UNIVERSITATEA DIN BUCURESTI

**FACULTATEA DE MATEMATICA SI INFORMATICA**

**SPECIALIZAREA INFORMATICA**

**Dezvoltarea unei aplicații mobile pentru manipularea documentelor de tip text**

***Savu Liviu-Gabriel***

***Avram Adrian-Constantin***

DECLARATIE PRIVIND ORIGINALITATE SI RESPECTAREA

DREPTURILOR DE AUTOR

Prin prezenta declar ca proiectul cu titlul „Dezvoltarea unei aplicatii mobile pentru manipularea documentelor de tip text” este scrisa de echipa formata din Avram Adrian-Constantin si Savu Liviu - Gabriel. De asemenea, declar ca toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate in lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar si în traducere proprie din alta limba, sunt scrise între ghilimele și deţin referinta precisa a sursei;
* reformularea in cuvinte proprii a textelor scrise de catre alti autori detine referinta precisa;
* codul sursa, imaginile etc. preluate din proiecte *open*-*source* sau alte surse sunt utilizate cu respectarea drepturilor de autor si detin referinte precise;
* rezumarea ideilor altor autori precizeaza referinta precisa la textul original.

# Introducere si motivatie

Aplicatia are ca scop salvarea de documente text. Aceasta aplicatie ofera aspecte precum creearea de noi documente, stergerea de documente si salvarea lor.  
Documentele sunt salvate in telefonul clientului.In creearea documentelor exista posibilitatea de a scrie sau de a vorbi pentru a produce text (speech to text). Feature-ul de speech to text este facut de catre API-ul oferit de Android, noi nu avem nici-un control asupra lui. Atunci cand utilizatorul doreste(prin apasarea unui buton) sa vorbeasca si sa produca text noi chemam API-ul de la android, iar android ia controlul asupra aplicatiei si la sfarsit noua ne intoarce textul. Unul dintre principalele avantaje al acestei aplicatii este managementul datelor, astfel clientul are posibilitate de accesare a datelor in modul offline, fara a utiliza conexiunea la internet.

Popularitatea acestui tip aplicatii a crescut enorm in ultima vreme, cresterea fiind sustinuta atat de dorinta utilizatorului de a comunica mai usor și mai ieftin dar si de investitiile dezvoltatorilor importanti de pe piata. Unele estimari arata ca aplicatiile de mesagerie au intrecut (ca numar de utilizatori activi lunar) chiar aplicatiile retelelor sociale1. Alti factori importanti sunt faptul ca acest tip de aplicatie ofera utilizatorului posibilitatea de a comunica cu alți utilizatori aflati chiar în alte tari fara a plati taxe suplimentare ca în cazul convorbirilor traditionale, aceste aplicatii utilizand internetul ca suport și disponibilitatea internetului mobil este în continua crestere (retelele wireless care ofera internet gratuit au o disponibilitate foarte mare acum).

Pentru a ajunge la un numar cat mai mare de utilizatori de dispozitive mobile am hotarat implementarea clientului pe platforma Android intrucat statisticile arată ca Android detine peste 70% din piața dispozitivelor mobile2.

Avand astfel o idee despre popularitatea acestui tip de aplicații este de așteptat ca in eventualitatea intrarii pe piata cu o astfel de aplicație aceasta trebuie sa respecte anumite cerinte ca sa poata fi competitiva:

* utilizarea aplicatiei sa fie intuitiva;
* sa accelereze procesul de operare asupra documentelor;
* sa nu utilizeze prea multe resurse (resurse energetice limitate);
* sa fie stabila in condiții de incarcare mare.

O serie de atribute ce nu sunt atat de transparente pentru utilizatorul de aplicatii , insa sunt imperative pentru buna functionare a acesteia sunt3 :

1. Disponibilitatea – posibilitatea de a accesa aplicatia mobila in mod continuu , fără intreruperi.
2. Securitatea – riscul minim de a avea loc o pierdere de informatii confidentiale sau personale.
3. Optimizarea – capacitatea aplicatiei de a aloca resurse in mod inteligent
4. Scalabilitatea- functionarea fără probleme a aplicației mobile chiar si atunci când volumul de date pe care le prelucrează devine mai mare.

Dezvoltarea aplicației in faza incipienta s-a realizat in editorul Visual Studio. Observand ca modul de lucru si timpii alocati pentru teste erau ineficienti, ne-am mutat pe platforma IntelliJ. Dupa mai multa cercetare am analizat si concluzionat faptul ca pentru o aplicatie de tipul careia noi dezvoltam cel mai util editor este Android Studio, deoarece este extrem de popular pentru dezvoltarea aplicatiilor java furnizand multe aspecte ce ii usureaza munca programatorului in programarea , depanarea si testarea programelor prin uneltele ajutatoare oferite de acest IDE. Editorul Android Studio este cea mai utilizata platforma de dezvoltare software pentru aplicatiile mobile ce se regasesc in Magazin Play. Aplicatia “Magazin Play” este una dintre cele mai folosite aplicatii mobile in care se regaseste continut digital securizat si de incredere.

Clasele principale ce definesc layout-urile aplicatiei se regasesc in folderul activity: “HistoryActivity”, “MainActivity” si “NoteActivity”. Restul sunt clase satelit ale acestora , oferind suport in construirea claselor principale . Acestea se afla in folderul adapter: “NoteListAdapter”; entity: “Note”; helper: clasa abstracta “LazyLoader”, clasa “NotesManager” si interfata “Response listener” si folder-ul helper, ce contine clasa dedicata testelor aplicatiei: “UnitTest”.

Verificarile componentelor aplicației s-au realizat in prima faza cu ajutorul framework-ului JUnit5; dar intrucat in cadrul proiectului la unele testari am avut anumite erori de sistem pe care nu le-am putut rezolva ne-am decis sa utilizam o versiune mai veche, JUnit 4. Controlul si calitatea acestei versiuni nu strica sau afecteaza calitatea serviciilor de teste pe care programul le furnizeaza, oferind in acelas timp si o eficientizare in spatiul de cod scris in cadrul clasei de testari.

Layout-urile specifice fiecarei clase ce defineste o fereastra sunt in format xml. Imaginile folosite in constructia paginilor sunt in format png si sunt salvate in folder-ul “drawable”. De asemenea, aplicatia a fost testata manual pentru a experimenta utilizarea acesteia cu ajutorul emulatorului Pixel2 API R.

Aplicatia a fost instantiata cu o activity de tip empty, ce continea doar bara de actiuni si un buton de intoarcere catre fereastra principala. Aplicatia a fost dezvoltata astfel incat sa fie compatibila pe toate telefoanele ce au un android mai nou de versiunea 4.1 (Jelly Bean). Conform unor date statistice, aplicatia ar fi compatibile pentru 99.8% dintre telefoanele ce indeplinesc acest criteriu.

Pentru realizarea acestei aplicatii programatorii s-au consultat si schimbat idei prin intermediul retelelor sociale. Codul a fost actualizat si imbunatatit de fiecare programator din echipa pe platforma github, furnizand commit-uri in care se specifica fiecare modificare adusa codului pe intregul proces de faurire al aplicatiei.

Aplicatia este integrata in doua mari folderele, cel de dezvoltare al aplicatiei “app” si cel in care se regaseste instrumentul de construire al aplicatiei “Gradle Scripts”. De-a lungul imbunatatirii aplicatiei, am intampinat erori de incompatibilitate intre functiile folosite in program si configuratiile Android-ului , asa ca anumite date din folderul “Gradle Scripts” au fost modificate . Variabila “testImplementation” defineste versiunea programului de tes, aceasta fiind JUnit 5.3.1. Din cauza unor erori aparute in sistem , am putut folosi doar anotatii din JUnit 4 , pentru a rezolva anumite probleme aparute .

androidTestImplementation "org.junit.jupiter:junit-jupiter-api:5.3.1"  
androidTestImplementation "de.mannodermaus.junit5:android-instrumentation-test:0.2.2"  
testImplementation "org.junit.jupiter:junit-jupiter-api:5.3.1"  
testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine:5.3.1"  
testImplementation "org.junit.jupiter:junit-jupiter-params:5.3.1"

**Secventa de cod 1: Implementarea versiunii de JUnit in tool-ul folosit in cadrul proiectului**

De asemenea, a fost modificata si versiunea gradle fata de cea originala. S-a instalat si s-a folosit pe parcursul dezvoltarii aplicatiei versiunea de gradle 3.6.2, deoarece multe clase pe care doream sa le importam in program nu erau recunoscute de compilator

In folderul „app” se prezinta intreg codul aplicatiei. In primul modul „manifests” se regasesc detaliile generale despre aplicatie: icon-ul aplicatiei ce se vede in telefon „ic\_launcher” , care se gaseste in folderul „res” dedicat interfetei aplicatiei , in modulul „mipmap” . Labelul „Note” defineste titlul aplicatiei ce se observa in interiorul telefonului. Tema aplicatiei este „AppTheme” si se infatiseaza in culorile albastru si alb. Ulterior, se specifica activitatile prezente in aplicatie: „MainActivity”, „HistoryActivity” si „NoteActivity”; „MainActivity” fiind cea care se declanseaza la deschiderea aplicatiei .

In modulul „java” din „app” se regaseste codul efectiv al aplicatiei, scris in java. In submodulul „activity” se afla cele trei activitati, specificate mai sus. In interiorul clasei din submodulul „adapter” se instantiaza fiecare chenar cu datele introductive despre fiecare notita. Clasa “NoteListAdapter” este prezenta in clasa „HistoryActivity” si intercomunica cu aceasta prin intermediul variabilelor „historyListView” si „adapter” . Obiectul „adapter” se prezinta drept un adaptor pentru clasa „HistoryActivity” si prin intermediul acestuia interfata „history\_activity” este populata cu layout-ul „adapter\_note\_list\_layout”, dedicat construiri chenarelor destinate notitelor. In submodulul „entity” se afla clasa „Note”, folosita pentru a salva datele din fisier in program, urmand sa fie manipulate iar apoi rescrise in fisierele sursa. Submodulul „helper” detine o clasa abstracta „LazyLoader”, dedicata eficientizarii memoriei folosite pentru incarcarea blocurilor de memorie, clasa „NotesManager” folosita pentru manipularea fisierelor si interfata ResponseListener ce deserveste drept scop incarcarea si afisarea in cadrul interfetei a chenarelor distribuite pe ecran. In cadrul modulului dedicat testelor, in submodulul “Helper” se gaseste clasa dedicata testarilor si verificarilor de cod al aplicatiei.

In modulul “java(generated)” se regaseste clasa “BuildConfig” ce contine datele generate automat de editor, ce ajuta la identificarea si manipularea mai usoara a codului prin intermediul instrumentelor oferite de acest editor .

Folderul “res” contine toata partea de grafica a aplicatiei .

Submodulul “drawable” al folder-ului dedicat interfetei grafice are in componenta imagini atat care sunt prezente in interfata , cat si unele folosite de test sau la care s-au renuntat a fi aplicate pe partea grafica .

Submodulul “layout” contine toate schemele prezente in aplicatie . “activity\_main” este layout-ul pentru fereastra principala ;iar “activity\_note” si “activity\_history” sunt layout-urile pentru ferestrele secundare . “adapter\_note\_list\_layout” contine interfata pentru fiecare chenar dedicat notitelor . In componenta arborelui de elemente , chenarul se prezinta ca un sir de elemente impachetate intr-o entitate view de tipul “MaterialCardView” . Ulterior , chenarul este incapsulat pe orizontala , respectiv pe verticala pe 2 nivele de tipul “LinearLayout” . Inauntru se afla doua variabile de tipul “TextView” ce afiseaza datele introductive despre notita . Ele sunt instantiate initial cu valoarea inregistrata ca string “loading” si se regaseste in xml-ul destinat retinerii string-urilor , “strings.xml” .

Realizarea interfetelor s-a facut prin constrangerea mai multor layout-uri arhitecturizate de programator . Pentru interfata “activity\_history” s-a facut o constrangere la layout-ul ce contine bara de instrumente a ferestrei ; care la randul ei , fereastra se contrange de layout-ul content\_history , unde obiectul “ListView” incapsuleaza toate chenarele ce contin notitele clientului .

Fereastra “activity\_note” are o structura similara cu cea a ferestrei “activity\_history” , formata prin constrangeri intre mai multe layout-uri . La fel ca la “activity\_history” , layout-ul principal are legatura la layout-ul ce contine bara de instrumente “app\_bar\_history”. La randul sau , noul layout se constrange de layout-ul dedicat continutului ferestrei “content\_note” ce contine campurile de completare pentru titlul si continutul notitei ce urmeaza a fi salvata sau nu de catre utilizator.

Interfata “activity\_main” este formata doar dintr-un singur bloc . Aceasta contine un text prietenos si plin de entuziasm ce reflecta ideea aplicatiei “Let’s take a note!” , irmat de doua butoane ce redirectioneaza clientul catre celelalte ferestre din aplicatie . Acestea sunt de culoare albastra avand scrisul pe un fond alb . Toate elementele prezentate sunt situate central-orizontal , in partea vertical-superioara a ferestrei.

In folder-ul urmator “menu” se regasesc doua meniuri ce sunt populate in bara de instrumente a aplicatiei .

In primul fisier XML “menu” se gaseste bara de meniu pentru fereastra “activity\_note” , in care se infatiseaza butonul de salvare al unui fisier si titlul continutului afisat .

In al doilea fisier XML “menu2” se afla bara de meniu pentru fereastra “activity\_history” , unde de asemenea se prezinta titlul continutului afisat , precum si butonul prin care se efectueaza o stergere a unui fisier . In cazul in care utilizatorul va dori sa stearga un fisier ,la momentul apasarii pe butonul “DELETE A FILE” ii va aparea o fereastra pop-up in care ii se va cere sa introduca titlul fisierului pe care il doreste sa-l stearga . Daca operatiunea a avut loc cu succes , fereastra se va inchide si ii va aparea un mesaj temporar in care se va specifica daca stergerea a avut loc cu succes sau nu .

Modulul “mipmap” contine in primul modul “ic\_launcher” iconitele afisate in aplicatie sub diferite rezolutii : hdpi,mdpi,xhdpi,xxhdpi,xxxhdpi si sub diferite formate: .png , .xml in forma lor naturala . In telefonul utilizatorului , iconita de lansare a aplicatiei va fi in forma imaginilor din al doilea modul “ic\_launcher\_round” , conturate intr-o forma rotunda.

Modulul “values” contine variabilele utilizate in construirea elementelor ferestrelor .

Primul fisier “colors” de tip XML are in componenta tag-urile pentru culorile folosite in constructia deisgn-ului aplicatiei . Salvarea culorilor si a altor proprietati pe care i le atribuim elementelor din interfata este foarte utila in momentul in care avem multe elemente asemanatoare si printr-o singura modificare dorim sa facem schimbarea pe toate elementele .

Al doilea fisier “strings.xml” contine toate textele destinate vizualizarii clientului . Din nou , acest lucru este folositor in momentul in care dorim sa modificam sau sa stergem textul incapsulat pe fiecare element ce contine textul respectiv.

In al treilea fisier “styles.xml” se definesc configuratiile asupra stilului de design al ferestrelor din aplicatie . S-a ales o tema care sa nu includa din fabrica bara de instrumente , ci sa fie ulterior creata si modelata de programatori . Culorile elementelor din ferestre si culoarea scrisului folosita se asorteaza intr-o maniera eleganta de nuante de albastru si alb , oferind o senzatie ambiental-placuta clientului la folosinta aplicatiei .

În capitolele ce urmează am prezentat modalitățile de implementare identificate de mine a căror utilizare duce la îndeplinirea criteriilor enumerate.

1. Popularitatea aplicațiilor de mesagerie <https://www.businessinsider.com/the-messaging-app-report-2015-11>
2. Popularitatea platformei Android - <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1>
3. Caracteristicile unei aplicatii competitive - <https://www.aplicatii-mobile.com/>

# Contribuții

În procesul de dezvoltare al clientului am analizat diverse tehnici pentru a minimiza amprenta aplicației asupra resurselor clientului dar fără să afecteze negativ calitatea procesului de comunicare al utilizatorului . Astfel ca dupa niste studii efectuate s-au ales urmatoarele metode considerate benefice:

Utilizarea librariei JUnit4 , API-ul folosit pentru intercomunicarea dintre programator si platforma de testare a metodelor dispuse pentru prelucrarea si dezvoltarea aplicatiei. In cadrul unitatii de test am folosit anotatii pentru identificare si rolul fiecarei metode : @Before si @Test.

Utilizarea ferestrelor pop-up pentru a atrage atentia clientului asupra modfiicarilor ce pot fi facute

Încărcarea asincronă a anumitor părți necesare interfeței grafice.

Incarcarea sincronizata cu actiunile utilizatorului a unor detalii inregistrate de utilizator pe care acesta doreste sa le acceseze , oferind o interfata libera si usor de folosit .

Preluarea unor tehnici din libraria android , care ofera utilizatorului un ambient placut atunci cand foloseste aplicatia.

Folosirea unor tool-uri recomandate pentru testarea aplicatiei precum Maven , pentru a spori productivitatea testelor si pentru a garanta ca aplicatia este eficienta si are o manevrabilitate fara probleme .

Contents

[Introducere si motivatie 3](#_Toc42527049)

[Contribuții 9](#_Toc42527050)

[Capitolul 1: Dezvoltarea aplicatiei client pe platforma Android 10](#_Toc42527051)

[1.1. Arhitectura generala a aplicatiei 11](#_Toc42527052)

[1.2. Interfata grafica 12](#_Toc42527053)

[1.3. Prezentarea Ferestrelor 13](#_Toc42527054)

[1.3.1. Fereastra principală 13](#_Toc42527055)

[1.3.2 Fereastra de istoric 15](#_Toc42527056)

[1.3.3 Fereastra de manipulare a notitelor 23](#_Toc42527057)

[1.4. Cadrul de testare JUnit 4 28](#_Toc42527058)

[1.5. Concluzii 32](#_Toc42527059)

[1.6. Bibliografie 33](#_Toc42527060)

# 

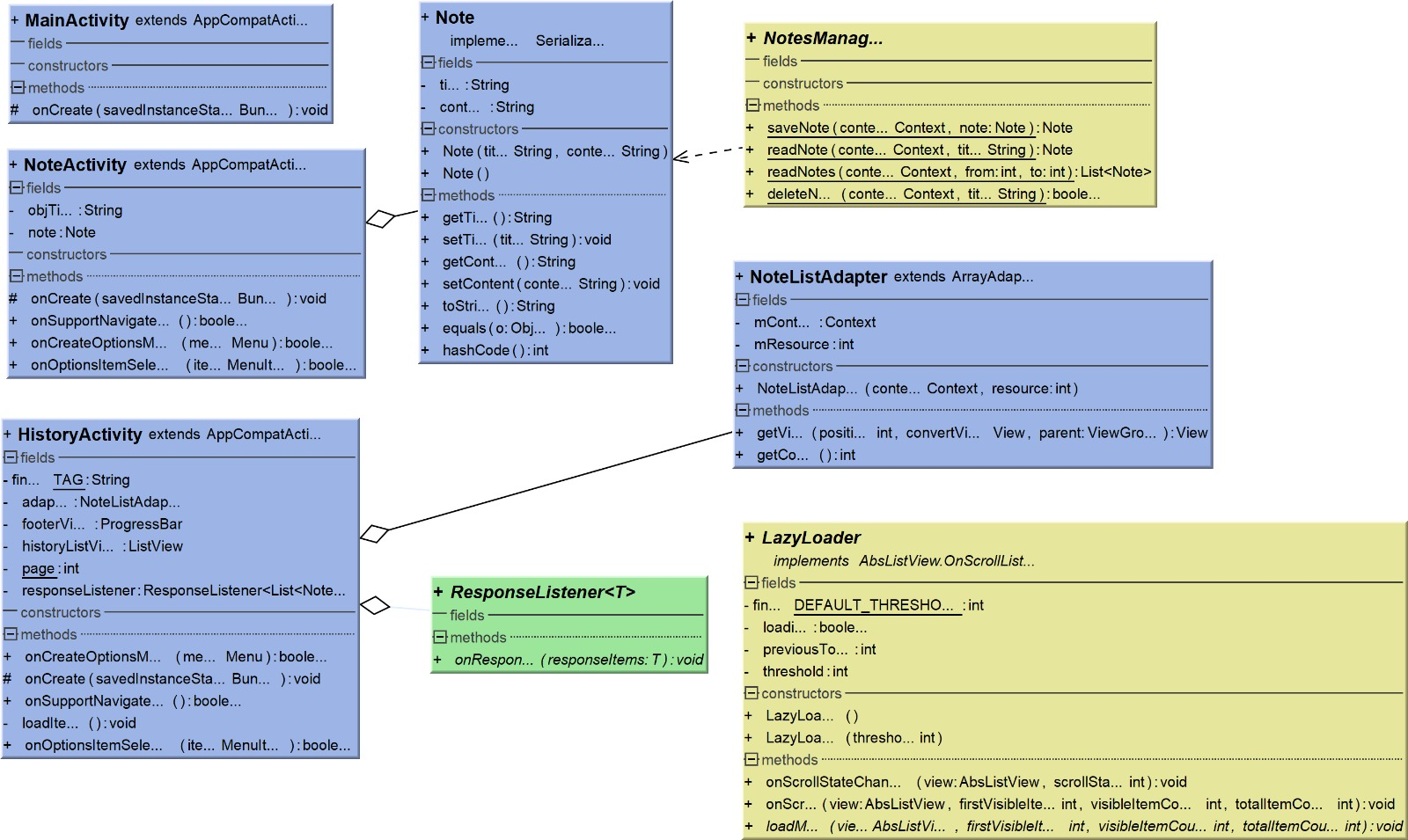
# Capitolul 1: Dezvoltarea aplicatiei client pe platforma Android

Aplicația client are ca scop facilitarea procesului de crearea de noi documente , stergerea de documente si salvarea lor . Această aplicație a fost dezvoltată cu ajutorul mediului de dezvoltare integrat pentru platforma Android – Android Studio. Aplicatia a fost creata si dezvoltata pe limbrajul de programare java , folosindu-ne pentru platforma de teste de unelte de management precum Maven. Pe parcursul dezvoltării aplicației am urmărit utilizarea unor tehnici specifice, descrise pe larg în subcapitolele următoare, astfel încât aplicația finală să poată fi utilizata în condiții normale iar consumul de resurse să fie redus.

## 1.1. Arhitectura generala a aplicatiei

Arhitectura unei aplicații de acest tip este diferită de cea a unei aplicații obișnuite. Aplicația client trebuie să fie capabilă să ofere utilizatorului o experiență plăcută la utilizare, să poata utiliza cu usurinta functiile aplicatiei si , fiindcă o astfel de aplicație are de obicei, un timp îndelungat de utilizare , trebuie să folosească cu moderație resursele puse la dispoziție de dispozitivul utilizatorului întrucât este bine-cunoscut faptul că dispozitivele mobile din noile generatii dispun de resurse energetice relativ reduse (vezi Figura 1).

**Figura 1:**



Pentru a îndeplini obiectivele enumerate mai sus am recurs cateva tehnici destul de frecvent utilizate în randul dezvoltatorilor de aplicatii mobile pentru platforma Android si nu numai:

* Utilizarea obiectelor de tip *AppCombatActivity* pentru interfață grafica, deoarece se faciliteaza ca aplicatiile dezvoltate cu versiuni mai noi sa functioneze cu versiuni mai vechi;
* Incarcarea asincrona a anumitor parti din interfață grafica;
* Portarea unor porțiuni de cod din Java în Maven

Astfel am ajuns la arhitectura ilustrata din *Figura 1*:

## 1.2. Interfata grafica

Utilizatorul are la dispoziție o interfață grafică simplă și intuitivă, care utilizează pe cât posibil pictograme sugestive și gesturile utilizatorului pentru a face navigarea în cadrul aplicației cât mai simplă și cursivă.

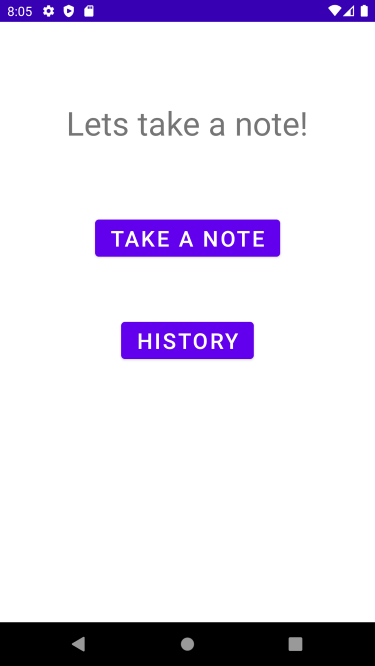
Pentru a păstra consumul de resurse al aplicației client la un nivel cât mai scăzut am recurs la utilizarea metodelor onScroll , pe care le-am suprascris astfel incat sa oferim o interfata libera si placuta la utilizare , evitand incarcarea tuturor datelor odata. Aceasta tehnica se numeste lazy loading si este folosita de majoritatea paginilor web.

Tema folosita in construirea aplicatiei are un aspect mladios , tonurile de alb si albastru combinate sunt deseori folosite in programele care au destinatie prelucrarea textelor si ofera o utilizare atractiva clientului datorita aspectului graficii furnizate. De asemenea , aspectul grafic este unul simplit construit si poate fi manipulat usor si cu placere de catre persoane din orice categorie de varsta , intrucat este o aplicatie folositoare pentru toata lumea.

In cadrul interfetei grafice am folosit dialogul de alerta pentru manipularea unor date pentru a-l face precaut pe utilizator atunci cand doreste sa modifice datele sale salvate.

## 1.3. Prezentarea Ferestrelor

### 1.3.1. Fereastra principală



Fereastra principală (MainActivity) este fereastră de start a aplicației client. Această fereastră permite utilizatorului să-si aleaga ce serviciu sa execute.

În cazul în care utilizatorul se afla deja intr-un alt serviciu si doreste sa acceseze unul nou trebuie sa revina la fereastra principala.Acest lucru se poate realiza prin 2 metode : fie prin apasarea icon-ului specifice comenzii “back” prin intermediul aplicatiei , aflat in bara de actiuni, fie prin comanda programata prin linia de cod getSupportActionBar().setDisplayHomeAsUpEnabled(true); sau prin apasarea butonului din cadrul tastaturii telefonului (operatiune permisa) .

Serviciile furnizate utilizatorului se pot accesa prin intermediul unor butoane albastre,asezate central in pagina , inscriptionate cu un text ce definesc rolul ferestrei catre care te va redirectiona.

Detaliile personale sunt memorate în memoria persistentă a dispozitivului prin intermediul clasei NoteActivity , care se poate accesa din fereastra principala apasand butonul “Take a note” ; aceasta te redirectioneaza catre pagina dedicata clasei. În cazul în care clientul a ajuns intr-o locatie necunoscuta a aplicatiei , acesta poate reveni la pagina principala prin executare operatiunii de tip back.

Fereastra default are un aspect placut , ce te intampina cu un mesaj energic si binevoitor “Let’s take a note!” ; ce defineste o mare parte din obiectivul aplicatiei.

De asemenea , in fereastra principala se regaseste butonul “History” , ce are drept scop

redirectionarea catre activitatea ce furnizeaza afisarea tuturor documentelor salvate de catre client .

In sectiunea de cod 1 am atasat felul in care se redirectioneaza fereastra catre pagina cu istoricul documentelor. Metoda este practicata la fel pentru redirectionarea catre pagina pentru crearea unui nou text.

Variabila historyBtn reprezinta butonul folosit care executa prin intermediul actiunii de “click” functia de redirectionare catre pagina cu istoric (ce apartine de clasa HistoryActivity) din pagina principala(ce tine de clasa MainActivity) cu ajutorul obiectului de tip Intent history , folosit la comunicarea dintre activitati.

historyBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent history = new Intent(MainActivity.this, HistoryActivity.class);  
 MainActivity.this.startActivity(history);  
 }  
});

**Secțiunea de cod 1:Redirectionarea catre pagina cu istoricul documentelor**

takeNoteBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent note = new Intent(MainActivity.this, NoteActivity.class);  
 MainActivity.this.startActivity(note);  
 }  
});

**Secventa de cod 2:Redirectionarea catre pagina cu crearea documentelor**

Procedeul este identic si pentru redirectionarea catre fereastra ce are drept scop crearea de noi documente.

Clasa MainActivity este clasa din spatele interfetei activity\_main , fereastra principala al aplicatiei.Butoanele catre ferestrele de serviciu sunt definite cu id-urile create in interfata principala:

Button takeNoteBtn = findViewById(R.id.*takeNoteBtn*);  
Button historyBtn = findViewById(R.id.*historyBtn*);

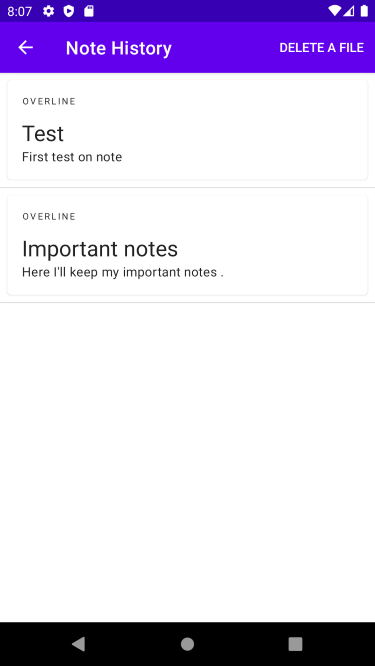
**Secventa de cod 3:Definirea butoanelor in activitatea java cu id-urile create in interfata grafica de tip XML**

Pentru utilizarea metodelor dar si a tipurilor de obiecte definite a trebuit sa import mai multe pasaje din libraria Android si AndroidX:

- androidx.appcompat.app.AppCompatActivity;

- android.content.Intent;  
- android.os.Bundle;  
- android.view.View;  
- android.widget.Button;

### 1.3.2 Fereastra de istoric



Prin intermediul acestei ferestre utilizatorul poate sa vizualizeze documentele sale stocate in telefon si sa-si stearga din lista anumite texte nedorite.

Grafica ferestrei se prezinta in aceeeasi nuanta de albastru si alb ca la pagina principala.Scrisul in cadrul paginii se infatiseaza in nonculori si in detrimentul puterii culorii de fundal.Introducerea textelor sunt stilistic prezentate prin intermediul unei componente de tip CardView , ce ofera o infatisare eleganta a capsulelor de text.

In lista se afiseaza toate mesajele , utilizatorul putand sa acceseze un document apasand pe introducerea acestuia . Pentru a gasi un anumit fisier , clientul se va folosi de functia telefonului “scroll” .

In cadrul barei de action (ActionBar) se afiseaza titlul ferestrei “Note History” , un buton ce te redirectioneaza catre pagina principala si un item cu numele “Delete a file” provenit din inflater-ul menu2 cu ajutorul caruia se afiseaza o fereastra de tip pop-up din cadrul careia se poate sterge fisiere din spatiul de stocare al mobilului .

In telefon o persoana ce utilizeaza aceasta aplicatie poate sa aiba numeroase documente , iar incarcarea tuturor odata poate conduce la delay-uri asupra aplicatiei atunci cand clientul doreste sa execute o comanda . In acest sens , am

ales sa aplic tehnica *Lazy Loading*1 , foarte folosita in crearea de pagini web , pentru a mentine timpi de initializare la un nivel minim . In secventa de cod 4 se observa cum este apelata metoda atunci cand dam scroll pe fereastra si ce operatii se desfasoara la momentul respectiv.Initial , din default , sunt incarcate 3 texte atunci cand accesam activitatea.

Variabila historyListView de tip ListView stocheaza toate fisierele create.Aceasta apeleaza functia setOnScrollListener care genereaza un nou obiect LazyLoader , pentru a incarca eficient un alte blocuri de memorie ce reprezinta texte din lista de documente.Operatia de incarcare de noi documente se face prin intermediul functiei loadItems(Secventa de cod 4) care incarca un numar limitat de 3 noi texte.

historyListView.setOnScrollListener(new LazyLoader() {  
  
 @Override  
 public void loadMore(AbsListView absListView, int firstVisibleItem, int visibleItemCount, int totalItemCount) {  
 loadItems();  
 }  
});

**Secventa de cod 4: Apelarea metodei setOnScrollListener atunci cand executam comanda scroll**

Clasa abstracta “LazyLoader” implementeaza clasa AbsListView.OnScrollListener si are rolul de a incarca eficient blocurile de memorie reprezentate prin chenarele ce afiseaza date introductive despre fiecare notita .

Clasa detine patru variabile : “DEFAULT\_THRESHOLD” de tip static final int definita cu valoarea 3 ce reflecta numarul de pagini ce se incarca la fiecare actiune “scroll” al clientului , variabila booleana “loading” ce urmareste evolutia incarcarilor blocurilor de memorie si verifica daca mai exista blocuri de incarcat , valoarea intreaga “previousTotal” ce tine cont la ce index am ajuns de incarcat notite si variabila int threshold , care in faza incipienta primeste valoarea “threshold” default , dar care poate primi si alte valori din instantierea clasei . Se prezinta doi constructori ; unul parametrizat care asigneaza noua valoare variabilei “threshold” si unul neparametrizat , programatorul decizand in functie de necesitati pe care il va folosi .

Functia suprascrisa “onScroll” se activeaza la fiecare actiune de scroll a clientului . Aceasta are ca parametrii lista de chenare cu notite vizibila in pagina , desemnata prin primul parametru ; indicele primului parametru visibil , variabila “visibleItemCount” ce defineste numarul de blocuri vizibile la momentul respectiv si totalItemCount , ce ilustreaza numarul total de componente .

Procedeul functiei se centreaza pe alternanta variabilei “loading” . In decursul declansarii functiei , aceasta verifica in prima faza daca variabila booleana e true ; astfel se verifica ulterior si daca numarul total de iteme numerotate a crescut , pentru a actualiza lista datelor din aplicatie . In cazul in care numarul de iteme citite prezent este mai mare decat totalul anterior , atunci totalul anterior reprezentat prin variabila “previousTotal” este actualizata cu noua variabila “totalItemCount” .

In cazul in care operatia s-a efectuat cu succes si “loading” a primit valoarea false ; se incearca intrarea in a doua conditie , pentru a incarca mai multe chenare cu notite . Se verifica daca mai am de incarcat ; daca nu mai sunt notite de incarcat , atunci valoarea “loading” ramane neschimbata . De asemenea , se apeleaza functia abstracta ( care urmeaza sa fie folosita in clasa HistoryActivity) si se incarca urmatoarele pagini .

public void onScroll(AbsListView view, int firstVisibleItem,  
 int visibleItemCount, int totalItemCount) {  
  
 if (loading) {  
 if (totalItemCount > previousTotal) {  
 loading = false;  
 previousTotal = totalItemCount;  
 }  
 }  
  
 if (!loading && ((firstVisibleItem + visibleItemCount) >= (totalItemCount - threshold))) {  
 loading = true;  
  
 loadMore(view, firstVisibleItem,  
 visibleItemCount, totalItemCount);  
 }  
}

**Secventa de cod 5 : Incarcarea datelor din notite in fereastra aplicatiei**

In variabila notes se stocheaza un numar de 3 Note (obiecte in care se salveaza fisierele) din baza de date ; procedura efectuata cu success cu ajutorul metodei readNotes din clasa NotesManager, aflata in pachetul helper.Notes-urile salvate sunt indexate de la page-ul curent pana la al 3-lea page dupa cel actuala (..page, page + 3).

Variabila footerView este folosita pentru a arata progresul loading-ului . In cazul nostru , ne afiseaza introducerea textelor pana in extremitatea de jos a ecranului.In cazul in care nu mai sunt notes de afisat , stergem footer-ul din lista history si returnam . Altfel , ne deplasam la index-ul paginii la care ne aflam in acest moment , si preluam datele din baza de date pentru a le afisa pe interfata.

private void loadItems() {  
 List<Note> notes = NotesManager.*readNotes*(HistoryActivity.this, *page*, *page* + 3);  
 if (notes.isEmpty()) {  
 historyListView.removeFooterView(footerView);  
 return;  
 }  
 *page* = *page* + 3;  
 System.*out*.println(notes);  
 responseListener.onResponse(notes);  
}

**Secventa de cod 6: Preluarea si incarcarea de noi documente din baza de date**

In variabila notes se stocheaza un numar de 3 Note (obiecte in care se salveaza fisierele) din baza de date ; procedura efectuata cu success cu ajutorul metodei readNotes din clasa NotesManager, aflata in pachetul helper.Notes-urile salvate sunt indexate de la page-ul curent pana la al 3-lea page dupa cel actuala (..page, page + 3).

Variabila footerView este folosita pentru a arata progresul loading-ului . In cazul nostru , ne afiseaza introducerea textelor pana in extremitatea de jos a ecranului.In cazul in care nu mai sunt notes de afisat , stergem footer-ul din lista history si returnam . Altfel , ne deplasam la index-ul paginii la care ne aflam in acest moment , si preluam datele din baza de date pentru a le afisa pe interfata.

Metoda statica readNotes are 3 parametrii : contextul de unde extragem fisierele , valoarea integra “from” si “output” ce deservesc pozitia primului , respectiv ultimului fisier din care se vor citi datele .

Variabila “notes” va defini lista cu datele notitelor aflate intre pozitiile “from” si “to” ce urmeaza a fi extrase din contextul trimis ca parametru . Initial , lista este vida , fiind instantiata cu tipul “ArrayList”.

Lista de string-uri “allFilesNames” incapsuleaza toata titlurile fisierelor aflate in context . Prin intermediul metodei “fileList” apelata de context se preiau din baza de date titlurile tuturor fisierelor si sunt asignate in lista de string-uri mentionate anterior .

In urmatoarea etapa se testeaza veridicitatea parametrilor “from” si “to” : se verifica ca valoarea pozitiei de inceput a fisierului sa nu fie negativ , respectiv ca valoarea pozitiei finale sa nu depaseasca lungimea listei ce continte titlurile fisierelor . Ulterior se parcurge vectorul de String-uri allFilesNames de la pozitia “from” pana la pozitia “to” si se adauga in vectorul de Note “notes” toate informatiile din fisierele text care ocupa o pozitie ce este cuprinsa intre cele doua extremitati . Gasirea fisierului cu titlul dat ca parametru in cadrul contextului si convertirea acestuia intr-un obiect de tipul “Note” se efectueaza cu ajutorul functiei statice “readNote” .

Dupa ce toate fisierele sunt salvate in variabila notes aceasta este returnata .

static public List<Note> readNotes(@NotNull Context context, int from, int to) {  
 List<Note> notes = new ArrayList<>();  
 String[] allFilesNames = context.fileList();  
 if (from < 0) {  
 from = 0;  
 }  
 if (to >= allFilesNames.length) {  
 to = allFilesNames.length;  
 }  
 for (int i = from; i < to; i++) {  
 notes.add(*readNote*(context, allFilesNames[i]));  
 }  
 return notes;  
}

**Secventa de cod 7 : Citirea si preluarea datelor din fisiere situate pe pozitiile intr-un interval dat.**

Pentru a beneficia de stergerea unui fisier este necesar ca utilizatorul sa apese pe textul situat in partea dreapta din bara de actiuni “DELETE A FILE”. Prin executarea acestei actiuni , clientului i se va afisa o fereastra Pop-up prin care i se cere sa scrie titlul documentului pe care acesta doreste sa-l stearga din telefon . Ideea de a folosi o fereastra Pop-up definita in activitate

printr-un obiect “AlertDialog” provine din motivul de a-i atrage atentia persoanei ce foloseste aplicatia in legatura cu actiunile pe care le va executa asupra datelor personale . Mesajul “Are you sure you want to delete this entry” din cadrul ferestrei Pop-up are ca intentie sa-l faca pe utilizator sa fie precaut asupra actiunii pe care doreste sa o infaptuiasca . Totodata , in cazul in care clientul s-a razgandit si nu mai vrea sa stearga acel document , poate apasa butonul Cancel , care va anula actiunea . Pentru a ilustra mai sugestiv fereastra Pop-up, am introdus icon-ul ic\_dialog\_alert , din libraria drawable. In cazul in care stergerea textului se efectueaza cu success , pe ecran va aparea un mesaj de scurta durata , creat cu ajutorul obiectului de tip Toast , mesajul “The file has been deleted “ . In cazul in care programul a esuat stergerea sau textul care avea titlul respectiv nu exista in baza de date , se va afisa “The files couldn’t be deleted”.

Pentru a se realiza stergerea , programul trebuie sa caute in adapter-ul listei cu documente titlul documentului scris in variabila input de tip EditText care se doreste a fi sters . Ulterior , item-ul historyListView de tipul ListView trebuie sa fie reactualizat , pentru a afisa si in interfata modificarea efectuata. In secventa de cod 5 se observa algoritmul de stergere si actualizare a listei prezente in interfata si a adaptorului conectat cu lista , precum si stergerea din baza de date a note-

ului , procedura efectuata cu ajutorul metodei “deleteNote” din clasa “NotesManager”.

public void onClick(DialogInterface dialog, int which) {  
 if(NotesManager.*deleteNote*(getApplicationContext(),input.getText().toString())==true) {  
  
 for(int i=0;i<=adapter.getCount();++i){  
 if(adapter.getItem(i).getTitle().equals(input.getText().toString()))  
 {  
 adapter.remove(adapter.getItem(i));  
 historyListView.setAdapter(adapter);  
 break;  
 }  
 }  
 Toast.*makeText*(getApplicationContext(), "The file has been deleted", Toast.*LENGTH\_LONG*).show();  
 }  
 else  
 Toast.*makeText*(getApplicationContext(),"The file couldn't be deleted",Toast.*LENGTH\_LONG*).show();  
}

**Secventa de cod 8 : Stergerea unui note si actualizarea listei de istoric**

In prelucrarea acestei activitati , clasa corespondenta acestei activitati s-a folosit si de alte clase cu rol secundar pentru a efectua actiunile puse la dispozitie utilizatorului .

S-a initializat un obiect de tipul NoteListAdapter , clasa aflata in package-ul cu numele adapter ce a deservit rolul de adaptor al listei historyListView . S-a preluat metoda remove pentru a infaptui actiuni de stergere din lista documentelor .

Clasa NotesManager nu are niciun obiect instantiat in clasa HistoryActivity , deoarece metodele folosite prin care gestionam fisierele aflate in baza de date a telefonului se fac prin intermediul explicit al metodelor statice , fiind redundant sa instantiam un obiect de genul.

De asemenea , clasa a avut contact si cu o alta clasa importanta , NoteActivity ; in cadrul posibilitatii aplicatiei de a vedea textul in stare completa si a-l modifica in situatia in care utilizatorul doreste acest lucru. Prin intermediul unui obiect de tip Intent , se realizeaza o conexiune intre clasa actuala si clasa NoteActivity care permite redirectionarea ferestrei din cea actuala in cea destinatie .

Metoda onBackPressed() ne ajuta sa accesam pagina anterioara (in cazul aplicatiei noastre , pagina de start) prin intermediul tastaturii virtuale a telefonului . Aceasta se afla in interiorul metodei suprascrise onSupportNavigateUp .

@Override  
public boolean onSupportNavigateUp() {  
 onBackPressed();

return true;  
}

**Secventa de cod 9 :Suprascrierea metodei onSupportNavigateUp ce deserveste ca scop posiblitatea de a te intoarce in pagina anterioara din comanda tastaturii virtuale a telefonului.**

Cu ajutorul metodei suprascrise onResponse adaptorul preia toate toate note-urile definite in lista de parametrii a functiei si le afiseaza in interfata user-ului

@Override  
public void onResponse(List<Note> responseItems) {  
 adapter.addAll(responseItems);  
}

**Secventa de cod 10 :prin intermediul acestei metode , adaptorul preia o lista de note-uri pe care le adauga in lista sa de obiecte , urmand apoi sa le scrie in interfata , prin intermediul interfetei historyListView**

In cadrul clasei , am folosit pasaje din librariile android , android , java si ro.mds.note pentru crearea si dezvoltarea calitatilor serviciilor oferite :

- android.content.DialogInterface

- android.view.Menu

- android.view.MenuItem

- android.widget.\*

- androidx.appcombat.app.AlertDialog

- androidx.appcombat.app.AppCombatActivity

- android.content.ident

- android.os.bundle

- java.util.List

- ro.mds.note.R

- ro.mds.note.adapter.NoteListAdapter

- ro.mds.note.entity.Note

- ro.mds.note.helper.LazyLoader

- ro.mds.note.helper.NotesManager

- ro.mds.note.helper.ResponseListener

### 1.3.3 Fereastra de manipulare a notitelor



Prin intermediul acestei ferestre utilizatorul isi poate scrie si salva notitele dorite .

Grafica ferestrei se prezinta in aceeeasi nuanta de albastru si alb ca la pagina principala.Toolbar-ul are o culoare violet-albastru in timp ce restul ferestrei are culoarea alba .Scrisul in cadrul paginii se infatiseaza in nonculori si in detrimentul puterii culorii de fundal.Pagina se prezinta cu doua campuri de completat destinate utilizatorului ce reprezinta titlul si textul notitei ce ar urma sa fie salvate.

Pentru a-l orienta pe utilizator asupra scopului fiecarui camp , am scris un hint text in fiecare camp ce defineste ideea si folosinta campului respectiv.

Functia protected void “onCreate” suprascrisa din clasa “AppCombatAtivity” se declanseaza la deschiderea ferestrei . Aceasta are ca parametru o variabila de tip “Bundle” si indeplineste rolul de initiere a ferestrei . De regula , toate metodele”onCreate” au acest incipit in cadrul functiei.

Functia void “setContentView” primeste ca parametru layout-ul “R.layout.activity\_note” si are insemnatatea de a stabili layout-ul pentru activitatea NoteActivity”. Bara de instrumente este definita de valoarea “toolbar” din cadrul interfetei. Se foloseste metoda “setSupportActionBar” pentru a stabili bara de instrumente definita anterior in cadrul ferestrei noastre . Ulterior se verifica daca toolbar-ul s-a salvat in sistemul clasei prin intermediul functiei “getSupportActionBar()” . In cazul in care functia returneaza un rezultat diferit de null , i se configureaza barei de instrumente optiunea de a se intoarce in pagina principala (home) , prin intermediul butonului sub forma de sageata si i se defineste titlul barei ce reprezinta ideea ferestrei (“New note” – crearea unui nou note).

androidx.appcompat.widget.Toolbar toolbar = findViewById(R.id.*toolbar*);  
setSupportActionBar(toolbar);  
if (getSupportActionBar() != null) {  
 getSupportActionBar().setDisplayHomeAsUpEnabled(true);  
 getSupportActionBar().setTitle("New note");  
}

**Secventa de cod 11 :definirea si configurarea barei de instrumente**

Ulterior , verificam daca nu cumva pentru a se deschide aceasta fereastra am fost redirectionati dintr-o alta pagina (in cazul de fata , se verifica daca pagina s-a creat dintr-o redirectionare de la un element din “HistoryActivity” . Pentru aceasta , se incearca sustragerea string-ului trimis probabil din “HistoryActivity” pentru a se afisa detaliile notitei deschise din clasa history cu ajutorul metodei “getStringExtra” apelata de functia ce returneaza obiectul folosit pentru intercomunicarea activitatilor si transfer de informatii “getIntent()”. In caz afirmativ de redirectionare din clasa HistoryActivity se executa mai multi pasi pentru sustragerea informatiilor fisierului respectiv din baza de date a mobilului :

In primul rand , se instantiaza variabilele pentru campurile de titlu si continut ce urmeaza sa fie scrise cu datele din notita salvata .

In al doilea rand , prin intermediul functiei statice “readNote” provenita din clasa “NotesManager” se citesc datele din notita deschisa in history\_activity . Se dau ca parametrii metoda ce returneaza contextul actual si titlul obiectului din care dorim sa extragem datele si sa le afisam in fereastra actuala . In cadrul metodei se instantiaza o variabila “Note” cu valoarea null , pe care urmeaza sa-i furnizam datele despre notita selectata . Liniile de cod din cadrul functiei sunt impachetate intr-o afirmatie “try”,urmat de un statement “catch” pentru a preveni si detecta prezenta unei erori IO ce se petrece regulat la citirea sau scrierea de fisiere . In interiorul impachetarii “try” se defineste variabila “fis” de tip “FileInputStream” care deschide fisierul text al carui titlu este transmis prin parametrul “title” . Obiectul “objectMapper” de tip “ObjectMapper” furnizeaza functionalitatea de citire in JSON a datelor din fisier in variabila note , completand parametrii constructorului “Note” a valorii note. Pentru aceasta aplicare variabila “objectMapper” apeleaza metoda “readvalue” ce primeste ca parametri fisierul din care se citesc bitii si tipul clasei , pentru care se vor structura datele citite citite din fisier ; si se va apela constructorul clasei date ca al doilea parametru avand ca parametrii datele obtinute din fisier , urmand apoi sa returneze obiectul de tip “Note” cu datele fisierului . Ulterior , se inchide fisierul “fis” si se returneaza note-ul construit .

static public Note readNote(@NotNull Context context, String title) {  
 Note note = null;  
 try {  
 FileInputStream fis = context.openFileInput(title);  
 ObjectMapper objectMapper = new ObjectMapper();  
 note = objectMapper.readValue(fis, Note.class);  
 fis.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return note;  
}

**Secventa de cod 12: functia de citire si preluare a informatiilor dintr-o notita .**

In al treilea rand , in cazul in care fereastra nu a fost accesata din redirectionarea din clasa NoteActivity , ci din MainActivity prin apasarea butonului ce executa actiunea de forward “TAKE A NOTE” ; atunci se instantiaza un nou obiect de tip “Note” fara niciun parametru , intrucat campurile sunt blank si urmeaza sa fie completate de utilizator .

In cadrul toolbar-ului definit ca variabila sub acelas nume , s-a furnizat un buton de intoarcere spre pagina principala in partea stanga a barei de instrumente , sub forma unei iconite de culoarea alba . Prin apasarea acestui buton , se activeaza functia void “onBackPressed()” din cadrul metodei “onSupportNavigateUp()” , provenita din cadrul clasei “AppCombatActivity”. Titlul ferestrei ce reflecta ideea paginii se afla pe partea stanga a toolbar-ului , in dreapta iconitei in forma de sageata . Butonul “save” se situeaza in extremitatea dreapta si indeplienste rolul de a salva datele introduse , urmand apoi ca utilizatorul sa fie redirectionat catre pagina de istoric. Metoda onOptionsItemSelected provenita tot din cadrul clasei “AppCombatActivity” si suprascrisa in clasa actuala are ca parametru item-ul pe care clientul il poate declansa si in cadrul caruia se vor executa mai multe linii de cod . Pentru a se identifica ce item a fost declansat se compara fiecare buton sau element declansator de actiuni cu item-ul returnat in urma declansarii serviciului “item.getItemId()”.Item-ul cu id-ul “saveNote” este folosit pentru a salva datele scrie in cadrul ferestrei . In urma declansarii item-ului se executa actiunile absolut necesare dar si actiuni standarde folosite pentru o utilizare placuta si eficienta a aplicatiei.

Variabila “title” de tip “EditText” este instantiata cu item-ul definit in interfata “noteTitle” prin intermediul metodei “findViewById” si este variabila ce contine titlul textului ce urmeaza sa fie salvat . Variabila “content” tot de tip “EditText” este instantiata cu item-ul definit in interfata “noteDetails” tot prin intermediul metodei “findViewById” si este variabila ce continte continutul textului .

Variabila privata note de tip-ul clasei “Note” definita in pachetul “entity” indeplineste scopul de a salva string-urile ce contin titlul si continutul fiecarei nota in cadrul programului .

In constructorul clasei se regasesc parametriii pentru variabilele clasei : “title” si “content”.

public Note(String title, String content) {  
 this.title = title;  
 this.content = content;  
}

**Secventa de cod 13: constructorul clasei Note**

De asemenea am definit metodele de tip “get” si “set” pentru fiecare variabila privata din clasa .

public String getTitle() {  
 return title;  
}  
public void setTitle(String title) {  
 this.title = title;  
}  
public String getContent() {  
 return content;  
}  
public void setContent(String content) {  
 this.content = content;  
}

**Secventa de cod 14: metodele prin care sustragem sau modificam datele notitei din cadrul programului.**

Revenind la utilitatea variabilei “note” de tipul “Note” . In cadrul metodei “onOptionsItemSelected” , aceasta preia si salveaza datele din variabilele “title” si “content” ; dupa care le salveaza in folder-ul cu fisiere al aplicatiei cu ajutorul functiei statice “saveNote” ce are ca parametrii contextul ferestrei din care dorim sa salvam fisierul si variabila de tip “Note” din cadrul careia extragem datele si le salvam intr-un fisier in memorie.Functia este declansata prin intermediul clasei ce si detine metoda , “NotesManager”.

In cadrul metodei “saveNote” atragem atentia asupra exceptiei de tip IO ce se poate activa in momentul in care nu putem scrie / citi dintr-un fisier , astfel ca impachetam tot codul scris intr-o afirmatie “try” , urmand apoi sa definim o afirmatie “catch” ce returneaza “null” si are drept scop afisarea erorilor aparute in urma tentativei de a scrie in fisier : “e.printStackTrace()”.

In interiorul impachetarii “try” definim variabila “fos” de tipul “FileOutputStream” ce ne ajuta in definirea si scrierea datelor in fisierul de output . In primul parametru introducem titlul notitei salvate . Titlul fiecarei notite va fi titlul ales si scris in fereastra aplicatiei . Am ales ca numai in cadrul acestei aplicatii sa poata fi modelat fisierul , astfel ca al doilea parametru ce are ca scop furnizarea sau nu a dreptului de acces l-am definit ca fiind “Context.MODE\_PRIVATE” ; pentru a putea fi accesat doar de catre aplicatia apelanta .

Valoarea “objectMapper” de tipul “ObjectMapper” are rolul de a scrie efectiv datele din note in sistemul de fisiere , in format JSON pentru a putea fi usor ulterior de a prelua datele din fisier si introduse in variabila “note”. Acest obiect este instantiat fara parametrii si executa actiunea prezentata mai sus prin intermediul metodei “writeValue” ce are ca parametrii stream-ul de output in cadrul caruia se vor scrie datele si clasa note , ce contine datele despre fisier .

Dupa scrierea datelor in fisier, se inchide stream-ul “fos” si returnam note-ul dat ca parametru (acest lucru ne ajuta la testarea metodei in cadrul tool-ului de testare Junit5).Ulterior , dupa ce iesim din metoda , revenim in functie si se doreste redirectionarea utilizatorului in fereastra istoric . Acest lucru se realizeaza in continuarea metodei “onOptionsItemsSelected” dupa parcurgerea pasilor prezentati mai sus . Se instantiaza variabila “intent” de tip “Intent” ce are drept folosinta redirectionarea utilizatorului in interfata “history\_activity”. Valoarea “intent” este definita de constructor ce primeste ca parametrii clasa actuala “NoteActivity.this” si clasa specifica interfetei care se doreste sa se redirectioneze utilizatorul “HistoryActivity.this” . Ulterior se apeleaza functia void “startActivity” ce primeste ca parametrul valoarea “intent” : variabila asupra careia sa se efectueze operatia de redirectionare . In cazul in care toti pasii se vor parcurge fara erori , variabila va returna “true” , cum ca operatiunea a fost executata cu succes .

static public Note readNote(@NotNull Context context, String title) {  
 Note note = null;  
 try {  
 FileInputStream fis = context.openFileInput(title);  
 ObjectMapper objectMapper = new ObjectMapper();  
 note = objectMapper.readValue(fis, Note.class);  
 fis.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return note;  
}

**Secventa de cod 15: Citirea datelor din fisier si introducerea acestora in variabila note**

In dezvoltarea acestei clase am folosit o multitudine de clase (deja definite sau definite de mine) ce imi ofera suport in modelarea codului :

- android.content.Intent

- android.os.Bundle

- android.view.Menu

- android.view.MenuItem

- android.widget.EditText

- androidx.appcombat.app.AppCombatActivity

- ro.mds.note.R

- ro.mds.note.entity.Note

- ro.mds.note.helper.NotesManager

## 1.4. Cadrul de testare JUnit 4

JUnit a fost folositt în dezvoltarea bazata pe teste si este una dintre familiile de cadre de testare a unitatilor, care este cunoscut colectiv ca xUnit, provenit de la SUni , un cadru de testare a unitatii pentru limbajul de programare Smalltalk. Aceasta dispune de o multitudine de unelte folosite pentru a testa fiecare metoda individual , realizand o testare mult mai simplista si eficienta decat introducerea de texte in codul propriu zis pentru testari ; astfel este o unealta foarte importanta pentru testarea unui cod de dimensiuni mari . Exceptiile si manipularea erorilor sunt mult mai usor de verificat datorita uneltelor si modului de afisarea a aparitiei unei erori . De asemenea , se furnizeaza si configuratii pe care programatorul le poate configura astfel incat sa prioritizeze anumite teste facute .

Toate testele efectuate se produc individual si intr-o ordine aleatoare, astfel ca niciun test nu e dependent de altul .

In cadrul proiectului am folost unitatea de testare JUnit 4 , ce dispune de uneltele necesare si suficiente pentru a testa orice metoda implementata in program . Fiecare clasa ce detine un test este clona acesteia ( in materie de locatia unde se afla si de nume partial) ; diferenta fiind doar adaugarea sufixului “Test” la finalul numelui clasei .

In proiect s-au incercat teste pe platformele JUnit4 si JUnit 5 . Deoarece am constatat unele lipsuri si erori de sistem venite din partea JUnit5 , programatorii s-au decis sa utilizeze doar JUnit 4 , intrucat in contextul aplicatiei instrumentele oferite de JUnit 4 erau suficiente .

Testul clasei “NotesManager” se numeste “NotesManagerTest” . Aceasta realizeaza teste practice pentru a evalua codul scrisi si a se analiza posibile erori ce pot aparea in cadrul utilizarii aplicatiei de catre client . Aceasta clasa este formata din variabilele private “context” si “note” de tip “Context” , respectiv “Note” si functiile folosite in cadrul testelor .

Initial , se defineste functia void “init” fara parametri ce are anotatia “@Before” , astfel ca functia se declanseaza inaintea fiecarui test realizat . Se initializeaza variabila “context” instantiata mai sus cu o instanta registru ce tine referinta catre instrumentul ce ruleaza in proces ; mai exact , i se defineste contextul aplicatiei tinta. Ulterior , se verifica daca contextul gasit este cel corect . Stim ca contextul trebuie sa se afle in pachetul “ro.mds.note” , asa ca prin intermediul metodei “assertEquals” ,metoda special folosita pentru comparatii in cadrul testelor , se verifica ca numele pachetului in care se afla contextul si pachetul in care ar trebui acesta sa fie sa coincida ; in caz negativ , testul esueaza . Ulterior , se defineste note-ul instantiat pentru teste , cu un titlu si un text standard : “Test Save” , “Save this note” .

Dupa parcurgerea fazei incipiente , programul intra in primul test scris (in timpul rularii acest test poate sa fie oricare din cele definite ; eu fac referire in ordinea lor cronologica gasite in codul scris ) . Acesta include , bineinteles , anotatia @Test pentru a se identifica rolul acelei metode si rezolva primul test din cadrul clasei “NotesManager” . Se verifica corectitudinea functiei “readNote” cu ajutorul metodei “assertEquals” , ce compara note-ul definit in functia “init” si cel returnat de functia statica “readNote” . Pentru ca testul sa treaca cu succes , note-ul din primul parametru si cel returnat de functie trebuie sa fie echivalente . Functia “readNote” primeste ca parametru contextul definit in functia incipienta si titlul “note-ului” pe care trebuie sa-l gaseasca . Dupa ce parcurge toti pasii functiei prezentate in cadrul capitolului 1.3.3 , aceasta ar trebuie sa returneze notita al carui titlu este dat in al doilea parametru . In urma testelui verificam corectitudinea acestor pasi .

Al doilea test efectuat are anotatia “@Test” si include functia statica “saveNote” a clasei “NotesManager” . Acest test verifica daca o notita se salveaza cu succes de catre metoda . Similar ca la primul test , se verifica ca note-ul returnat de functia “saveNote” si note-ul definit sa coincida . Metoda “saveNote” primeste ca parametrii contextul actual si note-ul ce ar urma sa fie salvat in sistem. Parcurgand pasii functiei din capitolul 1.3.3 , se efectueaza actiunile de salvare a note-ului in sistem si se returneaza note-ul (in cazul in care nu a aparut o exceptie , atunci ar intra in cazul de exceptie si s-ar returna null) . Daca valoarea returnata de functie este “null” , atunci cele doua note-uri nu coincid , deci testul pica iar functia trebuie reanalizata .

Al treilea test implica salvarea si citirea unei lsite de note-uri . Aceasta functie este aplicata in cadrul activitatii “HistoryActivity” , avand rolul de suport in operatia de incarcare eficienta a notitelor in aplicatie . Se defineste variabila “notes” de tipul listei de elemente “Note” si se initializeaza ca fiind un ArrayList . Se definesc doua notite de test ce se introduc in lista notes si se salveaza in sistem . Ulterior , se parcurg notitele din lista si se verifica daca acestea au fost salvate cu succes prin intermediul funcitiei “assertEquals” . Se verifica ca fiecare valoare de tipul “Note” din lista sa corespunda cu elementul returnat de functia “readNote” ce primeste ca al doilea parametru titlul respectivului obiect din lista . In cazul in care unul dintre obiecte nu corespunde cu obiectul returnat de functia “readNote” , testul va esua .

In a patra verificare se testeaza veridicitatea functiei de a sterge o notita . In cadrul metodei de stergere “deleteNote” se returneaza “true” daca notita cu titlul dat ca parametru este stearsa cu succes din sistem, altfel se returneaza “false” . Testul functioneaza doar daca metoda “deleteNote” returneaza “true” in urma procesului de stergere . Verificarea rezultatului returnat de metoda “deleteNote” este realizata de functia provenita din bibliotecile junit , “assertTrue”.

Pentru dezvoltarea platformei de testare programatorii au apelat la metodele claselor prezente in bibliotecile android , java , junit ; precum si cele folosite in cadrul testarii :

- android.content.Context

- androidx.test.platform.app.InstrumentationRegistry

- org.junit.Before

- org.junit.Test

- java.util.ArrayList

- java.util.List

- ro.mds.note.entity.Note

- org.junit.Assert.assertEquals

- org.junit.Assert.assertTrie

## 1.5. Concluzii

Scopul acestei lucrari a fost acela de a identifica provocarile ridicate de implementarea unei aplicatii pentru manipularea documentelor pe cale vocala sau textuala si de a gasi solutii optime pentru rezolvarea lor . Multe functionalitati din cadrul programului au fost implementate si fixate cu ajutorul conceptelor invatate din video-uri de pe youtube , comentarii de pe site-uri dedicate programatorilor sau din consultatii cu alti colegi .

Implementarea clientului pentru platforma Android a devenit usoara din momentul intelegerii arhitecturii unei asemenea aplicatii . Partea de implementare nativa a anumitor componente nu a ridicat probleme majore , desi suportul acestora pe platforma Android este inca marcat ca fiind experimental . Am vazut cum implementarea tehnicii “LazyLoader” reduce major complexitatea de timp a incarcarii notitelor pe pagina si cum implementarea unei unitati de test ma ajuta in a identifica erorile ce au aparut in timpul construirii fiecarei metode dedicata manipularii fisierelor .

Utilizarea in incipit a anotatiilor din JUnit 5 a ridicat mai multe probleme la nivelul scriptului de construire “Gradle” , fiind necesar sa abandonam utilizarea acestei versiuni de JUnit .

Partea de grafica a aplicatiei a fost un subiect dezbatut amanuntios de catre dezvoltatorii proiectului . Au fost facute analize si cercetari pe mai multe aplicatii Android pentru a se concluziona tipul si nuantele de culori ce se regasesc in trend in cele mai utilizate aplicatii de pe platforma Magazin Play .

Titlul aplicatiei “Note” a fost ales unul simplist si la obiect , fiind reprezentativ pentru ideea programului .

Codul a fost in totalitate comentat si explicat , linie cu linie , pentru a-l ajuta pe orice alt programator sa intealaga rapid programul si pe viitor , in cazul in care vor aparea anumite bug-uri in sistem .

## 1.6. Bibliografie

[1] Google Inc., Android Developers Documentation - <https://developer.android.com/reference/packages.html>

[2] JUnit 4

<https://junit.org/junit4/>