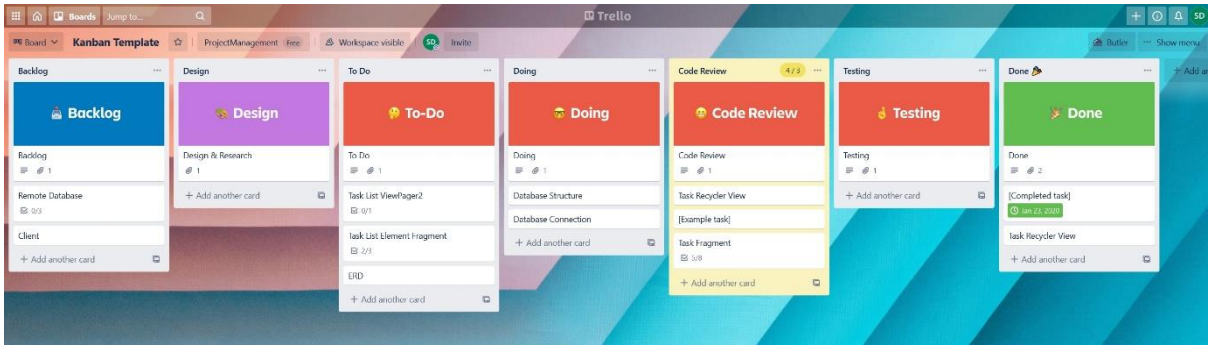


Figura 5.6.1 – Exemplu de Template Kanban din Trello

În documentul curent se va descrie o aplicație cu o structură a utilizării similară și care încearcă să acopere o parte din caracteristicile modelului Kanban al lui Trello, dar realizată în Java pentru platforma Android.

6 Capitolul IV – Arhitectura aplicației

Aplicațiile mobile trebuie să facă fată la modurile haotice în care sunt utilizate. Utilizatorii pot fi întrerupți din utilizarea aplicației de către o notificare de primire a unui mesaj sau un apel telefonic la care vor trebui să răspundă, trimițând aplicația curentă în fundal și revenind la ea după o perioadă scurtă sau mai lungă de timp. Dacă în timp ce aplicația este în fundal sistemul de operare Android are nevoie de mai multă memorie o poate închide. Astfel dacă datele din aplicație nu sunt gestionate corect se pot pierde. Pentru a evita aceste pierderi de date aplicația scrie direct în baza de date locală SQLite la fiecare modificare realizată.

Pentru a fi compatibilă cu cât mai multe dispozitive mobile, aplicația folosește ca versiune minimă 21 a SDK-ului și a fost dezvoltată pe Android Studio 4.2.1, Runtime 11.0.8.

În următoarele subcapitole voi prezenta implementarea arhitecturii aplicației împărțită pe elementele structurale componente.

6.1 Clasele model

Pentru a putea gestiona datele din spatele fiecărui tip de obiect creat pentru aplicație am realizat câte o clasă model. Aceste clase conțin datele proprii precum și settere si gettere pentru a putea fi apelate din aplicație.

Aceste modele vor fi folosite pentru crearea structurii tabelelor în baza de date. Numele și descrierea sunt variabile de tip String și se vor folosi in interfața grafică a aplicației pentru popularea câmpurilor de text din Fragmentul de detaliu al Task-ului dar și din view-urile din RecyclerView-ul din Fragmentul listei de Task-uri. La crearea unui Task nou este necesară o

instanță a clasei Task pentru a-l putea salva, ID-ul Task-ului este apoi auto-incrementat de baza de date. List_id este ID-ul listei din care face parte Task-ul, cu ajutorul lui sunt generate listele de Task-uri specifice fiecărei categorii.

```
public Task(int id, String name, String description, int list_id) {
    this.id = id;
    this.name = name;
    this.description = description;
    this.list_id = list_id;
}
```

Utilizatorii se pot înregistra în baza de date prin intermediul activității RegisterActivity care folosește clasa User ca model pentru a putea salva datele. Identificatorul folosit apoi pentru logare este email-ul.

```
public User(int id, String name, String email, String password,
String role_name) {
    this.id = id;
    this.name = name;
    this.email = email;
    this.password = password;
    this.role_name = role_name;
}
```

Altă clasă folosită în aplicație este clasa TaskList care este folosită pentru organizarea Task-urilor în liste și de asemenea conține ID-ul proiectului din care face parte. Pe lângă aceste clase am mai creat clasele model Role și Project care urmau să fie utilizate în implementarea sistemului de drepturi pentru utilizatori și respectiv a unui sistem de creare și gestionare a proiectelor multiple cu drepturi de acces diferite ale fiecărui utilizator asupra fiecărui proiect. Aceste module au rămas în categoria dezvoltărilor ulterioare.

6.2 Structura bazei de date

Baza de date este realizată inițial folosind SQLite pentru salvarea rapidă a datelor direct în memoria telefonului, în a doua fază a dezvoltării ar urma dezvoltarea unui conector cu o bază de date remote care va păstra încă o copie a datelor și de asemenea va permite colaborarea între utilizatori.

Structura tabelelor bazei de date poate fi ușor dedusa prin aranjamentul accesibil al valorilor String care determină denumirile tabelelor si ale coloanelor.

```
public class SQLiteDatabaseHelper extends SQLiteOpenHelper {
    public static final String LOG = "DBHelper";
    public static final String DB_NAME = "project.db";

public static final String TABLE_USER = "user";
  public static final String COLUMN_USER_ID = "user_id";
  public static final String COLUMN_USER_NAME = "user_name";
  public static final String COLUMN_USER_EMAIL = "user_email";
  public static final String COLUMN_USER_PASSWORD = "user_password";
  public static final String COLUMN_USER_ROLE = "user_role";

public static final String TABLE_PROJECT = "project";
  public static final String COLUMN_PROJECT_ID = "project_id";
  public static final String COLUMN_PROJECT_NAME = "project_name";
  public static final String COLUMN_PROJECT_DESCRIPTION = "project_description";
  public static final String COLUMN_PROJECT_CREATEDTIME = "project_createdtime";
  public static final String COLUMN_PROJECT_DEADLINE = "project_deadline";
  public static final String COLUMN_PROJECT_CREATEDBYUSERID = "project_createdbyuserid";
  public static final String COLUMN_PROJECT_SHAREDTOUSERID = "project_sharedtouserid";

public static final String TABLE_LIST = "list";
  public static final String COLUMN_LIST_ID = "list_id";
  public static final String COLUMN_LIST_NAME = "list_name";
  public static final String COLUMN_LIST_DESCRIPTION = "list_description";
  public static final String COLUMN_LIST_ICON = "list_icon";
  public static final String COLUMN_LIST_PROJECT_ID = "list_project_id";

public static final String TABLE_TASK = "task";
  public static final String COLUMN_TASK_ID = "task_id";
  public static final String COLUMN_TASK_NAME = "task_name";
  public static final String COLUMN_TASK_DESCRIPTION = "task_description";
  public static final String COLUMN_TASK_LIST_ID = "task_list_id";
```

Deoarece clasa SQLiteDatabasaHelper are peste 650 de linii de cod voi selecta câteva metode remarcabile pentru înțelegerea mai ușoară a structurii bazei de date și a felului în care funcționează introducerile și cererile de date.

```
public ArrayList<User> getAllUsers(){        //returnarea tuturor utilizatorilor
    ArrayList<User> allUsers = new ArrayList<User>();
    String queryGetAllUsers = "SELECT * FROM " + TABLE_USER;
    Log.e(LOG, queryGetAllUsers);
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(queryGetAllUsers, null);
    if (cursor.moveToFirst()){
        do {
            User user = new User();
user.setId(cursor.getInt(cursor.getColumnIndex(COLUMN_USER_ID)));
user.setName(cursor.getString(cursor.getColumnIndex(COLUMN_USER_NAME)));
user.setEmail(cursor.getString(cursor.getColumnIndex(COLUMN_USER_EMAIL)));
user.setPassword(cursor.getString(cursor.getColumnIndex(COLUMN_USER_PASSWORD)));
user.setRole_name(cursor.getString(cursor.getColumnIndex(COLUMN_USER_ROLE)));
            allUsers.add(user);
        }while (cursor.moveToNext());
    }
    cursor.close();
    db.close();
    return allUsers;
}
```

Metoda getAllUsers este folosita pentru verificarea existenței a cel puțin un utilizator pentru funcția de populare a bazei de date cu date mostră.

```
public ArrayList<ArrayList<Task>> getAllTaskListsTasks(int projectID){
//returnarea tuturor Task-urilor in cate un ArrayList pentru fiecare lista de
Task-uri
    ArrayList<ArrayList<Task>> projectLists = new ArrayList<>();
    ArrayList<Integer> listIds = new ArrayList<>();
    ArrayList<TaskList> lists = (ArrayList<TaskList>)
getAllTaskLists(projectID);
    for (int i = 0; i < lists.size(); i++) {
        listIds.add(lists.get(i).getId());
    }
    for (int i = 1; i <= listIds.size(); i++) {
        ArrayList<Task> tasks;
        tasks = (ArrayList<Task>) getListTasks(i);
        projectLists.add(tasks);
    }
    return projectLists;
```

Metoda getAllTaskListsTasks furnizează un vector de vectori de Task-uri folosit la afișarea listelor de Task-uri din RecyclerView din fragmentele generate de ViewPager2.

```
public int deleteTask(int taskId){     //Stergerea unui Task
    SQLiteDatabase db = getWritableDatabase();
    int deletedTaskId = db.delete(TABLE_TASK,COLUMN_TASK_ID + " = ?", new
String[]{String.valueOf(taskId)} );
    db.close();
    return deletedTaskId;
}
```

deleteTask este metoda apelata la ștergerea unui Task dar și la ștergerea fiecărui task din interiorul unei liste aflată în procesul de a fi ștearsă.

```
public int insertTaskList(int currentTaskListId, TaskList newTaskList){
//inserarea unei liste goale la pozitia curenta
    ArrayList<TaskList> taskList = getAllTaskLists(1);
    int dbtaskListSize = taskList.size();
    ArrayList<Task> listTasks = new ArrayList<>();
    taskList.add(currentTaskListId-1, newTaskList);
    for (int i = taskList.size(); i >= currentTaskListId; i--) {
//mutarea tutror Task-urilor si listelor de Task-uri spre sfarsit
        listTasks.clear();
        listTasks = getListTasks(i);
        for (int j = 0; j < listTasks.size(); j++) {
            listTasks.get(j).setList_id(listTasks.get(j).getList_id()+1);
            editTask(listTasks.get(j));
        }
        TaskList currentElement = new TaskList();
        currentElement=taskList.get(i-1);
        currentElement.setId(i);
        ContentValues values = new ContentValues();
        values.put(COLUMN_LIST_ID, currentElement.getId());
        values.put(COLUMN_LIST_NAME, currentElement.getName());
        values.put(COLUMN_LIST_DESCRIPTION, currentElement.getDescription());
        values.put(COLUMN_LIST_ICON, currentElement.getIcon());
        values.put(COLUMN_LIST_PROJECT_ID, currentElement.getProject_id());
        if (taskList.size() > dbtaskListSize) {
            dbtaskListSize++;
            SQLiteDatabase db = this.getWritableDatabase();
            db.insert(TABLE_LIST, null, values);
            db.close();
        }else {
            SQLiteDatabase db = this.getWritableDatabase();
            db.update(TABLE_LIST, values, COLUMN_LIST_ID + " = ?", new
String[]{String.valueOf(currentElement.getId())});
            db.close();
        }
    }
    int insertedTaskListId = taskList.get(currentTaskListId-1).getId();
    return insertedTaskListId;
}
```

Metoda insertTaskList este folosită pentru a introduce o listă între altele 2 și execută mutarea ID-urilor tuturor celorlalte liste de Task-uri și Task-uri de la dreapta față de ea.

```
public long createTask(Task task){                    //crearea unui task
    SQLiteDatabase db = getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_TASK_NAME, task.getName());
    values.put(COLUMN_TASK_DESCRIPTION, task.getDescription());
    values.put(COLUMN_TASK_LIST_ID, task.getList_id());
    long taskId = db.insert(TABLE_TASK, null, values);
    db.close();
    return taskId;
```

Metoda createTask este folosită la crearea unui Task nou prin intermediul butonului NEW TASK din activitatea principală.

6.3 Autentificare – Înregistrare

Activitatea de autentificare conține câmpurile editabile de text Email și Password și butoanele Login și Register. Pe butonul Login este implementat un OnClickListener care verifică dacă ambele câmpuri au fost completate și notifică utilizatorul despre lipsa datelor în oricare din câmpuri prin afișarea unui mesaj de tip Toast. În cazul în care utilizatorul a greșit la introducerea adresei email sau adresa introdusă nu exista în baza de date este afișat un alt mesaj tip Toast „Email address not registered!” apoi dacă emailul există în baza de date se verifică existența combinației email + parola prin apelarea metodei checkUserEmailPassCombo din clasa Helper a bazei de date.

```
    public void onClick(View v) {
        String email = emailLogin.getText().toString();
        String password = passwordLogin.getText().toString();
        if (email.equals("") || password.equals(""))
//Verificarea prezentei textului in ambele campuri
            Toast.makeText(LoginActivity.this, "Please fill all
fields!", Toast.LENGTH_LONG).show();
        else {
            int checkuser = db.checkIsUserEmail(email);
            if(checkuser < 1){
//Verificarea existentei adresei mail
                Toast.makeText(LoginActivity.this, "Email address not
registered!", Toast.LENGTH_LONG).show();
            }else {
                int checkUserPass = db.checkUserEmailPassCombo(email,
password);
                if (checkUserPass > -1){
//Verificarea combinatiei user + parola
                    Intent intent =new Intent(getApplicationContext(),
MainActivity.class);
                    intent.putExtra("email", email);
                    //finish();
//Intoarcerea in activitatea principala
                    startActivity(intent);
                }else
                    Toast.makeText(LoginActivity.this, "Email password
combination incorrect", Toast.LENGTH_LONG).show();

            }
        }

    }
});
```

Dacă combinația există atunci emailul utilizatorului este atașat unei intenții cu metoda Intent.putExtra apoi activitatea principală este pornită cu startActivity(intent). Dacă combinația nu există în baza de date, utilizatorul este notificat prin un mesaj de tip Toast de această discrepanță.

În cazul în care utilizatorul nu are cont sau dacă dorește crearea unui alt cont poate folosi butonul Register pentru a ajunge în activitatea de înregistrare unde își va putea crea un profil nou de utilizator. În aceasta activitate se găsesc câmpurile de text editabil Email, Name, Password și Repeat Password și butonul Register. Pe acest buton este implementat un alt OnClickListener care verifică daca toate câmpurile au fost completate, verifică dacă ambele câmpuri pentru parolă sunt identice și dacă emailul nu este deja folosit. Dacă toate verificările au rezultat pozitiv atunci utilizatorul este creat în baza de date apelând metoda createUser(user) a clasei helper a bazei de date apoi emailul nou introdus este atașat unei intenții care este folosită pentru pornirea activității principale.

```
registerButton.setOnClickListener(new View.OnClickListener() {
//Buton pentru inregistrarea utilizatorului
        @Override
        public void onClick(View v) {
            String email = emailRegister.getText().toString();
            String userName = userNameRegister.getText().toString();
            String password = passwordRegister.getText().toString();
            String passwordRepeat = passwordRepeatRegister.getText().toString();
            User user = new User(userName, email, password, "admin");
//Validarea informatiilor completate in  campuri
            if (email.equals("") || password.equals("") ||
passwordRepeat.equals(""))
                Toast.makeText(RegisterActivity.this, "Please fill all fields!",
Toast.LENGTH_LONG).show();
            else
                if (!password.equals(passwordRepeat))
                    Toast.makeText(RegisterActivity.this, "Passwords do not
match!", Toast.LENGTH_LONG).show();
                else{
                    int checkuser = db.checkIsUserEmail(email);
                    if(checkuser > 0) {
                        Toast.makeText(RegisterActivity.this, "Email already in
use!", Toast.LENGTH_LONG).show();
                    }else{
                        db.createUser(user);
//Inregistrarea utilizatorului in baza de date
                        Toast.makeText(RegisterActivity.this, "User created",
Toast.LENGTH_LONG).show();
                        Intent intent = new Intent(getApplicationContext(),
MainActivity.class);        //Autentificarea utilizatorului
                        intent.putExtra("email", email);
                        startActivity(intent);
                    }
                }

        }
    });
}
```

6.4 ViewPager2

Odată ajunși la pagina principală utilizatorii vor fi întâmpinați de un TabLayout care conține numele listelor deja existente în Tab-urile lui, iar sub el un RecyclerView în care este vizibilă lista de Task-uri corespunzătoare Tab-ului selectat din TabLayout. Acest RecyclerView se află într-un Fragment care este generat de FragmentStateAdapter al unui ViewPager2 cu care sunt sincronizate Tab-urile din TabLayout.

```
taskArrayList = (ArrayList<Task>)
getArguments().getSerializable("taskArray");
//recuperarea listei de Task-uri din argumente
RecyclerView.LayoutManager layoutManager = new
LinearLayoutManager(getContext());
taskRecyclerView.setLayoutManager(layoutManager);
```

View-urile din RecyclerView sunt alimentate cu date prin adaptorul TaskAdapter. Aceste date ajung în adaptor din Fragmentul generat în ViewPager2, care, la rândul lui, primește datele din argumentele extra din intenție. Aceste argumente sunt populate în TaskListAdapter cu date de forma putSerializable care conțin un vector ArrayList<Task> de elemente tip Task. Din adaptorul TaskListAdapter până la adaptorul din RecyclerView datele circulă sub aceasta formă apoi sunt afișate în View-urile lui sub forma unor TextView.

În activitatea principală datele sunt extrase din baza de date prin intermediul a două metode din clasa helper getAllTaskListsTasks pentru toate task-urile sub forma unui vector de vectori Task-uri (lista de liste de Task-uri) și getAllTaskLists care oferă un vector al obiectelor clasei TaskList, obiecte care conțin denumirea listelor, descrierea lor și o variabila String pentru stocarea unui URI către o imagine inițial prevăzută pentru identificarea rapidă a listelor. Această imagine încă nu este implementată în versiunea curentă.

```
SQLiteDatabaseHelper db = new SQLiteDatabaseHelper(this);
//Citirea Task-urilor pe liste si a listelor
ArrayList<ArrayList<Task>> taskListsTasks = db.getAllTaskListsTasks(1);
ArrayList<TaskList> taskLists = db.getAllTaskLists(1);
```

După retragerea datelor din baza de date, vectorul de TaskList ArrayList<TaskList> este trimis mai departe către TabLayout pentru a genera toate Tab-urile de deasupra RecyclerView-urilor, iar vectorul de vectori de Task ArrayList<ArrayList<Task>> este trimis mai departe FragmentManager-ului din adaptorul TaskListAdapter.

```
FragmentManager fragmentManager = getSupportFragmentManager();
taskListAdapter = new TaskListAdapter(fragmentManager, getLifecycle(),
taskListsTasks, taskLists);      //crearea ViewPager-ului
listViewPager2.setAdapter(taskListAdapter);
for (int i = 0; i < taskLists.size(); i++) {
//popularea Tab-urilor de deasupra ViewPager-ului

listTabLayout.addTab(listTabLayout.newTab().setText(taskLists.get(i).name
));
}
```

Butoanele din TabLayout au implementat pe ele două tipuri de listener.

La selectarea Tab-ului pagina este mutată pe poziția corespunzătoare.

```
public void onTabSelected(TabLayout.Tab tab) {
    listViewPager2.setCurrentItem(tab.getPosition());
```

La apăsare prelungită este setat câte un listener pe fiecare buton care identifică Tab-ul curent și deschide fragmentul de editare corespunzător listei selectate.

```
LinearLayout listTabStrip = (LinearLayout) listTabLayout.getChildAt(0);
//setarea OnLongClickListener pe fiecare Tab pentru editarea listei
for (int i = 0; i < listTabStrip.getChildCount(); i++) {
    View currentChild = listTabStrip.getChildAt(i);
    currentChild.setOnLongClickListener(new View.OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
            for (int j = 0; j < taskLists.size(); j++) {
                String currentChildName =
currentChild.getTooltipText().toString();
                if (taskLists.get(j).getName().equals(currentChildName))
                    openTaskListFragment(taskLists.get(j).getId());
            }
            return true;
        }
    });
}
```

În adaptorul TaskListAdapter al lui ViewPager2 datele ajung din activitatea principală sub forma unui vector de vectori de Task-uri ArrayList<ArrayList<Task>> și sub forma unui vector de liste de Task-uri ArrayList<TaskList>. Aceste date sunt apoi redistribuite fragmentelor care conțin RecyclerView.

6.5 Fragmente cu RecyclerView

Aceste fragmente conțin RecyclerView care afișează datele cu care utilizatorul va interacționa cel mai mult. Ele sunt generate de ViewPager2 și reciclate când ies din ecran pentru a economisi resurse, iar, pentru a putea fi create, metoda de deschidere a unui fragment este apelată din activitatea principală prin intermediul unei interfețe.

```
public void onItemClick(Task task) {
    ((MainActivity) getActivity()).openTaskFragment(task);
```

Fragmentele sunt apoi generate conform codului de mai jos cu adăugarea în Bundle a vectorului tip ArrayList<Task> și o verificare dacă lista este goala să se trimită un ArrayList<Task> nou pentru a evita o eroare de tip Null Pointer Exception.

```
public Fragment createFragment(int position) {    //crearea unui fragment
   TaskListFragment taskListFragment =  new TaskListFragment();
    Bundle listTaskBundle = new Bundle();
    if (taskListsTasks.size() > position)
//trimiterea datelor listei si a listei de Task-uri catre fragment prin
argumente

listTaskBundle.putSerializable("taskArray",taskListsTasks.get(position));
    else
        listTaskBundle.putSerializable("taskArray",new
ArrayList<ArrayList<Task>>());
    taskListFragment.setArguments(listTaskBundle);
    return taskListFragment;
}
```

Pe View-urile individuale din lista Recycler View sunt setate alte două listenere OnClickListener pentru schimbarea culorii de fundal din spatele unui obiect din listă cu care se simulează selectarea și deselectarea obiectului pentru permiterea funcțiilor butoanelor de la baza activității principale.

```
holder.parentTask_layout.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        int previousTaskItem = selectedTaskItem;
        selectedTaskItem = position;
        selectedTask = task;
```

Al doilea listener setOnLongClickListener este folosit pentru deschiderea fragmentului de detaliu al Task-urilor în care se pot modifica acestea.

```
public boolean onLongClick(View v) {
    Log.d(TAG, "onClick: Clicked on: " + task.getName() + task);
    listener.onItemClick(task);
    return false;
```

6.6 Fragmente de editare

La apăsarea lungă pe obiectele din RecyclerView sau pe Tab-urile din TabLayout se deschid fragmentele care permit editarea Task-urilor și respectiv a listelor de Task-uri.

Ambele conțin cate un câmp de text editabil pentru nume și descriere și două butoane, unul pentru salvarea modificărilor efectuate și unul pentru ștergerea obiectului corespunzător.

In cazul Task-urilor butonul de ștergere afectează doar Task-ul selectat, dar în cazul listelor de Task-uri, pe lângă lista selectat se șterg apelând metoda clasei helper a bazei de date deleteTaskList și toate Task-urile aparținând de ea, iar toate listele și Task-urile sunt mutate spre stânga la nivel de ID-uri pentru a permite logicii de organizare bazata pe ID-urile listelor să funcționeze mai departe.

7 Capitolul V – Funcționalitatea aplicației

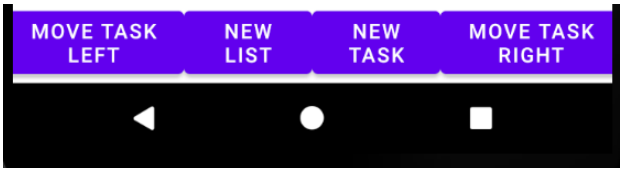


Figura 7.0.1 – Bara de butoane de la baza ecranului

Cele patru butoane din activitatea principală pot fi folosite pentru mutarea Task-urilor la stânga sau la dreapta(butoanele MOVE TASK LEFT și MOVE TASK RIGHT). Mai întâi se selectează unul din Task-uri prin o simpla atingere apoi se apasă butonul corespunzător direcției în care vrem să mutăm taskul în liste.

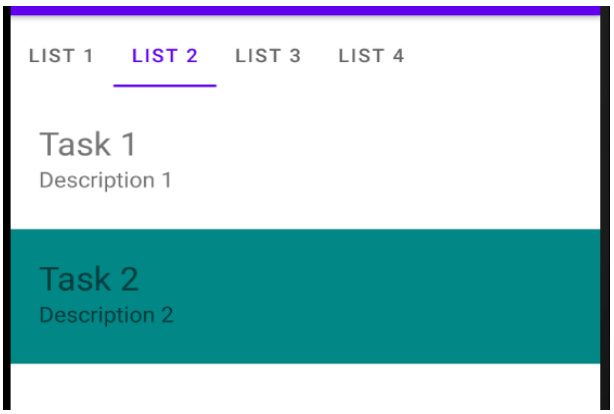


Figura 7.0.2 – Selectarea unui Task din listă

Butonul NEW LIST creează o nouă listă în locul listei selectate și deschide fragmentul care permite editarea datelor din listă și salvarea ei sau ștergerea în cazul în care se renunță la crearea ei.

Butonul NEW TASK creează un nou Task în lista selectată și deschide fragmentul corespunzător editării lui care permite salvarea sau ștergerea Task-ului nou creat.

Ambele butoane creează obiectele respective chiar daca butonul SAVE nu este apăsat și ele vor conține date default.

In cazul în care în baza de date de pe telefon nu exista un utilizator metoda

```
private void initTasklist()
```

Inițializează baza de date cu utilizatorul admin și date de test pentru liste și Task-uri.

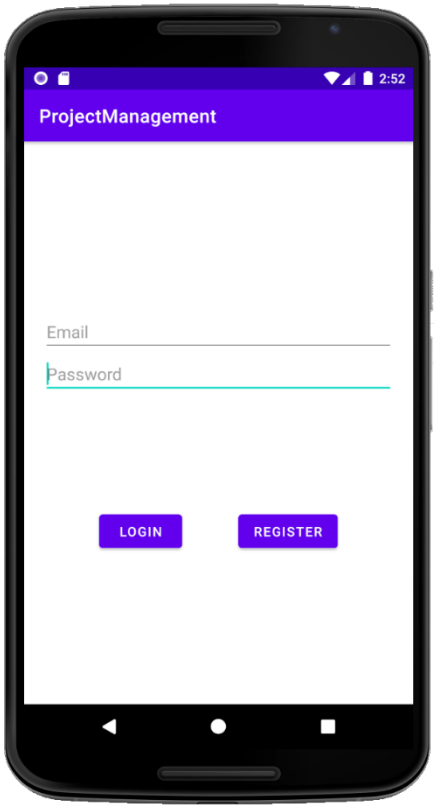
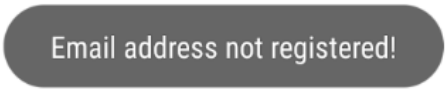


Figura 7.1.1 – Activitatea de autentificare

7.1 Activitatea de autentificare

Pentru a se autentifica în aplicație utilizatorul va trebui să furnizeze un email și o parola asociate anterior în baza de date.

Dacă combinația de email și parola nu se regăsește va fi afișat mesajul tip Toast



Dacă nici un câmp nu este completat se va afișa mesajul de atenționare

Dacă utilizatorul nu are cont poate apela la butonul REGISTER care îl va trimite în pagina de înregistrare.

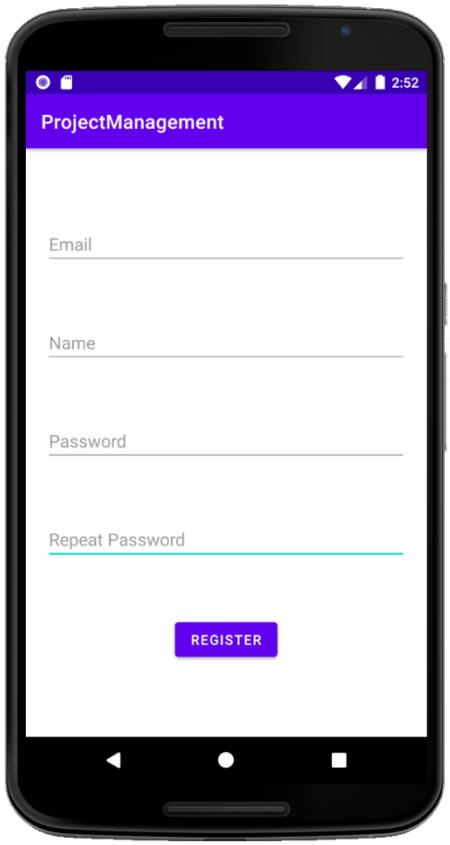


Figura 7.2.1 – Activitatea de înregistrare

7.2 Activitatea de înregistrare

În activitatea de înregistrare este necesară furnizarea unui email, nume, parola și repetare a parolei pentru asigurarea corectitudinii introducerii parolei.

Butonul REGISTER va verifica prezența textului în toate câmpurile, va verifica existența precedentă a emailului în baza de date și va verifica dacă cele doua parole sunt identice.

După efectuarea verificărilor, dacă contul a fost creat, utilizatorul va fi autentificat și trimis în activitatea principală, la fel ca în cazul butonului LOGIN de pe activitatea de autentificare.

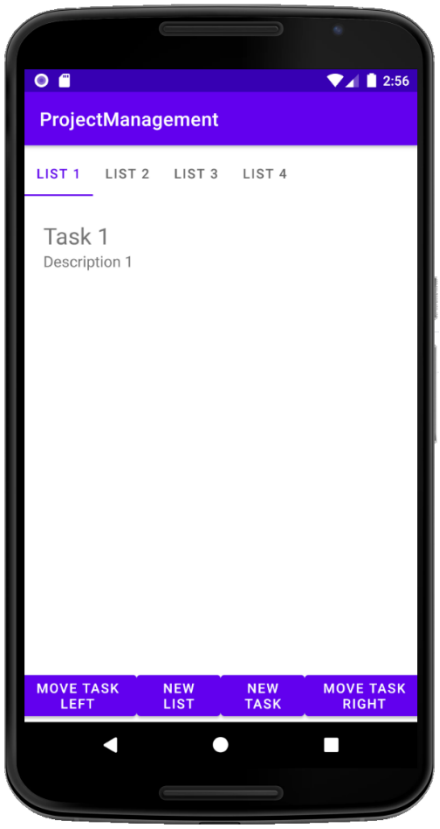
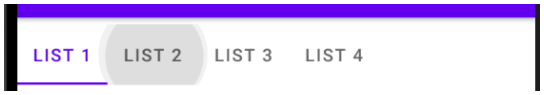


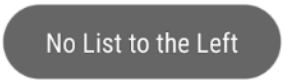
Figura 7.3.1 – Activitatea principala

7.3 Activitatea principala

În activitatea principală utilizatorul poate naviga între Task-uri și liste de Task-uri prin mișcări de tip swipe verticale sau orizontale, poate crea liste sau taskuri noi prin apăsarea butoanelor corespunzătoare din partea de jos sau poate edita listele de Task-uri sau Task-urile prin apăsarea prelungită pe ele pentru a deschide fragmentul de editare.



De asemenea se pot muta Task-urile între liste folosind butoanele MOVE TASK



7.4 Navigarea

Navigarea între fragmentele listelor cu RecyclerView din interiorul ViewPager2 se poate fie prin mișcări tip swipe spre stânga sau spre dreapta pe suprafața ecranului fie prin atingerea Tab-ului din TabView corespunzător paginii către care se dorește navigarea.

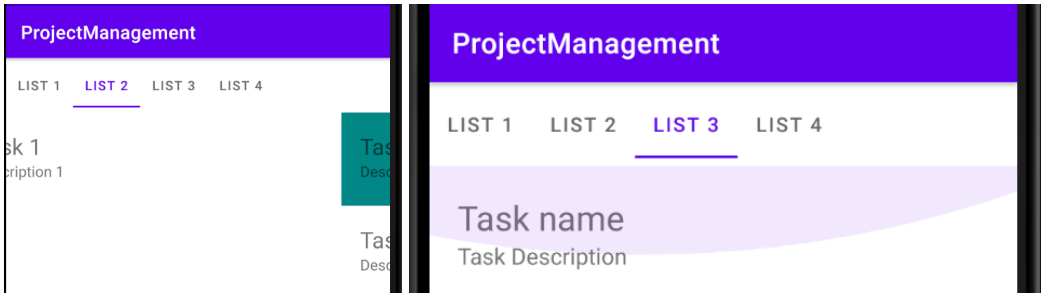


Figura 7.4.1 – Navigarea cu atingeri de tip swipe

Navigarea sus-jos în lista se poate realiza prin același tip de mișcări swipe doar că pe direcție verticală.

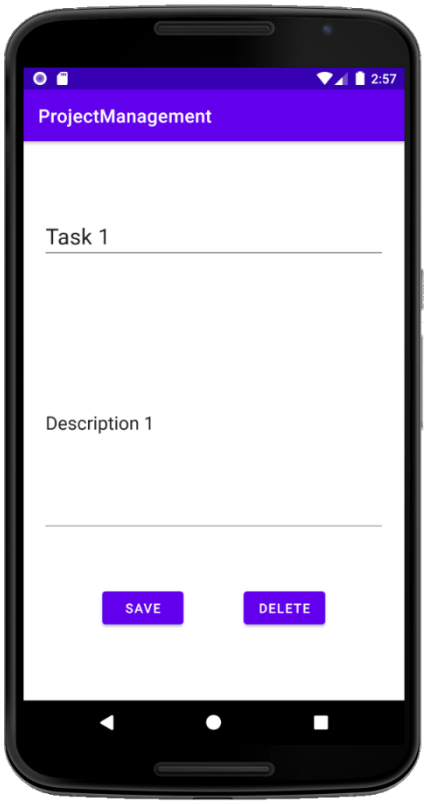


Figura 7.5.1 – Activitatea de editare a Task-urilor

7.5 Operații cu Task-uri

Pe lângă operațiile de mutat Task-uri între liste și butonul de creare a Task-urilor noi din activitatea principală. În fragmentul dedicat editării Task-urilor pe care îl putem deschide prin apăsare prelungită pe un Task avem posibilitatea editării câmpurilor de text componente, iar apoi salvarea modificărilor sau ștergerea în întregime a Task-ului corespunzător.

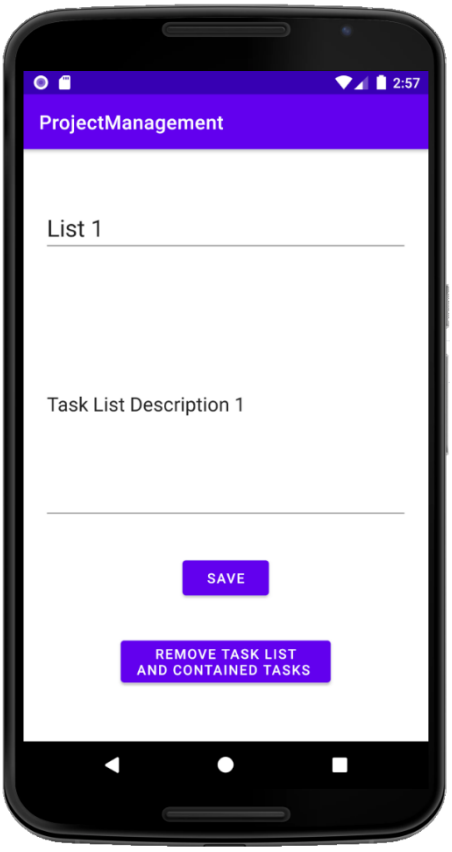


Figura 7.6.1 – Activitatea de editare a listelor de Task-uri

7.6 Operații cu liste de Task-uri

Singura operație cu liste de Task-uri disponibilă din activitatea principală este crearea unei liste noi. Aceasta va fi creata în locul listei selectate în TabView, iar listele de la dreapta ei vor fi mutate o poziție mai spre dreapta. În afară de butonul de ștergere a listelor de Task-uri, celelalte funcții ale fragmentului de editare a listelor de Task-uri sunt similare cu fragmentul de editare al Task-urilor. Diferența dintre butoanele de ștergere este că cel de pe pagina listelor șterge și toate Task-urile corespunzătoare listei selectate și muta ID-ul tuturor listelor și Task-urilor de la dreapta spre stânga.

8 Testarea aplicației

La etapa testării am folosit ALM (Application Lifecycle Management) pentru a crea câteva cazuri de testare de bază pentru a organiza testarea aplicației.

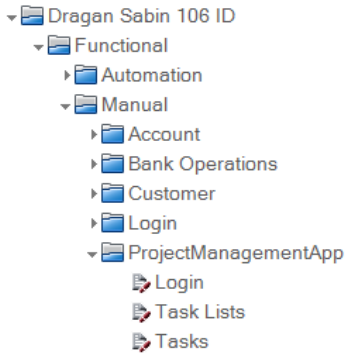
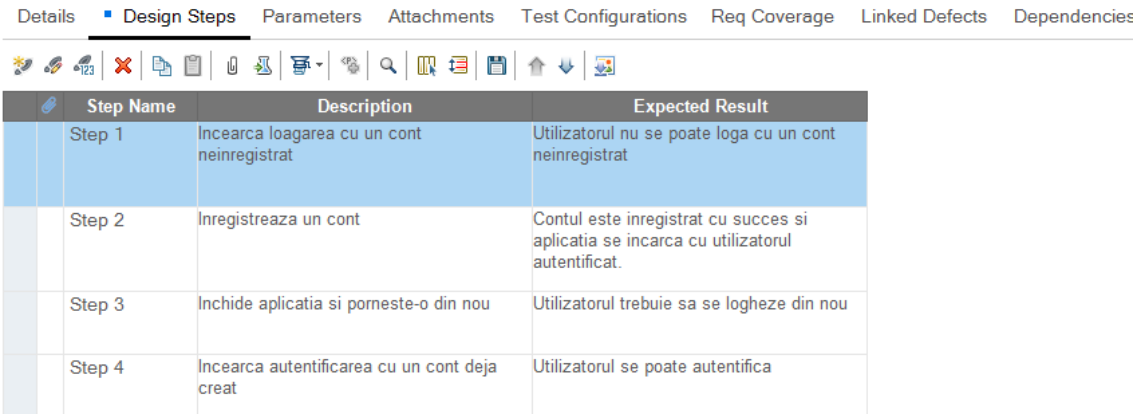
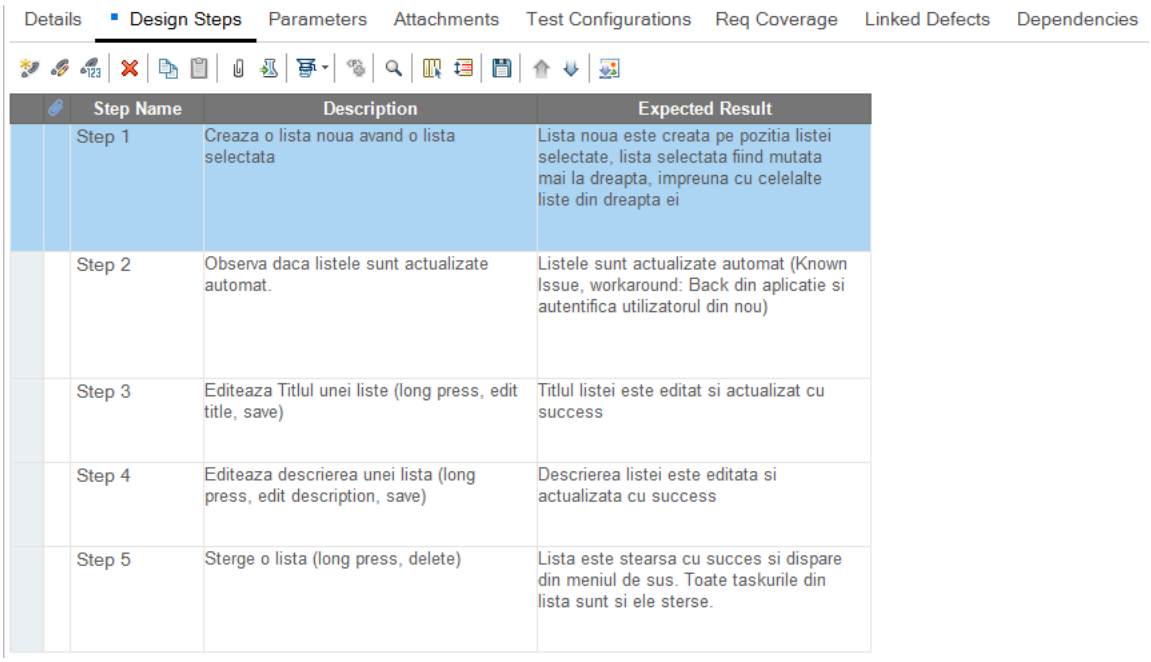


Figura 8.1 – Arborele de directoare al ALM



Step Name	Description	Expected Result
Step 1	Incearca loagarea cu un cont neinregistrat	Utilizatorul nu se poate loga cu un cont neinregistrat
Step 2	Inregistreaza un cont	Contul este inregistrat cu succes si aplicatia se incarca cu utilizatorul autentificat.
Step 3	Inchide aplicatia si porneste-o din nou	Utilizatorul trebuie sa se logheze din nou
Step 4	Incearca autentificarea cu un cont deja creat	Utilizatorul se poate autentifica

Figura 8.2 – Setul de teste pentru autentificare



Step Name	Description	Expected Result
Step 1	Creaza o lista noua avand o lista selectata	Lista noua este creata pe pozitia listei selectate, lista selectata fiind mutata mai la dreapta, impreuna cu celelalte liste din dreapta ei
Step 2	Observa daca listele sunt actualizate automat.	Listele sunt actualizate automat (Known Issue, workaround: Back din aplicatie si autentifica utilizatorul din nou)
Step 3	Editeaza Titlul unei liste (long press, edit title, save)	Titlul listei este editat si actualizat cu success
Step 4	Editeaza descrierea unei lista (long press, edit description, save)	Descrierea listei este editata si actualizata cu success
Step 5	Sterge o lista (long press, delete)	Lista este stearsa cu succes si dispare din meniul de sus. Toate taskurile din lista sunt si ele sterse.

Figura 8.3 - Setul de teste pentru liste de Task-uri

Details | Design Steps | Parameters | Attachments | Test Configurations | Req Coverage | Linked Defects | Dependencies

	Step Name	Description	Expected Result
	Step 1	Selecteaza o lista si creaza un task nou	Task-ul nou este creat in lista selectata
	Step 2	Observa daca elementele listei sunt actualizate automat.	Elementele din liste sunt actualizate automat (Known Issue, workaround: Back din aplicatie si autentifica utilizatorul din nou)
	Step 3	Edtiteaza titlul unui task (long press pe task, edit title, save)	Titlul taskului este schimbat
	Step 4	Editeaza descrierea unui task (long press pe task, edit title, save)	Descrierea taskului este schimbata in lista
	Step 5	Populeaza lista cu minim 4 taskuri	4 Taskuri sunt vizibile in lista unul sub altul
	Step 6	Sterge un task (long press, delete)	Taskul a disparut din lista, celealte Task-uri sunt neafectate
	Step 7	Muta un task spre stanga	Taskul este mutat in urmatoarea lista din stanga, daca nu exista nicio lista in stanga, el ramanane pe loc.
	Step 8	Muta un task spre dreapta	Taskul este mutat in urmatoarea lista din dreapta, daca nu exista alta lista in dreapta taskul ramane pe loc
	Step 9	Muta un task (stanga, dreapta) intre doua liste care aveau inainte alta lista intre ele, aceasta fiind stearsa intre timp. (ex: List 1, List 2, List 3; Delete List 2, move task from List 1 to List 3 and Back)	Taskul este mutat cu succes in ambele directii peste lista care a fost stearsa.

Figura 8.4 - Setul de teste pentru Task-uri

IDE-ul Android studio permite testarea simultană pe mai multe telefoane fizice conectate pe USB la calculator dar și pe mai multe telefoane virtuale create în emulatorul Android cu specificații hardware și software personalizabile.

9 Concluzii

În urma încercărilor și a experienței nou dobândite pe parcursul acestui proiect pot concluziona că aplicația ar ajuta într-o măsura considerabilă la ameliorarea problemelor care au dus la ideea inițiala a creării ei. Posibilitatea de a urmări cu ușurință orice proiect de mică sau de mare anvergura, de pe dispozitivul aflat întotdeauna în buzunar și completarea informațiilor despre ele la fel de ușor va face munca în domeniile care necesită lucrul pe proiecte mult mai ușoară.

Dezvoltarea în Android Studio este extrem de facilă, IDE-ul având funcții de auto-completare a codului avansate care găsesc toate metodele posibile ale claselor instantaneu. Astfel un programator familiarizat cu mediul Android poate completa un proiect complex într-o perioadă de timp foarte scurtă. IDE-ul oferă de asemenea funcții de build foarte rapid cu Gradle și oferă suport programatorului încă de la început până la lansarea aplicației cu emulatorul android bazat pe QEMU care permite verificarea instantanee a schimbărilor de cod pe un dispozitiv virtual fără a necesita măcar reconstruirea aplicației și reinstalarea pe telefon.

Cum nici o aplicație nu este niciodată complet terminată nici aceasta încă nu este fiindcă încă se mai pot găsi lucruri de adăugat și module de creat pentru ea. Una din funcțiile mai importante care lipsesc este partajarea proiectelor între mai mulți utilizatori, așadar implementarea modulelor de conectare la rețea, de gestionare a drepturilor utilizatorilor și dezvoltarea serverului care ar gestiona datele tuturor utilizatorilor ar putea fi etape următoare în dezvoltarea acestei aplicații. Pe lângă adăugarea acestor funcții aplicația ar mai putea fi îmbunătățita și pe plan vizual prin adăugarea de animații și tranziții moderne, regândirea paletei de culori și îmbunătățirea utilizabilității. Pentru aceasta se pot implementa redimensionări dinamice în funcție de dimensiunea și orientarea ecranului dispozitivului.

10 Bibliography

1. **Darwin, Ian F. 2012.** *Android Cookbook.* s.l. : O'REILLY, 2012.
2. **Lee, Wei-Meng. 2013.** *Android application development cookbook 93 recipes for building winning apps.* s.l. : Wrox, 2013.
3. **Rick Rogers, John Lombardo, Zigurd Mednieks & Blake Meike. 2009.** *Android Application Development.* s.l. : OReilly, 2009.
4. **Stack Overflow .** *Android™ Notes for Professionals.* s.l. : Creative Commons BY-SA.

5. **To, James Steele Nelson.** *The Android Developer's Cookbook Building Applications with the Android SDK.* s.l. : Addison-Wesley.
6. [Online] [Citat: 06 14, 2021.] https://www.statista.com/statistics/245501/multiple-mobile-device-ownership-worldwide/.
7. [Online] [Citat: 06 14, 2021.] https://developer.android.com/guide/platform#native-libs.
8. [Online] [Citat: 06 14, 2021.] https://www.geeksforgeeks.org/android-architecture/.
9. [Online] [Citat: 06 14, 2021.] https://source.android.com/devices/architecture.
10. [Online] [Citat: 06 14, 2021.] https://theiconic.tech/android-java-fdbd55aadc51.
11. [Online] [Citat: 06 14, 2021.] https://developer.android.com/guide/platform#native-libs.
12. [Online] [Citat: 06 14, 2021.] https://www.sqlite.org/about.html.
13. [Online] [Citat: 06 14, 2021.] https://www.guru99.com/recyclerview-android-list.html.
14. [Online] [Citat: 06 14, 2021.] https://guides.codepath.com/android/using-the-recyclerview.
15. [Online] [Citat: 06 14, 2021.] https://www.programmersought.com/article/95884244584/.
16. [Online] [Citat: 06 14, 2021.] https://www.section.io/engineering-education/android-viewpager2/.
17. [Online] [Citat: 06 14, 2021.] https://www.tutorialspoint.com/android/android_fragments.htm.
18. [Online] [Citat: 06 14, 2021.] https://developer.android.com/guide/fragments .
19. [Online] [Citat: 06 14, 2021.] https://medium.com/swlh/what-are-the-various-phases-of-mobile-app-development-4f0a1748e619.
20. [Online] [Citat: 06 14, 2021.] http://git-scm.com/about.
21. [Online] [Citat: 06 14, 2021.] https://en.wikipedia.org/wiki/GitHub.
22. [Online] [Citat: 06 14, 2021.] https://developer.android.com/studio/intro.
23. [Online] [Citat: 06 14, 2021.] https://developer.android.com/studio/.
24. [Online] [Citat: 06 14, 2021.] https://docs.gradle.org/current/userguide/userguide.html.
25. [Online] [Citat: 06 14, 2021.] https://en.wikipedia.org/wiki/Gradle.
26. [Online] [Citat: 06 14, 2021.] https://theqalead.com/topics/what-is-an-android-emulator/.