Editor de text cu salvare automată în cloud (server) - Simple Text Editor

Moldovan Eduard,

Informatică, anul 2

Cuprins

[1. Introducere 2](#_Toc1913134290)

[2. Tehnologii folosite 3](#_Toc2049239796)

[2.1. Servicii Android 4](#_Toc1535063157)

[2.1.1 Share intent 4](#_Toc1620849089)

[2.1.2 Create document intent 5](#_Toc2095350829)

[2.2. Biblioteci de cod externe 5](#_Toc1267556944)

[2.2.1. Ktor 5](#_Toc165925185)

[2.2.2. Google Material Icons Extended 5](#_Toc877608254)

[2.2.3. Compose Navigation 5](#_Toc500054788)

[3. Proiectarea aplicației 5](#_Toc155900834)

[3.1. Generalități 6](#_Toc1405621909)

[3.2. Funcționalități 6](#_Toc1750401257)

[3.3. Comunicare client-server 7](#_Toc1057333431)

[3.4. Arhitectura internă a aplicației 7](#_Toc717599021)

[4. Aspecte de implementare 9](#_Toc1004127454)

[4.1. Profilul utilizatorului 10](#_Toc2141619543)

[4.2. Stocarea pe dispozitiv (stocare internă) 10](#_Toc2077012240)

[4.3. Stocarea datelor pe server 11](#_Toc1048250473)

[4.4. Gestionarea modificărilor (undo/redo) 11](#_Toc204379137)

[4.5. Export local 13](#_Toc579375297)

[4.5.1. Share intent 13](#_Toc1635959422)

[4.5.1. Create document intent 13](#_Toc902292620)

[4.6. Comunicare client-server 14](#_Toc761768721)

[5. Testare și concluzii 14](#_Toc2061517486)

[Bibliografie 15](#_Toc1226976896)

# 1. Introducere

Smartphone-urile sunt aproape indispensabile în ziua de azi: de la gestionarea banilor la socializare, folosim aceste dispozitive în fiecare zi.

Putem îndeplini aproape orice sarcină pe aceste telefoane, însă una din sarcini care este puțin mai dificilă este editarea fișierelor de tip text simple. Aceasta se întâmplă deoarece aceste telefoane nu au în general un editor de text implicit (de exemplu, echivalentul Notepad pe Windows, TextEdit pe macOS sau nano pe Linux), așadar acest proiect încearcă să schimbe această realitate.

Proiectul va fi implementat pentru platforma Android, deoarece este cel mai folosit sistem de operare pentru smartphone-uri, costul unui dispozitiv Android este mult mai mic decât pentru dispozitivele rivale, iar dezvoltarea aplicațiilor pentru această platformă este facilitată de disponibilitatea sistemelor ajutătoare (Android SDK, Android Studio etc.) pe orice dispozitiv desktop (Windows, macOS, Linux).

Aplicațiile Android pot fi dezvoltate atât cu limbajul de programare Java, cât și Kotlin, care este un limbaj mai simplu, dar care rulează tot în JVM (Java Virtual Machine). Inițial, aplicațiile foloseau documente XML pentru a descrie interfața grafică, folosind și un editor vizual pentru implementare rapidă, însă mai recent, Google a dezvoltat un framework nou pentru dezvoltare și anume Jetpack Compose.

Jetpack Compose (sau pur și simplu Compose) este un cadru de dezvoltare creat în Kotlin care renunță la modelul tradițional, cu activități și fragmente, lăsând o singură activitate (cea principală), dar toată interfața fiind implementată direct în cod Kotlin. Fiecare componentă este o funcție Kotlin, așadar fragmentarea interfețelor complexe este mult mai ușoară deoarece orice componentă poate fi implementată într-o altă funcție, poate primi parametri și poate avea stare, adică variabile care rețin anumite date și recompun întreaga componentă când acea variabilă se schimbă. Toate aceste îmbunătățiri fac dezvoltarea în Compose mai rapidă decât modelul tradițional.

# 2. Tehnologii folosite

Așa cum s-a specificat, aplicația va fi dezvoltată pentru platforma Android, însă va folosi câteva servicii oferite de platformă, precum și câteva biblioteci de cod externe pentru a ușura dezvoltarea.

## 2.1. Servicii Android

Platforma Android dispune de o multitudine de servicii și utilități pentru a face dezvoltarea aplicațiilor mai ușoară. Multe dintre aceste servicii sunt parte din HAL (Hardware Abstraction Layer), care creează o interfață comună pentru diferitele tipuri și modele de periferice și senzori (de exemplu, Bluetooth, giroscop, accelerometru, cameră sau microfon), însă există o altă parte semnificativă care are scopul de a unifica experiența utilizatorilor între aplicație, în timp ce se păstrează securitatea prin a nu permite aplicațiilor acces deplin asupra sistemului pentru a îndeplini diferite sarcini.

Deoarece aplicația este un simplu editor de text, nu va folosi periferice precum camera sau giroscopul, dar va folosi funcționalități software care vor permite utilizatorilor să folosească fișierele editate și în afara aplicației, de exemplu să trimită un fișier text prin email sau să îl salveze într-o locație ușor accesibilă ulterior.

### 2.1.1 Share intent

Opțiunea de “Share” este extrem de utilă în aproape orice aplicație care lucrează cu fișiere deoarece permite utilizatorului să folosească un fișier în diferite moduri, cum ar fi trimitere prin email, prin aplicații de mesagerie (WhatsApp, Discord etc.), către alte dispozitive din apropiere prin Quick Share.

Implementarea acestei opțiuni are următoarele avantaje:

* Aplicația nu trebuie să determine dacă anumite aplicații de mesagerie sau de gestionare a emailurilor sunt prezente sau nu pe sistemul utilizatorului: Android va gestiona aplicațiile care vor putea primi fișiere. Aceasta scurtează dramatic timpul de dezvoltare, precum și dificultatea implementării acestei opțiuni.
* Interfața cu utilizatorul (UI) este aceeași pentru toate aplicațiile cu această opțiune, așadar utilizatorul este deja familiarizat cu procedeul de trimitere a fișierelor. Aplicația nu trebuie să creeze o nouă interfață pentru această funcționalitate.
* Aplicația nu trebuie să gestioneze în mod direct comunicarea cu aplicația aleasă de utilizator: Android va gestiona toate detaliile de implementare. Aplicația de editare nici nu are posibilitatea de a determina aplicația aleasă de utilizator, dar acest lucru nici nu este necesar.

### 2.1.2 Create document intent

Opțiunea de a crea un document este utilă în situația în care utilizatorul dorește să folosească fișierul și în afara aplicației sau pur și simplu să îl aibă salvat într-o locație ușor accesibilă. În acest caz, Android va deschide aplicația de sistem care va permite utilizatorului să aleagă locația, precum și denumirea fișierului.

## 2.2. Biblioteci de cod externe

Aproape orice aplicație folosește biblioteci (sau librării) de cod externe, create de alți dezvoltatori sau chiar de aceeași dezvoltatori, cu scopul de a reduce timpul de dezvoltare și a ușura munca.

### 2.2.1. Ktor

Ktor[2] este o bibliotecă de cod dezvoltată de JetBrains pentru a facilita crearea de cereri HTTP. Această bibliotecă de cod este mult mai flexibilă și mai ușor de folosit decât clasa URLConnection folosită implicit de Android.

### 2.2.2. Google Material Icons Extended

Google Material Icons[3] este un ajutor minunat pentru dezvoltatorii de aplicații, nu numai pe Android: oferă gratuit peste 2500 de pictograme pentru aproape toate tipurile de acțiuni. Android conține implicit un subset din aceste pictograme, însă pentru unele acțiuni, aplicația va folosi pictograme indisponibile implicit, așa că s-a adăugat această bibliotecă.

### 2.2.3. Compose Navigation

Compose Navigation[4] este componenta oficială pentru a gestiona navigarea între ecrane în Compose, pentru a nu se mai folosi modelul vechi cu multe intenții și fragmente. Această bibliotecă de cod permite navigarea simplă între pagini, tranziții și multe altele.

# 3. Proiectarea aplicației

## 3.1. Generalități

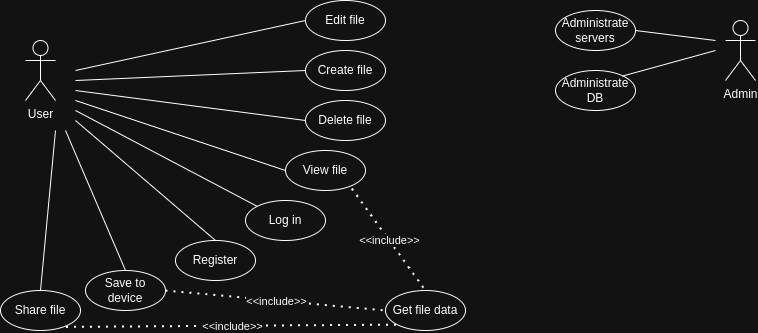
Aplicația va folosi Jetpack Compose pentru dezvoltare, iar versiunea Android minimă suportată va fi 7.0 (Nougat), aceasta fiind și cea mai veche versiune care încă primește actualizări de la Google la momentul actual. Conform statisticilor[1], procentul de dispozitive care au această versiune de Android sau mai mare este de aproximativ 97%, deci aplicația va rula pe 97% din dispozitivele în uz la momentul de față.

Serverul aplicației va fi dezvoltat în ASP.NET Core Web API, cu limbajul de programare C#. Platforma .NET (în prezent la versiunea 9) este una din cele mai folosite și apreciate pentru dezvoltarea de aplicații server, în principal deoarece dispune de o multitudine de ajutoare care fac dezvoltarea rapidă și plăcută. Pentru baze de date se va folosi MySQL 8, iar conectarea între server și DB va fi făcută cu ajutorul Entity Framework Core, un ORM care reduce complexitatea limbajului SQL și atacuri de tip “SQL injection”. Toate fișierele vor fi stocate într-o locație prestabilită pentru server pe mediul de stocare, fără a se folosi servicii cloud precum Amazon S3 sau Azure Blob Storage.

Editorul de text va dispune de funcționalități de anulare și refacere (undo/redo) a textului, acestea fiind stocate în memorie până la închiderea aplicației. De asemenea, editorul permite utilizatorului să schimbe documentul curent dintre cele stocate local sau cele de pe server. Schimbarea documentului NU va șterge detaliile despre undo/redo a documentului anterior, astfel că utilizatorul poate edita mai multe documente în același timp.

## 3.2. Funcționalități

* Facilitarea editării documentelor de tip text local, pe dispozitivul propriu
* Posibilitatea de a salva fișierele pe un server (unde se va crea un cont) în timpul editării
* Posibilitatea de a trimite fișierele prin metode controlate de către sistem (Share)
* Posibilitatea de a salva fișierele local, într-o locație aleasă de către utilizator
* Posibilitatea de a schimba dimensiunea fontului editorului pentru persoanele cu deficiențe de vedere sau pur și simplu o preferință pentru dimensiunea textului

Fig. 3.1. Diagrama cazurilor de utilizare

## 3.3. Comunicare client-server

Comunicarea între client și server se va face prin protocolul HTTP, de departe cel mai utilizat protocol pentru a comunica între dispozitive. Pentru această funcționalitate, Android pune la dispoziție clasa URLConnection, însă aplicația va folosi biblioteca Ktor[2] pentru a facilita procesul de comunicare.

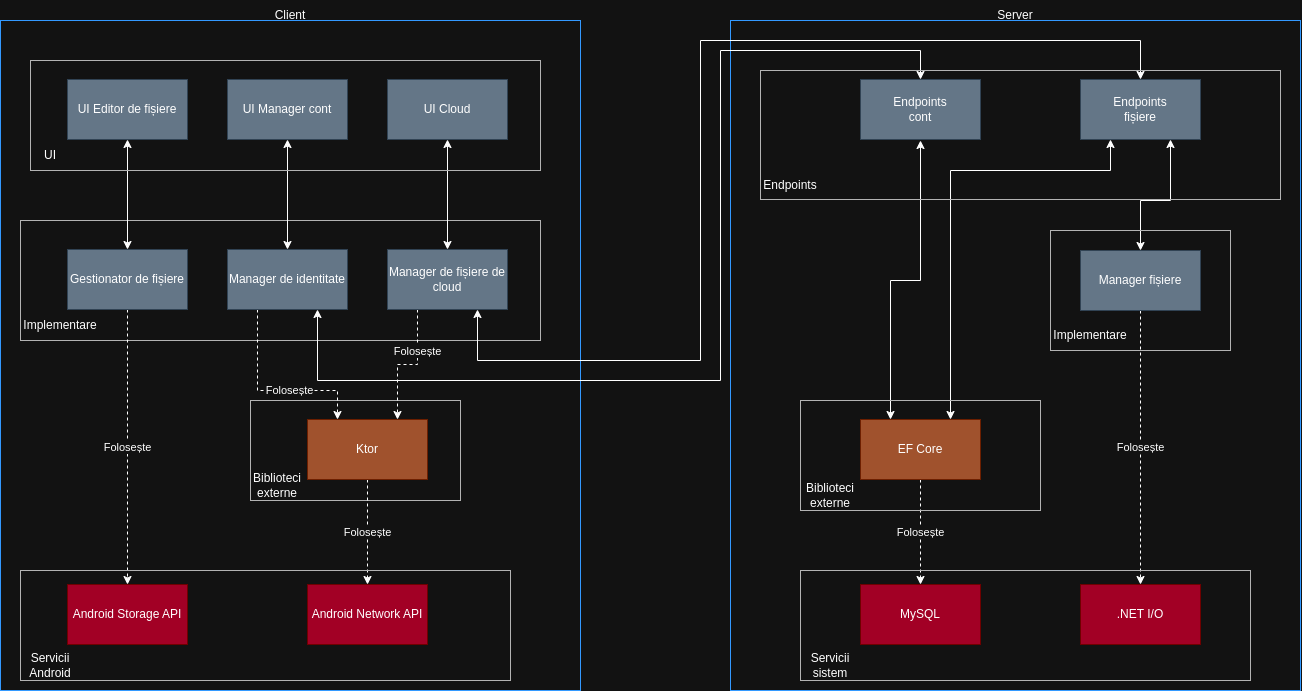
Fiecare cerere va primi un răspuns de pe server: unele cereri vor avea un răspuns care conține date (ex. lista tuturor fișierelor create de către utilizator pe server), altele care sunt goale când totul s-a configurat cu succes sau un mesaj de eroare când ceva a mers prost; acest mesaj de eroare va fi afișat de către aplicație dacă e necesar.

Utilizatorul va putea să își creeze cont, să se autentifice într-un cont existent, să își reseteze parola și să se deconecteze. Pentru fișiere, toate operațiile uzuale sunt implementate: creare, citire, actualizare și ștergere. Fișierele nu pot avea o dimensiune mai mare de 500KB, deoarece aceasta ar permite potențialilor atacatori să “inunde” serverul cu fișiere foarte mari și să creeze potențiale pagube sau costuri de întreținere ridicate. De asemenea, foarte puține fișiere text sunt utilizabile la dimensiuni atât de mari. Dacă un fișier depășește 500KB, serverul nu va mai accepta cereri care vor crește dimensiunea fișierului, iar aplicația va afișa acest avertisment.

De fiecare dată când utilizatorul modifică documentul, iar aplicația consideră că această schimbare e semnificativă și trebuie salvată (ex. a trecut o secundă de la ultima editare, poziția cursorului s-a schimbat), se va face o cerere către server, conținând doar modificările, salvând astfel timp și resurse deoarece nu se trimite întreg fișierul de fiecare dată. Când utilizatorul părăsește aplicația sau schimbă documentul activ, o cerere ce va conține întreg fișierul va asigura sincronizarea între server și client.

## 3.4. Arhitectura internă a aplicației

Intern, aplicația folosește o combinație de sisteme, biblioteci externe și API-uri Android pentru a funcționa. O viziune de ansamblu asupra sistemului este prezentată în **Fig. 3.1.**

Fig. 3.2. Arhitectura aplicației

Deși MySQL poate stoca fișierele ca și obiect BLOB sau chiar ca LARGETEXT, serverul va stoca fișierele direct pe sistemul de fișiere deoarece:

* Fișiere multe sau mari vor încetini baza de date
* Editarea unei porțiuni mici din fișier se poate face mult mai rapid în fișiere fizice decât într-o bază de date deoarece fluxurile (FileStream în C#) pot să scrie date la orice index spre deosebire de DB unde se va modifica tot conținutul de fiecare dată
* În acest mod nu este necesar întregul conținut, ci doar partea modificată, reducând astfel timpul de răspuns și lățimea de bandă folosită de server

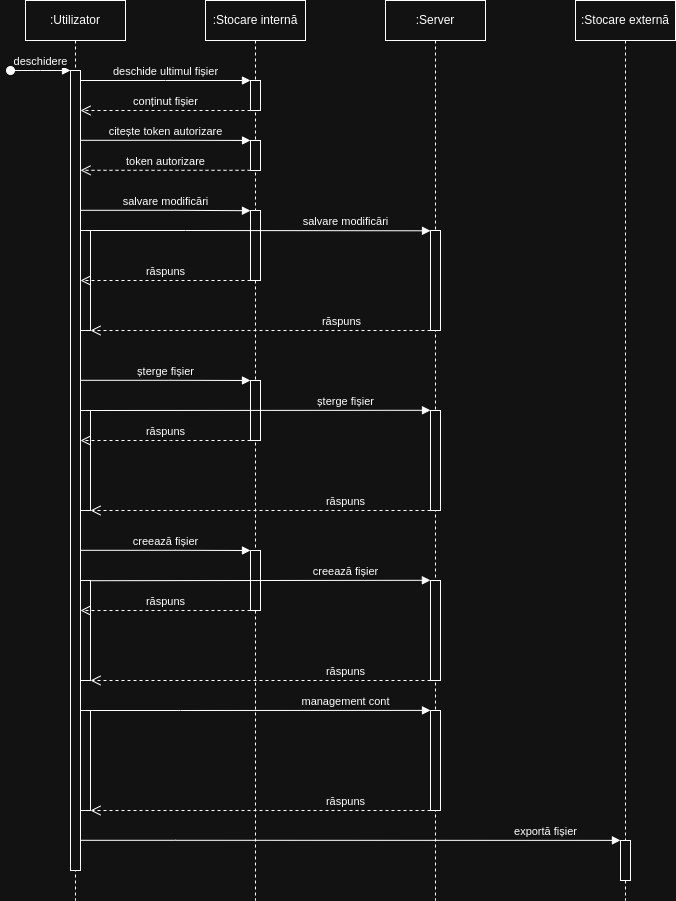


Fig. 3.3. Diagrama de secvență a aplicației

În **Fig. 3.4.** se poate vedea modul în care utilizatorul poate naviga între paginile principale ale aplicației.

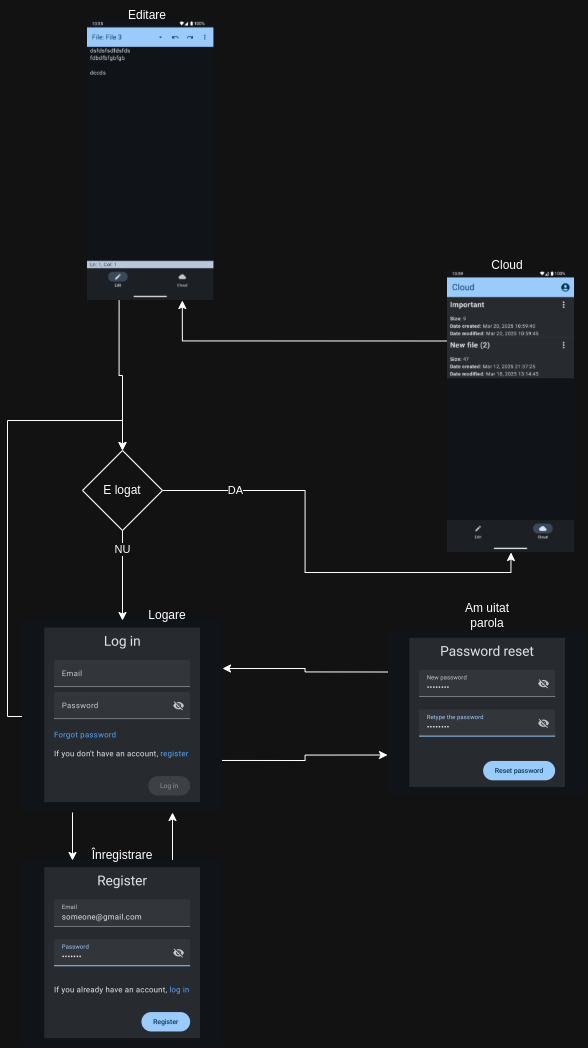


Fig. 3.4. Diagrama UI a aplicației

# 4. Aspecte de implementare

## 4.1. Profilul utilizatorului

Utilizatorul va trebui să își creeze un cont pentru a folosi serviciul cloud. Aplicația va rula și fără un cont, dar nu va putea folosi serviciile cloud.

Utilizatorul se poate autentifica (Login) specificând emailul și parola sa. Dacă există acel email în baza de date, iar parola este corectă, se va stoca în baza de date un token JWT (JSON Web Token), care va fi, de asemenea, trimis către client, acesta din urmă stocând acel token în datele aplicației. Pentru fiecare cerere care interacționează cu fișiere, clientul va trimite acel token în header-ul HTTP Authorization ca valoarea “Bearer: <TOKEN>”; pe server se va verifica dacă token-ul dat este același cu cel din baza de date, iar dacă token-urile nu se potrivesc se va trimite un mesaj care îi indică utilizatorului să se autentifice în cont. Utilizatorul își poate reseta parola dacă a uitat-o.

Dacă nu are un cont, utilizatorul își poate crea cont (Register) specificând emailul și o parolă (minimum 6 caractere, cel puțin o literă mare și cel puțin o cifră). Dacă emailul există deja, cererea va răspunde cu o eroare. Altfel se vor stoca aceste date, iar aplicația va trimite și o cerere de autentificare automat.

Parola va fi stocată ca un hash în baza de date folosind PasswordHasher din .NET, care se va ocupa automat de a adăuga un “salt” (un șir aleatoriu de bytes, care se va asigura că 2 parole la fel au un hash diferit pentru a spori securitatea).

## 4.2. Stocarea pe dispozitiv (stocare internă)

Fișierele create de utilizator vor fi stocate pe dispozitiv în spațiul de stocare intern al apliației. Acestea nu vor fi accesibile din afara aplicației, dar utilizatorul va putea să exporte aceste fisiere (discutat în detaliu în capitolul 2.1). Fiecare fișier va avea ca și nume un identificator unic (UUID) pentru a evita eventualele conflictele între denumiri. Utilizatorul poate crea sau șterge aceste fișiere, dar nu va avea acces la ID-ul lor, ci doar la numele specificat de el. Numele, precum și ID-ul și dacă fișierul e stocat și pe cloud sau nu, sunt stocate ca o listă într-un fișier JSON. Modelul pentru stocarea acestor date este următorul:

@Serializable  
data class SavedFileDetails(  
 val name: String,  
 var id: String,  
 var isStoredOnCloud: Boolean  
)

## 4.3. Stocarea datelor pe server

Așa cum a fost deja specificat, serverul folosește MySQL ca și bază de date, aceasta având 2 tabele: Files și Users (modelele sunt mai bine descrise în **Fig. 4.1.**). Fișierele sunt stocate în locația specifică a datelor aplicației (ex. pe Linux va fi “/home/<USER>/.local/share/SimpleTextEditorServer”), dar pentru fiecare zi se va crea un director cu numele în formatul “yyyy-MM-dd" (ex. 2025-02-25), pentru a scala aplicația deoarece zeci de mii de fișiere într-un singur director vor face sistemul foarte lent. Așadar, ziua în care fișierul a fost creat va fi folosită pentru a determina locația finală a acelui fișier. Data creării fișierului nu va fi modificată, deci este cel mai bun candidat pentru acest tip de ierarhizare.

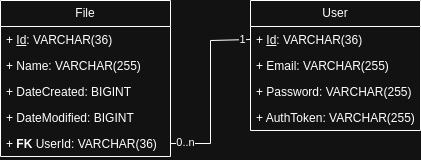


Fig. 4.1. Diagrama UML pentru modelele DB

## 4.4. Gestionarea modificărilor (undo/redo)

O funcționalitate importantă a oricărui editor de text este abilitatea de a anula o greșeală și de a reface o anumită modificare anterioară care a fost anulată. Acest sistem este numit colocvial ca și sistem “undo/redo”. Într-o formă simplă, funcționează foarte asemănător cu o stivă:

* Undo: stochează cea mai recentă modificare într-o stivă și modifică conținutul pentru a se potrivi cu ceea ce era înainte de modificare
* Redo: scoate cel mai recent element de pe stivă și modifică conținutul potrivit cu datele din elementul proaspăt scos

Totuși, câteodată este util să se anuleze câteva refaceri. Așadar, fiecare operație de Redo va fi stocată pe o altă stivă, iar dacă se cere Undo, se va scoate din acea stivă cel mai recent element. Astfel, se pot repeta operații de Undo sau Redo indefinit. Însă în acest sistem, dacă se vor folosi operații de Undo după Redo, dar apoi se face o modificare nouă (ex. Se scrie un caracter), această stivă va fi golită pentru a evita coruperi de date (deoarece indecșii de modificare nu se vor mai potrivi). Diagrama de stare a acestui modul este prezentată în **Fig. 4.2.**

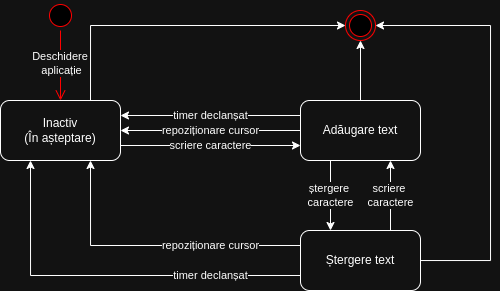


Fig. 4.2. Diagrama de stare a modulului de modificări

Pentru a fi util, acest sistem nu va înregistra modificări ale fiecărui caracter, ci doar când nici o editare nu s-a mai înregistrat timp de o secundă, atunci când poziția cursorului s-a schimbat sau atunci când direcția modificărilor se schimbă (adică atunci când, după câteva caractere, utilizatorul va începe să șteargă caractere sau invers). Acest model este ilustrat în **Fig. 4.3.**

La fiecare modificare, dacă fișierul e stocat pe server, se va face o cerere HTTP pentru a sincroniza datele de pe server cu cele de pe client, însă această cerere va conține doar datele necesare pentru modificare, nu întreg documentul.

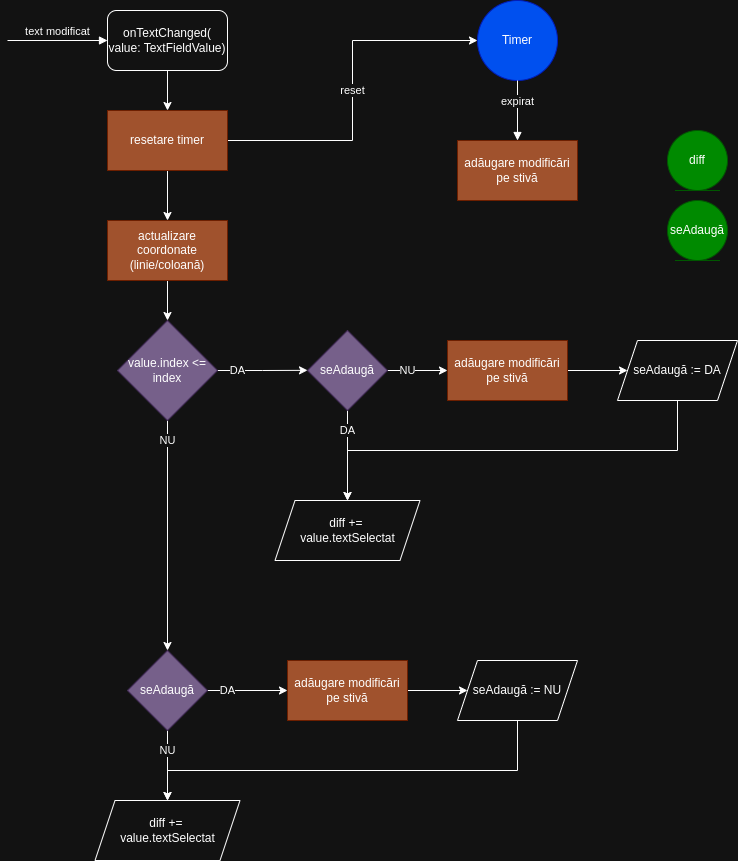


Fig. 4.3. Diagrama de activitate a funcției de înregistrare modificări

## 4.5. Export local

Fișierele create de utilizator sunt stocate în spațiul privat al aplicației, ceea ce face imposibilă accesarea lor de către alte aplicații, deci implicit utilizatorul nu poate să folosească fișierele în alte aplicații sau să le trimită altor persoane. Pentru a rezolva această problemă, se vor folosi funcționalitățile deja consacrate pentru a trimite fișierele: funcționalitatea de Share și salvarea într-o locație aleasă de utilizator (“Create document” pe Android).

### 4.5.1. Share intent

Pentru a folosi funcționalitatea de Share, implementarea este relativ ușoară:

* Se specifică un obiect <provider> în AndroidManifest.xml care va da permisiunea sistemului să trimită acest fișier către alte aplicații. Acest va avea o proprietate android:authorities, unde se specifică un identificator unic al aplicației, în acest caz: “com.example.simpletexteditor.fileprovider”.
* Se creează o copie a fișierului inițial, care se va stoca în cache-ul aplicației cu numele dat de utilizator.
* În final se creează URI-ul fișierului nou creat și se specifică intent-ul ACTION\_SEND, care va permite utilizatorului să aleagă aplicația destinație cu următorul cod:  
    
  val contentUri: Uri = getUriForFile(context, "com.example.simpletexteditor.fileprovider", friendlyFile)  
    
  val shareIntent: Intent = Intent().apply {  
   action = Intent.ACTION\_SEND  
   putExtra(Intent.EXTRA\_STREAM, contentUri)  
   type = "text/plain"  
  }  
    
  MainActivity.getActivity()?.startActivity(Intent.createChooser(shareIntent, null))

### 4.5.2. Create document intent

Pentru a salva fișierele într-o locație aleasă de utilizator, implementarea este următoarea:  
val intent = Intent(Intent.ACTION\_CREATE\_DOCUMENT).apply {  
 addCategory(Intent.CATEGORY\_OPENABLE)   
 type = "text/plain"   
 putExtra(Intent.EXTRA\_TITLE, fileName)  
}  
startActivityForResult(activity, intent, 1, null)

, unde fileName este denumirea propusă a fișierului, iar 1 este un ID care va fi folosit ulterior pentru a deosebi această cerere de altele. Apoi, se va copia fișierul original în noua locație.

## 4.6. Modificarea fișierelor pe server

Așa cum a fost prezentat, fișierele sunt stocate direct pe sistemul de fișiere, într-un director accesibil atât aplicației de server, cât și administratorului de sistem. Fiecare fișier este stocat în directorul care corespunde zilei în care fișierul a fost creat (informație stocată în baza de date). Fiecare fișier este denumit după ID-ul din baza de date (UUID) pentru a evita potențialele conflicte între denumiri și denumiri de fișiere ilegale (care denumiri sunt ilegale depinde, de obicei, de sistemul de fișiere, cât și de sistemul de operare folosit).

Atunci când se șterge un fișier, se verifică dacă directorul mai conține alte fișiere. Dacă nu, atunci se va șterge și directorul pentru a curăța structura de directoare.

O cerere de modificare va conține:

* **Textul**: fragmentul de text modificat
* **E adăugat**: adevărat dacă textul e adăugat la document, fals dacă e șters din document
* **Index**: poziția la care textul se va insera sau de la care se va șterge; la ștergere, poziția este cea la care se va termina ștergerea, adică capătul din stânga

Pentru a modifica, următorii pași sunt respectați:

* Se deschide fișierul  
  fs = File.Open(filePath, FileMode.OpenOrCreate, FileAccess.ReadWrite);
* Se transformă textul dat în bytes  
  byte[] diffBytes = Encoding.UTF8.GetBytes(diff);
* Dacă textul e adăugat:
  1. Se poziționează cursorul de fișier la indexul dat  
     fs.Seek(index, SeekOrigin.Begin);
  2. Se citesc bytes care sunt după poziția de inserare  
     byte[] remainingBytes = new byte[fs.Length - index];  
     await fs.ReadExactlyAsync(remainingBytes, 0, remainingBytes.Length);
  3. Se repoziționează cursorul la indexul dat inițial și se scriu datele deja existente  
     fs.Seek(index, SeekOrigin.Begin);  
     fs.Write(diffBytes);  
     await fs.WriteAsync(remainingBytes);
* Dacă textul e șters:
  1. Se poziționează cursorul la sfârșitul zonei de ștergere  
     int offset = index + diffBytes.Length;  
     fs.Seek(offset, SeekOrigin.Begin);
  2. Se citesc bytes care sunt după poziția de inserare  
     byte[] remainingBytes = new byte[fs.Length - index];  
     await fs.ReadExactlyAsync(remainingBytes, 0, remainingBytes.Length);
  3. Se repoziționează cursorul la începutul zonei șterse și se scriu datele deja existente  
     fs.Seek(offset - diffBytes.Length, SeekOrigin.Begin);  
     fs.SetLength(fs.Length - diffBytes.Length);  
     await fs.WriteAsync(remainingBytes);
* Se închide fișierul cu fs.Close();

# 5. Testare și concluzii

Aplicația a fost testată atât pe emulator, cât și pe un dispozitiv Android fizic (Samsung Galaxy A54, Android 14). De asemenea, pe emulator au fost testate diferite dimensiuni și diferite densități de pixeli (dpi) și s-a ajuns la concluzia că aplicația arată bine și funcționează corect majoritatea configurațiilor, chiar și în orientarea landscape. Aplicația răspunde rapid cererilor utilizatului și are un aspect modern datorită utilizării componentelor Material 3.

Testarea funcționalităților de Share și Create document au fost ambele testate pe dispozitivul fizic, aplicația trecând cu succes ambele teste. Funcțiile cloud au fost testate doar pe emulator, însă au funcționat foarte bine.

În concluzie, această aplicație și-a demonstrat eficacitatea și utilitatea în viața de zi cu zi a unui utilizator și poate fi folosită de publicul larg cu încredere.

# Bibliografie

[1] <https://apilevels.com/>

[2] <https://ktor.io/>

[3] <https://fonts.google.com/icons/>

[4] <https://developer.android.com/develop/ui/compose/navigation>