

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

PhotoBomb


propusă de

Gabriela Neculea

Sesiunea: iulie, 2023

Coordonator științific

Conf. Dr. Andreea-Valentina Arusoaie

Avizat,
Îndrumător lucrare de licență,
Conf. Dr. Andreea-Valentina Arusoaie.
Data: 22.06.2023 Semnătura: 


Declarație privind originalitatea conținutului lucrării de licență

Subsemnatul **Neculea Gabriela** domiciliat în **România, jud. Neamț, Sat Goșmani, Com. Români**, născut la data de **20 noiembrie 2001**, identificat prin CNP **6011120270031**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2023, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **PhotoBomb** elaborată sub îndrumarea doamnei **Conf. Dr. Andreea-Valentina Arusoaie**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului ei într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data: 22.06.2023

Semnătura: 

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **PhotoBomb**, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de informatică.

De asemenea, sunt de acord ca Facultatea de informatică de la Universitatea "Alexandru-Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Gabriela Neculea**

Data: 22.06.2023.....

Semnătura:.....

Cuprins

Motivație	2
Introducere	3
Contribuții	5
1 Aplicații similare	6
2 Arhitectura	8
2.1 Arhitectura client-server	9
2.2 Diagrame use-case	9
3 Detalii de implementare	11
3.1 "Front-end"-ul aplicației	11
3.1.1 Tehnologii utilizate	11
3.1.2 Descrierea implementării	13
3.2 "Back-end"-ul aplicației	20
3.2.1 Tehnologii utilizate	20
3.2.2 Design-ul bazei de date	22
3.2.3 Descrierea implementării	24
4 Ghidul utilizatorului	28
4.1 Navigarea în aplicație	28
4.2 Vizualizarea colecțiilor de imagini	29
4.3 Editarea unei imagini	31
4.4 Creare unui colaj	31
Concluzii	33

Motivație

Dintotdeauna oamenii au căutat o modalitate de a păstra amintirile intacte. Această necesitate poate fi satisfăcută prin intermediul fotografiilor. Fotografia a evoluat într-un mod uimitor de-a lungul secolelor, oferindu-ne capacitatea de a înregistra și de a împărtăși în moduri întotdeauna inventive viziunea noastră asupra lumii. Inventatorul francez Joseph Nicéphore Niépce a făcut prima fotografie realizată vreodată, „View from the Window at Le Gras”, în 1826 sau 1827. Această fotografie arată o imagine a ferestrei casei sale din Burgundia, Franța. În anii '90, apariția camerelor digitale a dus fotografia la un nou nivel. Vizualizare imaginilor era imediată, nemaifiind necesar procesul de dezvoltare. Tot atunci a apărut și nevoia de editare a imaginilor, astfel că, în 1987, Adobe a lansat Photoshop, un program de editare care a devenit un standard în timp.

În zilele noastre, telefoanele mobile au devenit principalele aparate de fotografiat pentru majoritatea oamenilor. De asemenea, popularitatea aplicațiilor de editare a fotografiilor a crescut într-un mod remarcabil, astfel încât aproape fiecare aplicație de socializare are în prezent funcționalitatea de editare și de încărcare de fotografii. Fotografia a devenit una dintre cele mai folosite forme de exprimare și de comunicare, existind astfel o multitudine de aplicații pentru editarea de imagini.

Motivația dezvoltării aplicației Photobomb a plecat de la ideea de a avea o soluție simplă care integrează pe de o parte editarea imaginilor și crearea de colaje, iar pe de altă parte gestionarea fotografiilor într-un mod simplu și creativ. În alegerea temei am fost inspirată de admirația mea pentru artă, arhitectură și frumos. Cred cu tărie că această pasiune se poate vedea în lumea digitală, unde fiecare fotografie poate fi considerată un tablou pregătit de artist. Prin aplicația dezvoltată sper să ajut utilizatorii să-și folosească creativitatea și să descopere frumusețea din jurul lor prin prisma propriilor fotografii.

Introducere

Fotografia este o formă de artă și o modalitate captivantă de a înregistra și păstra momentele din viața noastră. De-a lungul timpului, fotografia a evoluat într-un mod impresionant, devenind o modalitate indispensabilă de comunicare vizuală și de documentare a lumii înconjurătoare. Ea ne permite să transmitem emoții, fotografia având un impact semnificativ în mai multe domenii, precum jurnalism, publicitate, artă. Datorită erei în care trăim, oamenii sunt capabili să facă acest lucru datorită telefoanele inteligente care au revoluționat lumea fotografie. Cu toate acestea, deseori, fotografiile au nevoie de un pic de strălucire pentru a se ridica la standarde mai înalte. Astfel că, numărul aplicațiilor de editare și partajare de imagini a crescut exponențial în ultimii ani. În acest context, aplicația mea, „Photobomb”, oferă o soluție simplă și eficientă pentru gestionarea colecțiilor de fotografii, crearea de colaje și editarea imaginilor.

Lucrare de față se concentrează pe dezvoltarea unei aplicații mobile, numită Photobomb, care furnizează utilizatorilor o platformă completă de editare și gestionare a fotografiilor. Există deja pe piață diverse aplicații care au funcționalități multiple de editare de imagini sau de creare de colaje, precum Adobe Photoshop sau Adobe Lightroom, Pixlr sau VSCO, dar toate acestea ori necesită cunoștințe mai avansate pentru a putea fi utilizate, ori au funcționalități multe prea multe, care ne pot îngreuna în unele cazuri utilizarea. Astfel că, aplicația mea își propune să pună la dispoziția utilizatorilor un mediu ușor de folosit, pentru a putea edita și crea colaje în timp util. Această aplicație intuitivă și ușor de manevrat oferă utilizatorilor posibilitatea de a edita imagini, încărcând o imagine de pe dispozitiv sau realizând o captură foto, de a crea colaje expresive și de a-și organiza fotografiile create în galeria ușor de utilizat a aplicației.

Photobomb pune la dispoziție o gamă largă de instrumente de editare, printre care aplicarea de filtre, modificarea luminozității și a saturației, permitând utilizatorilor să transforme fotografiile obișnuite în adevărate opere de artă. Astfel că, prin intermediul acestei aplicații, utilizatorii nu sunt doar consumatori de conținut vizual,

ci pot deveni chiar creatori, având la dispoziție instrumentele necesarii pentru a-și exprima creativitatea într-un mod unic și artistic. Mai mult, Photobomb încorporează și o funcție de realizare a colajelor, asigurând astfel un mod unic și plin de inovație pentru utilizatori de a-și expune fotografiile. Aceste ansambluri de imagini pot fi, de asemenea, structurate în colecții pentru a ușura localizarea lor în viitor.

Lucrare de față este structurată în patru capitole astfel: primul capitol, "Aplicații similare", unde este realizată o comparație între aplicația de față și alte două aplicații asemănătoare de pe piață, cel de-al doilea capitol, "Arhitectura", prezintă arhitectura de bază care a fost utilizată în implementare și funcționalitățile principale ale aplicației printr-o diagrama use-case, al treilea capitol, "Detalii de implementare", face o trecere în revistă a tehnologiilor utilizate, atât pe partea de frontend cât și pe partea de backend, și descriere procesul implementării celor două componente, ultimul capitol, "Ghidul utilizatorului", prezintă manualul de utilizare al aplicației, cu exemple și explicații pentru fiecare funcționalitate în parte.

Contribuții

Există deja o varietate de aplicații care oferă funcționalități diverse pentru manipularea imaginilor, astfel că ideea aplicației nu este una originală, fiind o combinație obținută în urma studierii a mai multor software-uri populare de acest tip. În schimb, structura, implementarea și design-ul estetic al aplicației au la bază toate ideile proprii, care au fost șlefuite cu ajutorul a mai multor librării și instrumente care au dus la obținerea unei aplicații stabile și complet funcționale. Dintre acestea pot aminti: Ngrok, utilizat pentru a expune server-ul local către internet, Amazon S3, care a fost folosit pentru a stoca imaginile încărcate de utilizatori, astfel, atunci când un utilizator încarcă o imagine în aplicație, aceasta este trimisă la un bucket S3, unde este stocată în mod sigur și eficient, biblioteca OpenCV, folosită pentru implementarea diverselor funcționalități de editare de imagini.

Cu toate acestea, am contribuit personal la crearea unei interfețe de utilizator intuitive și atractive, îmbinând funcționalitatea cu estetica pentru a asigura o experiență de utilizare plăcută și eficientă. În plus, am implementat structura bazei de date, concepând și dezvoltând endpoint-urile necesare pentru o comunicare fluidă cu clientul.

Prin realizarea acestei aplicații am avut ocazia de a pune împreună și de a îmbunătăți cunoștințele în domeniul dezvoltării de aplicații mobile, platforma Android și limbajul de programare Kotlin, limbajul de programare Python, prin intermediul "micro-framework"-ului Flask, dar și a cunoștințelor de baze de date.

Capitolul 1

Aplicații similare

Pot spune că ideea realizării acestei aplicații s-a născut în urma utilizării de-a lungul anilor a mai multor aplicații de editare de imagini, precum Adobe Photoshop, Adobe Lightroom, Pixlr etc. Având în vedere faptul că aplicația de față este una mobile pentru Android, în continuare voi prezenta câteva aplicațiile mobile similare, care au fost semnificative în dezvoltarea aplicației mele.

VSCO: Photo & Video Editor:

VSCO (sau VSCO Cam) este o aplicație populară de fotografiere și editare care este disponibilă pe dispozitivele Android și iOS. Această aplicație are o multitudine de funcționalități, de la editarea fotografiilor, crearea de clipuri video, conceperea de colaje până la funcționalitatea de postare a fotografiilor pentru a putea fi văzute de alți utilizatori, astfel că putem spune că aplicația este mai mult decât o aplicație de editare, fiind și o aplicație de socializare, cu accent pe interacțiunea dintre utilizatori. Din acest punct de vedere pot afirma că aplicația mea este diferită de cea în cauză, având strict rolul de editare, creare de colaje și de gestionare a imaginilor, fiind ușor de înțeles și utilizat, interacțiunea dintre utilizatori neexistând. Deși cu mai puține funcționalități, aplicația mea este mai justă și mai prietenoasă cu utilizatorii neexperimentați, oferind exact funcționalitățile principale pe care un utilizator le-ar putea avea de la o aplicație de editare foto.

Collage Maker - Photo Editor:

Collage Maker – Photo Editor este o aplicație populară care le permite utilizatorilor să facă colaje personalizate de fotografii pe telefoanele lor mobile. Aplicația, care este disponibilă pe sistemele de operare Android și iOS, oferă o varietate de instrumente utile pentru a îmbunătăți și personaliza fotografiile.

Asemănarea dintre această aplicație și aplicația mea constă în modulul de creare de colaje pe care ambele aplicații îl expun utilizatorilor. Collage Maker are o interfață ușor de folosit, deoarece la imediata deschidere ești deja direcționat spre pagina de alege a fotografiilor pentru colaj. Odată alese fotografiile, poți alege layout-ul colajului pe care vrei să îl creezi și îl poți schimba în orice moment al creării colajului, acesta funcționalitate aducând un plus funcționalităților. Dar pe lângă crearea de colaje, aplicația nu pare să mai aibă alte funcționalități care să îmbunătățească experiența utilizatorilor, ea neavând niciun instrument de editare a imaginii în sine, de schimbare a luminozității sau contrastului și nici posibilitatea de organizare a imaginilor în cadrul aplicației. Din acest motiv, consider că aplicația de față este mai restrânsă în funcționalități față de aplicația mea, însă este mai dezvoltată pe partea de creare a colajelor, având mai multe template-uri și o abordare mai dinamică.

Capitolul 2

Arhitectura

Arhitectura unei aplicații mobile se referă la modul în care sunt integrate și organizate diferitele componente ale aplicației astfel încât să fie atinse obiectivele dorite. În general, arhitecturile bune își propun să ofere structuri care sunt modulare, scalabile și ușor de întreținut, asigurând în același timp performanța optimă și o experiență plăcută pentru utilizatori.

Iată câteva aspecte importante pe care ar trebui să le îndeplinească o arhitectură pentru a oferi utilizatorilor o experiență adecvată:

- Interfață cu utilizatorul(UI): intuitivă, responsive, design simplu și curat pentru a le permite utilizatorilor să navigheze cu ușurință în aplicație.
- Securitatea: protejarea datelor personale ale utilizatorilor, securitatea autentificării, criptarea datelor cu caracter personal, gestionarea sesiunilor și protejarea aplicației de atacuri și vulnerabilități [9].
- Accesul la date: modul în care aplicația interacționează cu datele și sursele de stocare, cum ar fi bazele de date, serviciile web sau API-urile. Ar trebui să existe un strat de acces la date care să permită obținerea, crearea, actualizarea și ștergerea eficientă a datelor.

În aplicația pe care o prezentăm am integrat aceste principii printr-o arhitectură compusă din două componente: clientul Android și serverul Flask. Clientul Android oferă o interfață cu utilizatorul rapidă și simplă care facilitează editarea fotografiilor și crearea de colaje. Serverul Flask gestionează interacțiunile dintre date și sursele de stocare, asigurându-se că toate operațiunile de acces la date sunt efectuate rapid și

sigur. În plus, aplicația integrează Amazon S3, un serviciu de stocare în cloud oferit de Amazon, pentru ca utilizatorii să poată stoca fotografiile într-un mod sigur și eficient.

2.1 Arhitectura client-server

Pentru dezvoltarea acestei aplicații am utilizat o arhitectura de tip server-client datorită eficienței, scalabilității și flexibilității pe care aceasta le oferă. Unul dintre punctele forte ale acestei arhitecturii îl reprezintă separarea clară dintre logica de afaceri și prezentarea datelor către utilizator, ceea ce face organizarea și menținerea aplicației mult mai ușoară. Serverul gestionează logica de afaceri și stocarea datelor pentru a se asigura că datele sunt procesate corect și disponibile în orice moment. În același timp, clientul se concentrează pe prezentarea datelor și interacțiunii cu utilizatorul; cere informațiile de la server și le prezintă utilizatorului într-un mod util și concis.

Este de menționat că implementarea unei arhitecturi client-server facilitează crearea și actualizarea aplicației. Putem implementa noi funcții sau îmbunătățiri la fiecare componentă în mod independent, deoarece clientul și serverul sunt construiți și găzduiți separat.

În primul rând, partea de client, dezvoltată în Android, este componenta cu care interacționează utilizatorii în mod direct. Aceasta este responsabilă cu solicitarea datelor de la server și cu prezentarea lor către utilizatori într-un mod concis. Astfel că, ori de câte ori este nevoie, clientul trimite cereri către server, serverul îi trimite un răspuns, iar clientul afișează aceste date primite într-o manieră prietenoasă utilizatorului.

În al doilea rând, partea de server, care a fost realizată în Python, este responsabilă de gestionarea și furnizarea datelor clientului. Când serverul primește o cerere de la un client, procesează cererea, interoghează baza de date sau efectuează alte operații necesare, apoi trimite un răspuns clientului. Dacă, de exemplu, un client solicită o listă a colecțiilor sale, serverul va interoga baza de date pentru a obține informațiile și apoi le va returna clientului.

2.2 Diagrame use-case

În cele ce urmează, voi prezenta modul în care un utilizator poate interacționa cu aplicația propusă. În primă fază, nefiind autentificat, utilizatorul nu are acces decât la pagina de autentificare (login) și la cea de înregistrare (register). După autentificare,

acesta își poate vizualiza colecțiile, acestea fiind afișate în pagina principală a aplicație, dacă dorește să vadă o colecție în detaliu poate face click pe aceasta, iar aplicația îi va afișa imaginile din colecția respectivă într-o galerie. Utilizatorul are la dispoziție o varietate de opțiuni de editare atunci când alege să editeze o imagine; fiecare dintre aceste opțiuni are setări diferite pe care le poate modifica. De exemplu, atunci când se adaugă un filtru, acesta are capacitatea de a selecta dintre diferitele filtre care sunt disponibile.

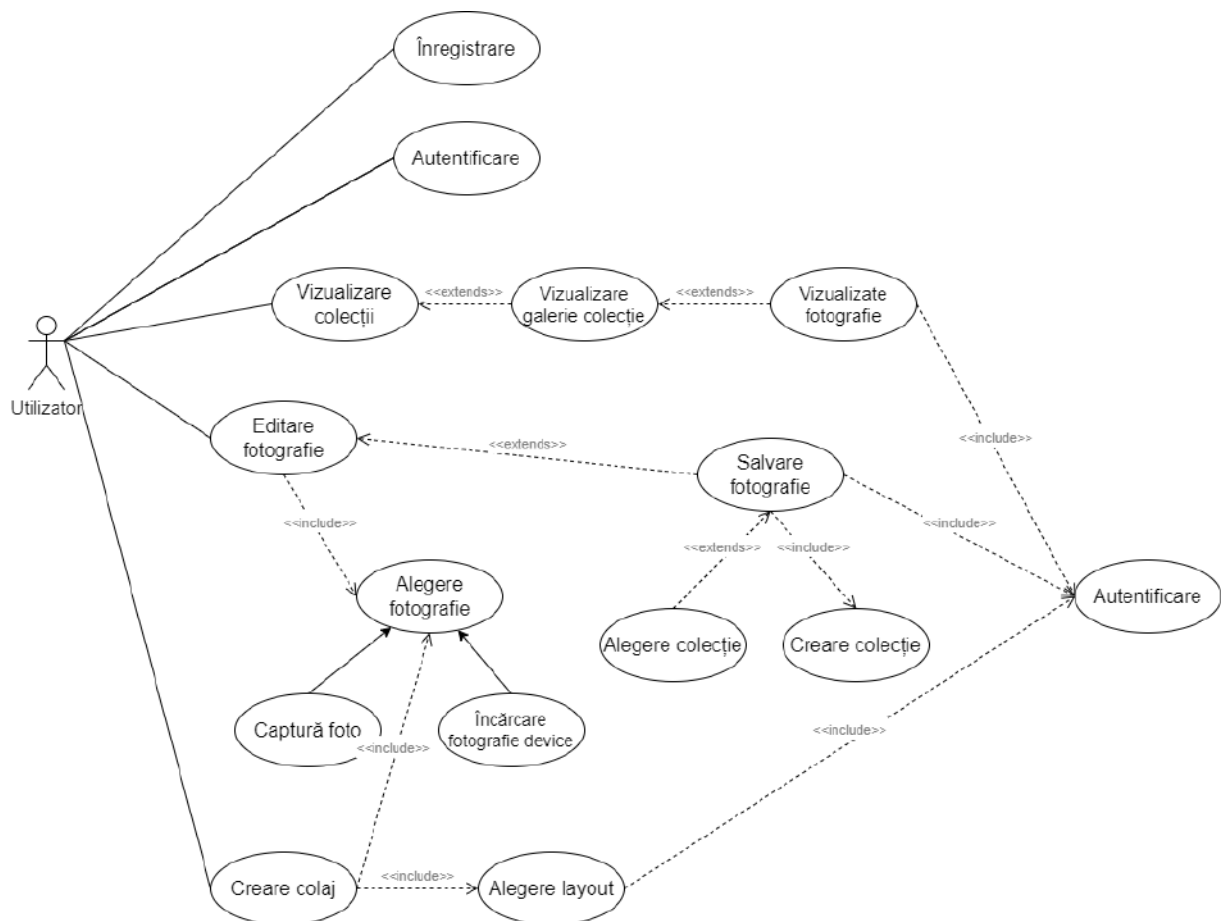


Figura 2.1: Diagrama use-case

Utilizatorul poate salva imaginea editată într-o colecție existentă sau poate crea o nouă colecție după ce a finalizat editarea. Utilizatorul începe crearea de colaje prin alegerea unui template de colaj. După aceea, poate adăuga imagini în fiecare cadru al template-ului de colaj selectând direct din galerie. Acestea sunt doar câteva dintre scenariile de utilizare pe care le poate avea un utilizator al aplicației. Toate aceste use-case-uri sunt reprezentate în diagrama de la Figura 2.1.

Capitolul 3

Detalii de implementare

Acest capitol al lucrării va conține prezentarea procesul de realizare a aplicației, accentuând tehnologiile utilizate, deciziile de design și strategiile de implementare adoptate. Soluția propusă implică o arhitectură client-server, cu un server Flask, care este responsabil de managerierea datelor și care va expune un API prin care clientul va avea acces la aceste date, și un client Android, care va ține locul interfeței grafice a aplicației.

Vom începe prin prezentarea în detaliu a interfeței aplicației, incluzând atât tehnologiile utilizate cât și detaliile tehnice de implementare. Vom sublinia modul în care am realizat o interfață ușor de utilizat, care oferă utilizatorilor o experiență plăcută și eficientă. Ulterior, vom trece la modul în care am realizat serverul Flask și la cum acesta facilitează comunicarea dintre client și server.

3.1 "Front-end"-ul aplicației

După cum am precizat și în capitolul anterior, pentru partea de interfață am utilizat platforma Android, mai exact, am lucrat cu limbajul de programare Kotlin. Am făcut această alegere din simplitatea și flexibilitatea cu care vine platforma, dar și deoarece aveam nevoie ca aplicația să aibă funcționalitatea de a captura și încărca fotografii, iar într-un mediu mobile, acest lucru este mult mai ușor.

3.1.1 Tehnologii utilizate

Android este un sistem de operare pentru mobile dezvoltat de Google și care este utilizat pe o varietate de dispozitive, de la telefoane mobile la tablete și dispozitive

IoT. Android deține o cotă de piață semnificativă în industria dispozitivelor mobile. Potrivit datelor recente, Android are o cotă de piață de aproximativ 72-75% la nivel global, fiind lider în acest domeniu. Am ales să lucrez pe partea de mobile, mai exact Android, datorită flexibilității și diversității pe care această platforma le oferă, cât și datorită accesibilității, fiind prezent într-o variată gamă de dispozitive și prețuri, acesta este mai accesibil global, față de alte sisteme de operare mobile.

Ca limbaj de implementare am ales Kotlin, limbaj dezvoltat de JetBrains, companie de software specializată pe dezvoltarea de instrumente și medii de dezvoltare integrate. Kotlin este concis, oferă o sintaxă expresivă și este interoperabil cu Java, care rulează pe JVM (Java Virtual Machine). Fiind adoptat pe scară largă pentru dezvoltarea aplicațiilor, alegerea Kotlin aduce avantaje precum o productivitate mai mare, mai puține erori și o experiență de dezvoltare mai plăcută.

Pentru comunicarea dintre client și server am utilizat biblioteca Retrofit, care oferă o abordare simplă și eficientă pentru comunicarea cu serverul HTTP. Retrofit propune o interfață intuitivă pentru a defini servicii REST, folosindu-se de adnotări pentru a specifica endpoint-urile, metodele HTTP, parametrii și răspunsurile. Aceasta se bazează pe biblioteca OkHttp pentru gestionarea cererilor. Retrofit oferă suport nativ pentru apelurile asincrone prin intermediul tipurilor de returnare precum `Call<T>` și `Deferred<T>`.

În dezvoltarea aplicației noastre, o componentă esențială pe partea de frontend o reprezintă crearea unei interfețe Retrofit care definește metodele HTTP cu ajutorul cărora se stabilește comunicarea cu serverul. Acest element asigură legătură dintre funcționalitățile aplicației și datele pe care serverul le stochează în baza de date. În interfață am definit o serie de metode pentru autentificare, înregistrare, gestionarea colecțiilor de imagini, încărcarea imaginilor și primirea datelor de la server, toate acestea facilitând experiență generală a utilizatorilor în aplicație.

În urma studierii mai multor abordări, am ajuns la concluzia că cea mai bună modalitate de a realiza operațiile pe imagini, anume cele din partea de editare, ar fi ca aceste operații să fie efectuate pe partea de client. Unul dintre motivele acestei alegeri este faptul că operațiile erau destul de ușoare, iar comunicarea cu serverul pentru efectuarea lor doar ar fi îngreunat întreg procesul. Alt motiv ar fi ușurința utilizării kit-ului de dezvoltare software OpenCV SDK, ce conține biblioteci și instrumente pentru a dezvolta aplicații Android care utilizează funcționalitățile de prelucrare de imagini.

3.1.2 Descrierea implementării

După cum am amintit mai sus, am definit o interfață utilizând biblioteca Retrofit, care facilitează conectarea la serviciile API-ului server-ului Flask. Interfața conține toate metodele necesare comunicării dintre client și server. O metodă a acestei interfețe are, în primul rând, o adnotare HTTP, specifică Retrofit, care indică că metoda respectivă va fi utilizată pentru a efectua un request HTTP, tipul acestuia poate fi: GET, POST, PUT, DELETE. Între paranteze, după verb-ul HTTP, vom găsi endpoint-ul pe care îl va apela acea metodă. Pentru exemplificare, puteți găsi în codul de mai jos câteva metode ce aparțin acestei interfețe.

```
@Multipart
@POST("/collections/{collectionId}/images")
fun addImageToCollection(
    @Path("collectionId") collectionId: Int,
    @Part image: MultipartBody.Part,
    @Part("desc") desc: RequestBody,
    @Header("Authorization") apiKey: String
): Call<AddImageResponse>

@GET("collections/{collection_id}/images")
fun getImagesFromCollection(
    @Header("Authorization") token: String,
    @Path("collection_id") collectionId: Int
): Call<List<Image>>

@DELETE("/collections/{collection_id}/images/{image_id}")
fun deleteImage(
    @Path("collection_id") collectionId: Int,
    @Path("image_id") imageId: Int,
    @Header("Authorization") apiKey: String
): Call<Void>
```

Spre exemplu, adnotarea "@Path" indică faptul că parametrul ce urmează a fi declarat ar trebui să fie utilizat pentru a înlocui placeholder-ul specificat între parantezele din URL-ul cererii HTTP. Adnotarea "@Header" indică faptul că parametrul ar trebuie

să fie utilizat ca valoare pentru header-ul "Authorization" al cererii HTTP.

O componentă esențială a oricărei aplicații Android o reprezintă activitățile. Acestea constituie părți esențiale ale interfeței, având rolul de ecran principal cu care utilizatorii interacționează. Fiecare activitate are asociat un layout(o interfață grafică) care definește modul în care sunt afișate informațiile utilizatorului. O altă componentă importantă a unei aplicații Android o reprezintă fragmentele. Un fragment este o parte independentă a unei interfețe, care poate fi adăugată la o activitate, fiind văzută că o "sub-activitate". Aceste fragmente pot avea propriul lor layout, ciclu de viață și comportament. Folosirea fragmentelor oferă o serie de avantaje, precum: reutilizabilitate - fragmentele, odată create, pot fi utilizate în mai multe activități simultan, acestea neinteracționând între ele; evitarea duplicării codului, gestionarea complexității unei activități - dacă activitatea are funcționalități multiple, acestea pot fi împărțite în fragmente, codul fiind mai ușor de gestionat.

În cadrul unei aplicații Android pot exista mai multe activități, fiecare fiind responsabilă pentru realizarea câtorva funcționalități din aplicație. Astfel că, aplicația de față este alcătuită din următoarele activități mai importante: activitatea de autentificare, activitatea de creare cont, activitatea principală, care este completată de mai multe fragmente, activitatea de creare a unui colaj, activitatea de editare a unei imagini.

Activitatea principală, cea care este afișată după ce utilizatorul se autentifică, conține mai multe meniuri. "BottomNavigationView" este o bară de navigare situată în partea de jos a ecranului care permite navigarea rapidă între diferitele nivele ale aplicației. Aceasta folosește un meniu definit în fișierul "bottom_navigation_menu.xml". Acesta este alcătuit din trei butoane, primul, ce este pentru galeria aplicație, cel de-al doilea, pentru încărcarea unei imagini și începerea procesului de editare, și ultimul, pentru funcționalitatea de creare a unui colaj. De asemenea, această activitate mai conține și un meniu glisant, care se activează prin apăsarea butonului din partea stângă sus, în acest meniu utilizatorul poate naviga spre profilul său, își poate modifica setările contului sau se poate deconecta.

De asemenea, o parte foarte importantă din această activitate o reprezintă fragmentul de afișare a colecțiilor unui utilizator, acest fragment ocupând mai bine de 80% din activitate. În momentul în care aplicația este deschisă, se execută funcția "onCreate", iar deoarece în aceasta metodă creăm și atașăm fragmentul pentru colecții, este trimisă o cerere HTTP de tip GET pentru a obține lista de colecții ale utilizatorului

autentificat în acel moment în aplicație. Astfel, în acest fragment, sunt afișate toate colecțiile utilizatorului, având un thumbnail cu ultima poza salvată în acea colecție și numărul de fotografii din acea colecție. Apăsarea oricărei dintre colecții va declanșa crearea altui fragment, și anume a fragmentului ce se ocupă de afișarea fotografiilor din colecție. La fel ca la fragmentul pentru colecții, acest fragment va efectua un request de tip GET către server pentru a prelua lista imaginilor din acea colecție. Imaginile sunt afișate într-o galerie, iar la apăsarea oricăreia dintre ele, imaginea va fi afișată în full screen. De asemenea, în fragmentul de full screen a imaginii, vom avea două butoane în partea de jos, un buton de delete și unul de edit, cel din urmă declanșând o activitate de editare cu imaginea respectivă. Operația de delete se realizează printr-o cerere de tipul DELETE către server și are ca parametrii id-ul colecției și id-ul imaginii pe care o ștergem. La apăsarea celui de-al doilea buton din meniul de jos, "Upload photo", ni se va deschide un meniu de tip "pop-up" unde va trebui să alegem dacă dorim să încărcăm o imagine de pe dispozitiv, din galerie, sau dacă dorim să realizăm o captură foto, pentru acest lucru am utilizat biblioteca "ImagePicker". Așa cum spune și numele, această bibliotecă permite selectarea imaginilor din galeria dispozitivului sau capturarea unei imagini noi cu camera dispozitivului. După alegerea imaginii, va începe activitatea de editare.

În ceea ce privește componenta interfeței grafice a activității principale, am utilizat numeroase "view"-uri disponibile pe platforma Android. "View"-ul reprezintă o componentă de bază a interfeței cu utilizatorul, astfel că fiecare buton, imagine, listă, text cu care interacționează utilizatorul este un View. Pentru afișarea colecțiilor în "CollectionFragment" am utilizat un "RecyclerView", care lucrează împreună cu un "CollectionAdapter" [7]. Un "Adapter" în Android, este o punte între "RecyclerView" și datele care sunt afișate în acesta. Fiecare item (în cazul de față, o colecție) din RecyclerView este reprezentat de un "ViewHolder" ce conține o referință a item-ului și metadatele despre locul său în RecyclerView. "CollectionAdapter" este responsabil cu crearea view-urilor pentru fiecare item și cu înlocuirea acestora cu noile date atunci când este cazul. "RecyclerView"-ul dispune "ViewHolder"-urile în container. Acest proces se face în funcție de LayoutManager setat pe RecyclerView, în cazul de față am utilizat un "GridLayoutManager" care va dispune colecțiile în două coloane [3].

Pentru a urma principiul separării responsabilităților, în cadrul fragmentelor activității principale am utilizat ViewModel-uri. În Android, un ViewModel este o clasă creată pentru stocarea și gestionarea datelor legate de interfața cu utilizatorul. De

exemplu, în fragmetul colecțiilor, utilizez un "CollectionViewModel", acesta descrie o metodă, "fetchCollections" care este utilizată pentru a prelua colecțiile de imagini ale utilizatorului prin apelarea endpoint-ului creat pentru asta. Mai apoi, datele preluate de la server sunt stocate în variabila "_collections", care este o valoare LiveData. Acest lucru înseamnă că orice modificare a acestei valori vă declanșă o actualizare a interfeței cu utilizatorul. O caracteristică semnificativă a utilizării ViewModel-urilor este că acestea fac posibilă separarea logicii de afișare de logica de manipulare a datelor. Acest lucru permite fragmentelor și activităților să rămână cât mai simple și mai curate, în timp ce ViewModel-urile se ocupă de sarcinile dificile.

Cea de-a doua activitate, activitatea ce se ocupă de editarea imaginii, conține, în primul rând, un PhotoView pentru afișarea imaginii către utilizator. Un PhotoView este un ImageView personificat astfel încât să permită și funcționalitatea de zoom pe imagine. În partea de jos a activității avem un meniu, ce conține toate uneltele ce pot fi utilizate pentru editarea imaginii. Avem butonul pentru filtre, care, odată apăsăat va declanșă afișarea unui fragment ce conține o listă de butoane cu denumirile filtrelor ce pot fi aplicate pe imagini. Toate aceste operații pe imagini sunt realizate cu ajutorului bibliotecii OpenCv.

Dintre filtre amintesc: "Sepia" - culoarea sepia însăși este o nuanță de maro deschis cu o tentă de roșu sau portocaliu. Când este aplicat unei fotografii, filtrul sepia transformă toate culorile în tonuri de maro, de la deschis la închis, în funcție de intensitatea originală a culorii; ideea de bază a acestui filtru este de a înmulți matricea imaginii cu un kernel creat special pentru a obține efectul de "Sepia", kernelul are următoarele valori [2]:

```
[[0.393, 0.769, 0.189, 0.0],  
 [0.349, 0.686, 0.168, 0.0],  
 [0.272, 0.534, 0.131, 0.0],  
 [0.0, 0.0, 0.0, 1.0]]
```

Un alt filtru prezent în lista de filtre este „Vignette”. Acesta reprezintă o tehnică de post-procesare a imaginilor ce face că perimetrul imaginii să fie acoperit de o umbră sau o lumină diminuată, pe când centrul rămâne clar. În general, atunci când este folosit pentru o imagine, acest filtru atrage atenția către centrul imaginii și îi conferă un aspect artistic sau nostalgic. Pentru a crea efectul de umbră la marginea imaginii sunt create două kerneluri Gaussiene, unul pentru axa x și unul pentru axa y. Ma-

tricile de valori cunoscute sub numele de kerneluri Gaussiene sunt calculate pe baza distribuției normale (Gaussiene) [1], care este mai mare în centru și scade pe măsură ce ne îndepărtăm. În continuare, aceste două kerneluri Gaussiene sunt înmulțite pentru a forma un kernel 2D. Obținându-se o matrice cu valori mari în centru și valori mici la margini, ceea ce corespunde efectului de „vignette” pe care îl dorim. Pentru a putea fi aplicat imaginii, acest kernel este normalizat la valori între 0 și 255, care sunt valorile standard pentru imagini în tonuri de gri. În continuare, kernelul este convertit la tipul de date CV_8UC1 (8-biți, 1 canal, adică tonuri de gri), iar culoarea este transformată din tonuri de gri în RGB (Red, Green, Blue) și apoi în RGBA (Red, Green, Blue, Alpha) pentru a se potrivi cu imaginea inițială, astfel că se obține o mască Vignette, care se înmulțește cu matricea imaginii, fiecare pixel al imaginii este înmulțit cu pixelul corespunzător al măștii. Rezultatul este o imagine clară în centru și mai întunecată la margini, deoarece masca are valori mari în centru și mici la margini.

Toate aceste operații pentru filtre sunt efectuate într-un context IO al unei corutine, pentru a permite funcției să ruleze în paralel cu alte operații și pentru a nu bloca thread-ul principal. Corutinele sunt o caracteristică a limbajului Kotlin care ne permit să scriem cod asincron într-un stil de programare liniar, fără a bloca de fapt threadul în care rulează. Fiecare corutină este executată într-un “CoroutineScope” specific. Acesta este responsabil pentru gestionarea duratei de viață a corutinelor. Când creăm o corutină, trebuie să specificăm un “CoroutineDispatcher” pentru a alege pe ce thread sau pool de threaduri va rula. De exemplu, “Dispatchers.IO” este un dispatcher optimizat pentru operațiuni care blochează I/O, cum ar fi operațiunile de rețea, citirea și scrierea de fișiere sau, în cazul aplicației de față, procesarea imaginilor. “CoroutineScope” se ocupă de “când” se execută o corutină (în sensul că gestionează durata de viață a corutinei), în timp ce “CoroutineDispatcher” se ocupă de “unde” se execută (adică pe care thread).

O altă unealtă de editare prezentă în aplicație este reglarea luminozității și saturației imaginii. În cele ce urmează voi prezenta doar una dintre ele, și anume saturația. La apăsarea butonului pentru saturație, fragmentul anterior va fi înlocuit de un fragment ce va afișa un seekbar pentru setarea saturației. Inițial, acest seekbar va fi poziționat în centrul seekbar-ului, crescând procentul, saturația va fi mai contrastată, scăzând, aceasta se va diminua, la 0% imaginea devenind alb-negru. Funcția “adjustSaturation”, care modifică saturația unei imaginii, primește ca parametrii bitmap-ul imaginii și procentajul saturației. Această funcție se bazează pe calculele utilizate în ma-

nipularea imaginilor în spațiul de culoare HSV, astfel că, primul lucru pe care îl face funcția este de a transforma matricea obținută din bitmap, dintr-o reprezentare RGB într-una HSV, acronim pentru Hue (nuanță), Saturation (saturație) și Value (valoare). Următorul pas este împărțirea elementelor HSV în canale diferite. În consecință, vom avea trei matrici distincte, fiecare reprezentând un element de culoare: nuanță, saturație și valoare. Acest lucru este realizat cu ajutorul funcției "split" din OpenCV. Ne concentrăm pe al doilea canal deoarece ne dorim modificarea saturației. Pentru a permite operațiuni matematice mai complexe, canalul de saturație este transformat într-o matrice de tip float. Atunci când valorile tuturor pixelilor sunt înmulțite cu procentul de ajustare dorit, care este reprezentat de parametrul "percentage" al funcției, se modifică saturația imaginii. Matricea de saturație este transformată înapoi într-o matrice pe 8 biți înainte de a fi reintrodusă în imaginea HSV, iar în final, convertim imaginea înapoi în spațiul de culoare RGB.

De asemenea, partea de editare a imaginii are și funcționalitatea de adăugare a textului pe imagine. Acest lucru a fost realizat prin crearea unui clase, Draggable-TextView, care extinde clasa AppCompatActivity și creează un TextView personalizat ce poate fi mutat, rotit și redimensionat cu gesturi de atingere ale ecranului. Clasa conține un "ScaleGestureDetector", utilizat pentru identificarea și răspunderea la gesturi de tip "pinch to zoom", ce permite redimensionarea textului. În plus, Draggable-TextView poate fi și rotit, grație unei alte componente cheie, "RotationGestureDetector". Acesta calculează unghiul dintre două puncte de atingere pe ecran și comunică acest unghi către un "RotationListener", care răspunde prin rotirea efectivă a elementului TextView.

```
class RotationGestureDetector
    (private val mListener: OnRotationGestureListener) {
    var angle: Float = 0.toFloat()
    private var mPrevAngle: Float = 0.toFloat()
    interface OnRotationGestureListener {
        fun onRotation(rotationDetector: RotationGestureDetector)
    }
    fun onTouchEvent(event: MotionEvent): Boolean {
        when (event.actionMasked) {
            MotionEvent.ACTION_POINTER_DOWN ->
```

```

        mPrevAngle = calculateAngle(event)
        MotionEvent.ACTION_MOVE ->
        if (event.pointerCount > 1) {
            angle = calculateAngle(event) - mPrevAngle
            mListener.onRotation(this) } }

        return true
    }

    private fun calculateAngle(event: MotionEvent): Float {
        val xTouch = event.getX(1) - event.getX(0)
        val yTouch = event.getY(1) - event.getY(0)
        return (atan2(yTouch.toDouble(), xTouch.toDouble())
            * (180 / Math.PI)).toFloat()
    }
}

```

În plus, culoarea textului adăugat poate fi setată. Dialogul de alegere a culorii este creat folosind un "ColorPickerDialogBuilder", care este o clasă din biblioteca Flask ColorPicker. De asemenea, după ce un text a fost adăugat, poate fi șters sau modificat după preferințe.

Activitatea de editare a imaginii are în partea superioară un meniu cu două butoane, unul de "Undo" și unul de "Save". Butonul de "Undo" inversează ultima acțiune de editare efectuată de utilizator. Funcționalitatea de "undo" este o parte esențială a designului interfeței cu utilizatorul pentru orice aplicație de editare, deoarece permite utilizatorilor să experimenteze și să încerce diferite efecte sau modificări, știind că pot reveni ușor la starea anterioară a imaginii. Butonul pentru salvare, odată apăsător, declanșează apariția unui dialog ce conține lista de colecții ale utilizatorului, prin selectarea unei colecții și apăsarea butonului save, imaginea este salvată în colecție. Acest lucru se realizează prin apelarea unui cereri HTTP de tip POST, ce trimite către serverul Flask imaginea și id-ul colecției în care trebuie salvată. Clasa "CollectionDialogFragment" este utilizat pentru afisarea listei cu colecțiile de imagini ale utilizatorului. O cerere HTTP de tip GET este trimisă către server pentru a obține colecțiile. În partea inferioară a acestui dialog personalizat se află un RecyclerView și două butoane: de salvare și de adăugare a unei noi colecții. Când se apasă butonul pentru adăugare a unei noi colecție, se deschide un dialog care permite utilizatorului

să introducă un nume pentru colecției. Dacă numele este corect, se trimite o altă cerere HTTP pentru crearea colecției, iar lista este modificată pentru a include noua colecție.

Utilizarea limbajului Kotlin și Android Studio, precum și utilizarea bibliotecilor externe pentru comunicare cu backend-ul, cum ar fi Retrofit, au fost toate componente ale proiectării interfeței. Funcțiile de bază ale aplicației Android, cum ar fi navigarea între ecrane și afișarea datelor, au fost implementate cu succes, ceea ce o face aplicația de față una funcțională și simplă de utilizat.

3.2 "Back-end"-ul aplicației

Backend-ul aplicației a fost realizat în Python, împreună cu "microframework"-ul Flask. El servește ca un "pod" între client (aplicația Android) și datele stocate în baza de date. Acesta este responsabil de stocarea și gestionarea datelor utilizatorilor și imaginilor, dar și de autentificarea utilizatorilor și de furnizarea de servicii către aplicația client. Comunicarea dintre backend-ul Flask și aplicația Android se realizează prin cereri și răspunsuri HTTP. Cererile clientului sunt compuse din metoda HTTP (GET, POST, PUT, DELETE), un URL și, în unele cazuri, un corp al cererii.

Serverul Flask utilizează SQLAlchemy, un ORM (Object-Relational Mapping) pentru Python, pentru a putea realiza interacțiunea cu baza de date. Prin intermediul modelelor definite, serverul poate realiza operații de creare, citire, actualizare și ștergere (CRUD) asupra datelor.

3.2.1 Tehnologii utilizate

Am folosit o varietate de instrumente și biblioteci în procesul de dezvoltare pentru a crea un server puternic și robust. Datorită utilizării acestor resurse a fost posibilă o dezvoltare mai rapidă și mai eficientă. De asemenea, acestea au îmbunătățit funcționalitățile și performanța finală a serverului.

- a) **Python 3.6:** numeroasele beneficii oferite de Python au determinat alegerea mea de a folosi acest limbaj pentru dezvoltarea serverului. Sintaxa simplă și ușor de înțeles a limbajului facilitează o înțelegere rapidă a codului. Pentru a simplifica și accelera procesul de dezvoltare, o varietate de funcționalități și biblioteci ale sale pot fi utilizate. Acesta dispune de o multitudine de biblioteci și funcționalități care ușurează procesul de dezvoltare.

- b) **Flask**: am ales sa utilizez acest micro-framework in Python datorita abordarii sale flexibile si simpliste care mi-a dat libertatea de a modela structura si functionalitatea serverului in functie de nevoile aplicatiei. Totodata este o alegere excelenta pentru dezvoltarea de API-uri REST, avand un suport nativ pentru cereri si raspunsuri HTTP [4].
- c) **SQLAlchemy**: SQLAlchemy este un instrument ORM (Mapping Object-Relational) pentru Python care oferă o intr-un mod abstract interacțiunea cu bazele de date relaționale. ORM-urile permit lucrul cu baze de date într-un mod orientat-obiect, ceea ce accelereaza productivitatea și calitatea codului. Daca nu foloseam SQLAlchemy ar fi trebuit sa scriu manual interogările SQL si sa gestionez conexiunea la baza de date. Acest lucru ar fi fost costisitor si ar fi putut duce mult mai usor la posibilitatea de a avea erori.
- d) **Amazon S3**: am utilizat Amazon S3 pentru a stoca imaginile fiecărui utilizator. Deoarece imaginile pot fi voluminoase și pot ocupa mult spațiu de stocare, Amazon S3 a fost o soluție ideala pentru gestionarea lor. Amazon S3 este un serviciu de stocare oferit de Amazon Web Services (AWS). Fiecare element stocat este considerat un obiect, acestea fiind împărțite în "buckets", un fel de foldere sau directoare. Fiecare obiect are o cheie unică și un URL prin care poate fi accesat. Dacă nu aş fi folosit Amazon S3, ar fi trebuit să stochez imaginile local pe server, ceea ce ar fi fost mult mai complicat și costisitor [8].
- e) **ngrok**: în cadrul aplicației, ngrok a fost folosit pentru a permite comunicarea dintre client (aplicația Android - care rulează pe emulator) și serverul Flask, care rulează local. Ngrok este utilizat pentru găzduirea server-ului local la o adresa web, creând un tunel securizat și generând un URL public pe care clientul îl folosește. Versiunea gratuita oferă o găzduire de maxima de 8 ore, după care server-ul este închis.
- f) **SQLite**: este o bază de date server-less folosită în special pentru aplicații de dimensiuni mici, unde nu este nevoie de o bază de date de capacitate foarte mare. SQLite este ușor de configurat și utilizat, nu necesită un proces de configurare ca pentru alte sisteme de baze de date. Baza de date este stocată într-un singur fișier, care poate fi copiat și mutat cu ușurință. Sqliite este compatibil cu o multitudine de limbaje de programare, printre care și Python, având o integrare bună

cu majoritatea framework-urilor, Flask inclusiv. Faptul că imaginile sunt stocate în Amazon S3 și am un număr redus de tabele în baza de date, SQLite a fost suficient pentru nevoile aplicației noastre.

- g) **Flask-JWT Extended**: este o extensie Flask care facilitează lucrul cu JSON Web Tokens. JWT-urile sunt folosite pentru autentificarea și autorizarea utilizatorilor în aplicație. Dacă nu aș fi folosit Flask-JWT ar fi trebuit să implementez funcționalitățile de la zero, crearea JWT-urilor, semnarea lor, verificarea validității JWT-urilor, extragerea identității utilizatorilor etc, ceea ce ar fi fost costisitor și ar fi putut introduce erori în implementare.

3.2.2 Design-ul bazei de date

În aplicația descrisă am structurat baza de date în trei tabele principale: "Users", "Images", "Collections". Aceste tabele stochează informațiile necesare pentru buna funcționare a serverului Flask și a aplicației în general.

- a) Tabela "Users": în această tabelă sunt stocate informațiile despre utilizatori. Când un utilizator intră pe pagina de "Register", completează datele necesare creării unui cont și apasă butonul de creare a contului, o cerere HTTP de tip POST este trimisă către server, iar după ce sunt efectuate câteva verificări, precum ca adresa de email să nu fie deja utilizată, serverul introduce în tabela "Users" o nouă înregistrare. Fiecare utilizator are un set de informații, acestea fiind reflectate prin coloanele din tabelă, astfel că avem: "id"-ul unui utilizator care este unic, o adresa de email ("email"), o parolă ("password") care este criptată înainte de a fi stocată în tabelă, un prenume ("first_name"), un nume de familie ("last_name"), genul ("gender"), un nume de utilizator ("username"), care nu trebuie neapărat să fie unic, o imagine de profil ("profile_picture"), în baza de date fiind stocat de fapt url-ul către Amazon S3, unde este salvată imaginea de profil, inițial această coloană are valoarea nul, deoarece utilizatorul nu își poate seta poza de profil în faza de creare a contului, și o descriere ("bio"), pe care utilizatorul o poate seta din pagina de setări a contului, la fel ca imaginea de profil. În această tabelă, "id" este cheia primară, ceea ce înseamnă că fiecare înregistrare este identificată în mod unic prin acest câmp.
- b) Tabela "Images": această tabelă stochează informațiile despre imaginile încărcate

de utilizatori. Fiecare imagine are un "id" care este cheie primară, id-ul utilizatorului căruia îi aparține imaginea ("user_id"), aceasta fiind o cheie straină, URL-ul către locația din Amazon S3 a imaginii ("s3_url"), dimensiunea fișierului ("file-size") și data la care imaginea a fost încărcată ("uploaded_at").

- c) Tabela "Collections": fiecare colecție are, de asemenea, un "id" unic care este cheie primară, un nume ("name") și id-ul utilizatorului care a creat colecția ("user_id") - cheia straină.
- d) Tabela de asociere "images_collections": se ocupă de relația "many-to-many" dintre imagini și colecții. Această tabelă conține doar două coloane: "image_id" și "collections_id", pentru id-ul imaginii, respectiv colecției. Ambele sunt chei străine. În Figura 3.1 se poate observa arhitectura bazei de date.

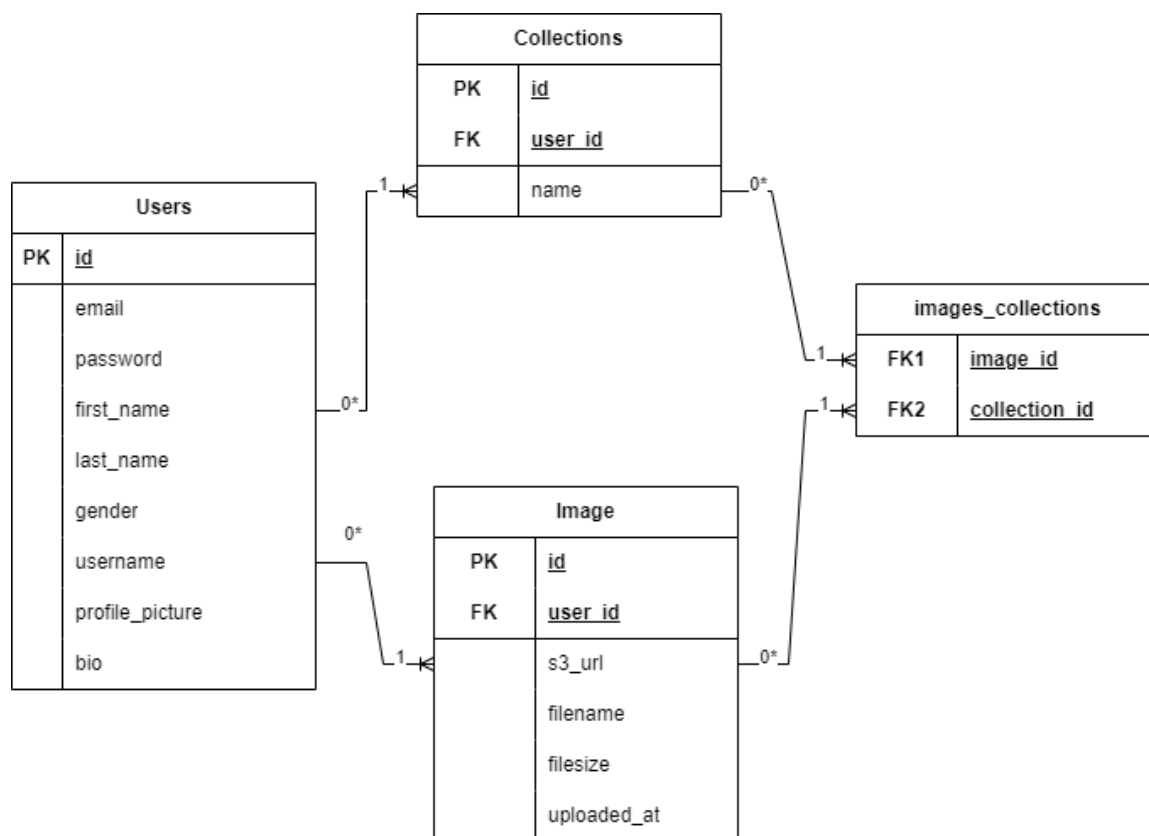


Figura 3.1: Diagrama bazei de date

În crearea tabelor am ținut cont de criteriile de normalizare a unei baze de date, astfel că, modelele de mai sus respectă următoarele: Prima formă normală (1NF) - fiecare coloană a tabelor conține o valoare atomică și fiecare înregistrare este identificată în mod unic prin intermediul unei chei primare. Spre exemplu, în tabela "Users" fiecare utilizator are un singur email, un singur nume de utilizator, o singură parola

etc, adică fiecare câmp este indivizibil; A doua formă normală (2NF) - aceasta cere ca toate elementele unei tabele să fie dependente funcțional de totalitatea cheii primare, dacă unul sau mai multe elemente sunt dependente funcțional numai de o parte a cheii primare, atunci ele trebuie să fie separate în tabele diferite. Deoarece fiecare tabel are o cheie primară care identifica în mod unic fiecare înregistrare, putem afirma că modelele noastre respectă 2NF; A treia formă normală ((3NF): modelele respectă această formă deoarece toate coloanele non-cheie sunt dependente de cheia primară în fiecare tabel [5].

Modelele demonstrează, pe lângă aceste principii, utilizarea eficientă a relațiilor dintre tabele. De exemplu, relația „many-to-many” dintre „Images” și „Collections” este gestionată folosind un tabel de asociere, care este o practică obișnuită în designul bazelor de date pentru a gestiona astfel de relații.

3.2.3 Descrierea implementării

Arhitectura aplicației server a fost modelată ținând cont de “design pattern”-ul Model-View-Controller (MVC). Acesta împarte aplicația în trei părți principale: Modelul - reprezintă datele și regulile de afaceri ale aplicației, “View”-ul -reprezentarea vizuală a datelor, în cazul nostru aplicația Android, care afișează datele către utilizatori, și “Controller”-ul - cel care procesează datele primite de la Model și le trimite înapoi la “view”.

```
14 usages
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(100), unique=True)
    password = db.Column(db.String(1000))
    first_name = db.Column(db.String(100))
    last_name = db.Column(db.String(100))
    gender = db.Column(db.String(30))
    username = db.Column(db.String(50))
    profile_picture = db.Column(db.String())
    bio = db.Column(db.String(1000))
    images = db.relationship('Image', backref='user', lazy=True)
    collections = db.relationship('Collection', backref='user', lazy=True)
```

Figura 3.2: Clasa “User”

În cazul nostru, modelele sunt reprezentate de clasele "User", "Image" și "Collection" deoarece acestea definesc structura datelor gestionate de aplicație. Controller-ul este reprezentat de rutele definite în modulul "routes", deoarece acestea preiau datele din cererile HTTP și returnează un răspuns către aplicația Android, view-ul. În figura 3.2 se poate observa structura unui model.

Structura serverului este una destul de simplă. Fiecare fișier din modulul "routes" conține rute care sunt legate de o anumită funcționalitate a aplicației. Spre exemplu, fișierul "login.py" conține rutele ce se ocupă de logarea și de înregistrarea utilizatorului, "gallery.py" conține rutele care gestionează galeria utilizatorului, iar "profile.py" se ocupă de gestionarea profilului utilizatorului. Acest modul poate fi numit "controller" deoarece "controlează" modul în care se răspunde la diferitele cereri HTTP.

Fișierul "_init_.py" reprezintă fișierul de configurare al aplicației Flask, aici este realizată conexiunea cu serviciul Amazon S3, se setează o cheie secretă pentru JWT, se inițializează SQLAlchemy etc. Fișierul „models.py” conține definițiile tuturor modelelor bazei de date, modul în care interacționează între ele și modul în care contribuie la structura generală a aplicației.

În proiectul de față am folosit biblioteca Flask-JWT-Extended pentru a implementa autentificarea și autorizarea utilizatorilor. În fișierul "login.py" avem un endpoint pentru autentificare. Când un utilizator trimite o cerere de tip POST către acest endpoint, se caută în baza de date un utilizator care are adresa de email egală cu cea primită în cerere. Dacă un utilizator cu acea adresa de email nu există, serverul trimite ca răspuns un mesaj corespunzător. Dacă există, serverul verifică dacă parola primită se potrivește cu parola stocată în baza de date, dacă acest lucru se întâmplă, atunci serverul generează un token de acces folosind biblioteca Flask-JWT-Extended și îl trimite ca răspuns clientului.

Autorizarea se referă la procesul prin care serverul determină ce acțiune poate efectua un utilizator. Flask-JWT-Extended ne permite protejarea anumitor endpointuri cu ajutorul decoratorului "@jwt_required", acesta permite doar utilizatorilor autentificați (care au trimis un token JWT valid în header-ul "Authorization" al cererii) să acceseze acel endpoint:

Prin folosirea decoratorului "@app.route()", care face parte din Flask, metoda devine un "controller". Atunci când serverul primește o cerere HTTP de tip POST la url-ul "/users/collections", acesta va apela funcția ce este definită sub decorator.

Modelele pe care le-am definit gestionează datele pe care le utilizăm pentru stoca-

```
@app.route('/users/collections', methods=['POST'])
@jwt_required()
def add_user_collection():
```

Figura 3.3: Exemplu adnotari - "@jwt_required" si "@app.route"

rea și manipularea informațiilor despre utilizatori, despre colecții de imagini și despre imaginile în sine. Fiecare model în parte este conceput pentru a reprezenta o anumită entitate a aplicației. Spre exemplu, modelul "User" deține informații despre utilizatori, precum numele, adresa de email, parola etc. Relațiile dintre modele sunt gestionate cu ajutorul funcționalităților ORM oferite de SQLAlchemy.

```
@app.route('/collections/<collection_id>/images', methods=['POST'])
@jwt_required()
def add_image_to_collection(collection_id):
    user_email = get_jwt_identity()
    user = User.query.filter_by(email=user_email).first()

    if user is None:
        return jsonify({'message': 'User not found'}), 404

    if 'image' not in request.files:
        return jsonify({'message': 'No image provided'}), 400
    image = request.files['image']
```

Figura 3.4: Exemplu endpoint, partea 1

La primirea unei cereri de la client, backend-ul manipulează cu ajutorul controller-ului datele și generează un răspund pe care îl trimite înapoi clientului. În cele ce urmează voi detalia implementarea unei rute pentru a exemplifica modul în care serverul gestionează o cerere. Atunci când un utilizator dorește să adauge o imagine într-o colecție, clientul trimite serverului o cerere HTTP de tip POST, care conține detaliile despre imagine și colecția în care se dorește adăugarea acesteia. Pe partea de backend, endpoint-ul corespunzător preia datele clientului, creează o instanță a imaginii folosind modelul "Image", și adaugă imaginea în colecție. Acesta "adăugare" este reprezentată ca o operațiune obiectuală deoarece SQLAlchemy abstractizează interogarea SQL (figura 3.5) [6].²

Din captura de ecran a codului ce conține endpointul de adăugare a unei imagini într-o colecție (Figura 3.4), putem observa implementarea unui set de verificări suplimentare. Acestea au rolul de a valida datele primite de la client, dar și de a verifica

autorizarea acestuia de a accesa ruta specificată.

```
if not valid_image_extension(image.filename):
    return jsonify({'message': 'Invalid image format'}), 400

if collection_id is not None:
    collection = Collection.query.get(collection_id)
    if collection is None:
        return jsonify(error="Collection not found"), 404
    else:
        collection = Collection(user_id=user.id, name="New Collection")
        db.session.add(collection)
        db.session.commit()

collection = Collection.query.get(collection_id)
if collection.user_id != user.id:
    return jsonify({'message':
        'You do not have permission to add images to this collection'}), 403

s3_url = upload_image_to_s3(image, user.id)

new_image = Image(user_id=user.id,
                  s3_url=s3_url,
                  filename=image.filename,
                  filesize=image.content_length)
db.session.add(new_image)
db.session.commit()

collection.images.append(new_image)
db.session.commit()
return jsonify({'image': new_image.serialize(),
                'message': "Image uploaded successfully!"}), 200
```

Figura 3.5: Exemplu endpoint, partea 2

În concluzie, implementarea server-ului se bazează pe principiile Model-View-Controller, ceea ce permite o împărțire eficientă a sarcinilor legate de logica de afaceri, prezentarea și manipularea datelor. Cu ajutorul "micro-framework"-ului Flask și SQLAlchemy am implementat "controller"-ele necesare manipulării datelor și a bazei de date. Am creat un sistem de autentificare și am asigurat autorizarea utilizatorilor prin utilizarea JWT-urilor folosind Flask-JWT-Extended.

Capitolul 4

Ghidul utilizatorului

Acest capitol al lucrării conține ghidul de utilizare al aplicației Android. Funcționalitățile principale ale acestei aplicații sunt: editarea imaginilor, creare de colaje, gestionarea imaginilor în colecții.

4.1 Navigarea în aplicație

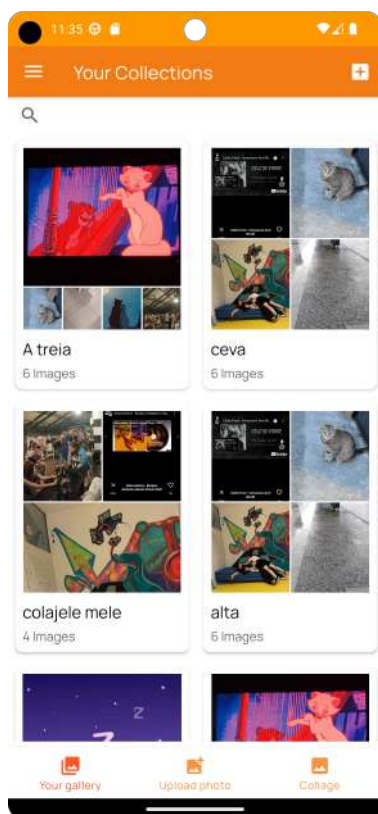


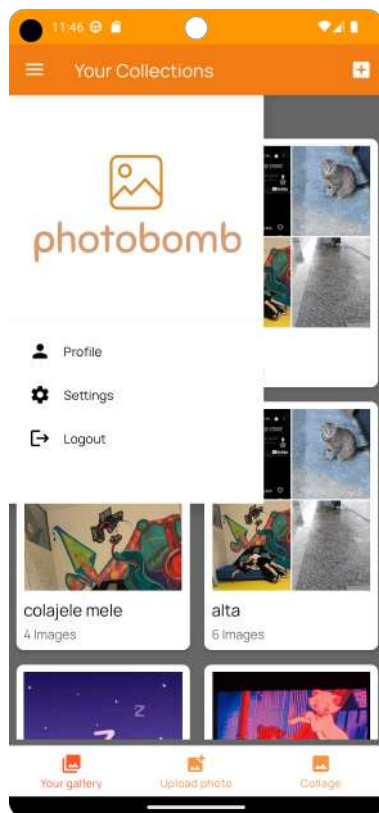
Figura 4.1: Pagina principală

După autentificarea cu succes în aplicație, utilizatorului îi este afișată pagina principală, ce conține lista colecțiilor sale. Fiecare colecție este reprezentată de ultima imagine încărcată și numărul de imagini pe care îl conține. În Figura 4.1 se poate observa meniul principal care afișează colecțiile utilizatorului.

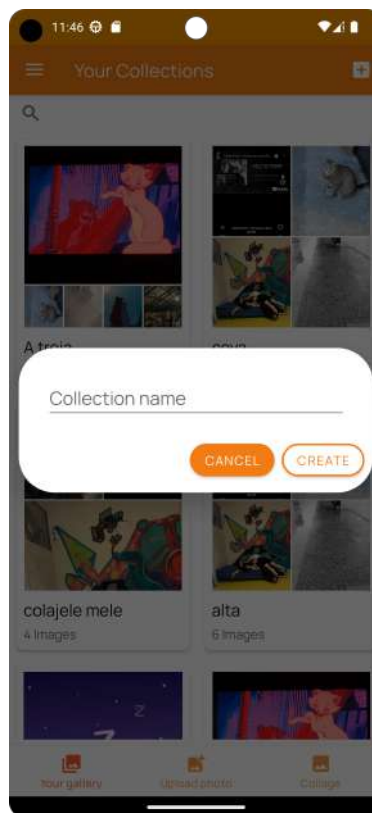
În partea de jos a acestui "screen" avem meniul principal al aplicației, ce conține trei butoane. Primul buton este cel pentru redirectionarea către colecțiile utilizatorului, al doilea, cere utilizatorului să aleagă în ce mod dorește să încarce o imagine în aplicație, făcând o captură foto sau alegând o imagine din galerie, iar cel de-al treilea buton trimite utilizatorul în activitatea de alegere a "template"-ului pentru colaj.

În partea superioară avem în stângă un buton

de tip "hamburger", care declanșează afișarea meniului glisant, funcționalitate prezentată în Figura 4.2 (a), iar în dreapta, butonul care adaugă o nouă colecție. Pentru crearea unei noi colecții, utilizatorul trebuie neapărat să introducă un nume și să apese butonul "Create", Figura 4.2 (b).



(a) Meniul glisant



(b) Adaugarea unei colecții

Figura 4.2: Funcționalitatea butoanelor superioare

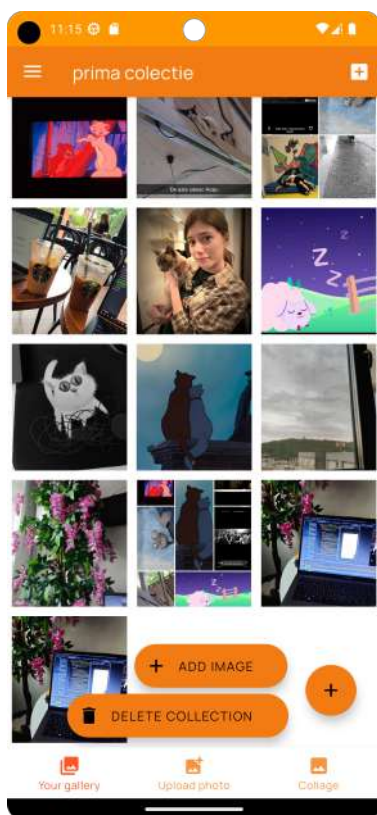
Meniul glisant, din Figura 4.2 (a), are următoarele butoane: "Profile", ce duce utilizatorul pe pagina profilului său, "Setting", ce permite utilizatorului să-și modifice datele personale și "Logout", cu ajutorul căruia utilizatorul se deconectează.

De asemenea, în partea superioară avem un "search bar" care permite utilizatorului să introducă cuvinte cheie pentru a găsi anumite colecții.

4.2 Vizualizarea colecțiilor de imagini

Pentru a vizualiza conținutul unei colecții, utilizatorul, care se află în pagina principală, trebuie să "atingă" o colecție pentru a o deschide. În urma acestei acțiuni, utilizatorul va fi redirectionat către un "screen" care conține toate imaginile din colecția respectivă, Figura 4.3 (a).

În partea de jos a acestui "screen" avem un buton care, odată apăsător, afișează alte funcționalități ale galeriei. Primul buton, "Add image", este pentru adăugarea unei noi imagini în colecție, iar al doilea, "Delete Collection", este pentru ștergerea colecției.



(a) Galeria unei colecții



(b) Imaginea "full screen"

Figura 4.3: Galeria de imagini

Prin apăsarea oricărei imagini, utilizatorul va fi redirecționat către pagina de "full screen" a imaginii, Figura 4.3 (b).

O altă funcționalitate a galeriei constă în faptul că imaginile pot fi selectate, acest lucru fiind exemplificat în Figura 4.3 (a). Odată ce utilizatorul a selectat prima imagine, acesta intră în "Selection Mode", astfel că, orice imagine "atinge" în continuare va intra în selecție. Având selecția dorită, utilizatorul poate șterge imaginile din selecție, sau le poate muta în altă colecție.

După ce utilizatorul atinge o imagine din galerie, un ecran precum cel din Figura 4.3 (b) îi va fi afișat. Cu ajutorul butoanelor din meniul de jos, utilizatorul poate efectua următoarele operații: să ștergă imaginea, butonul "Delete", să editeze imaginea, butonul "Edit", apăsarea acestui buton redirecționându-l în activitatea de editare a unei imagini, să mute imaginea în altă colecție, butonul "Move", apăsarea acestui buton îi va afișa lista colecțiilor sale și să salveze imagine în galeria "device"-ului, butonul "Save".

4.3 Editarea unei imagini

Prin apăsarea butonului "Edit image" utilizatorului i se va solicita să aleagă o modalitatea de a încarca o imagine, realizând o captura foto sau alegând o imagine din galerie.

După ce imaginea a fost aleasă cu succes, utilizatorul va fi redirecționat către activitatea de editare a imaginii, Figura 4.4. Meniul de jos conține toate uneltele de editare disponibile: filtrele ("Filters"), funcționalitatea de "decupare" a imaginii ("Crop"), reglarea luminozității ("Brightness"), reglarea saturației ("Saturation"), adăugarea "blur"-ului ("Blur") și adăugarea textului pe imagine ("Add text").

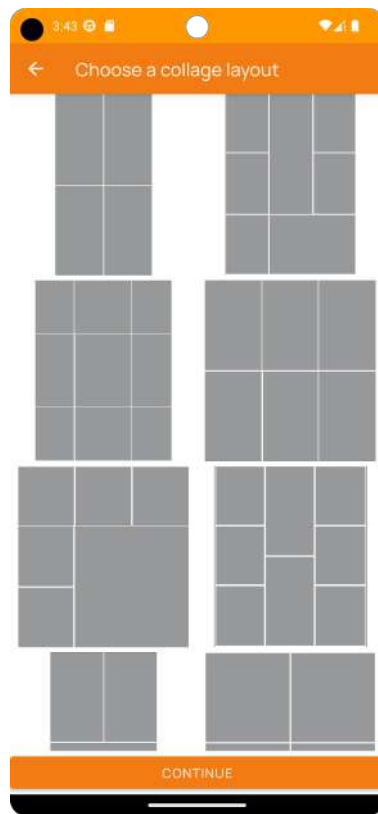
Meniul de sus conține două butoane: butonul de "Save", cu ajutorul căruia utilizatorul poate salva imaginea, ori în galerie personală ori într-o colecție din aplicație, și butonul "Undo", care are funcționalitatea de a elimina ultima modificare efectuată imaginii. Textele nu vor putea fi șterse prin apăsarea butonului de "undo", acestea trebuind să fie eliminate manual. O componenta de text adăugată pe imagine poate fi editată și stersă cu butonul de "Delete" din dialogul de editare, care poate fi afișat apăsând direct pe textul respectiv. Dialogul de editare are următoarele funcționalități: schimbarea textului, alegerea culorii textului, ștergerea completa a componentei.



Figura 4.4: Pagina de editare a imaginii

4.4 Creare unui colaj

Atunci când utilizatorul apasă cel de-al treilea buton al meniului principal, "Collage", acesta este redirecționat către pagina de alegere a unui "layout" pentru colaj, Figura 4.5 (a). După ce utilizatorul face o alegere și apasă pe buton "Continue", va fi redirecționat către pagina de creare a colajului.



(a) Galeria unei colectii



(b) Imaginea "full screen"

Figura 4.5: Creare unui colaj

Prin apăsarea fiecarul "cadran" gri din "layout", utilizatorului i se va deschide galeria personală de unde trebuie să aleagă o imagine. După alegere, acesta este redirectionat înapoi în pagina de editare a colajului, unde poate continua procesul de creare. Dacă un "cadran" al colajului este apăsat lung ("long pressed"), acesta va fi selectat, iar un buton de edit va apărea în colțul drept al colajului, Figura 4.5 (b). Prin apăsarea acestui buton, utilizatorul poate edita imaginea respectivă. Dacă în acel "cadran" nu este nicio imagine, buton de edit nu va funcționa și un mesaj corespunzător va fi afișat.

Butonul din dreapta sus declanșează funcționalitatea de salvare a colajului, utilizatorul poate alege să salveze imaginea rezultat într-o colecție sau în galerie.

Concluzii

Lucrarea de față s-a concentrat pe dezvoltarea unei aplicații mobile, numită "PhotoBomb", care furnizează utilizatorilor o platformă completă de editare și gestionare a fotografiilor. În cadrul acestei lucrări, am plecat de la o motivație personală puternică: ideea de a avea o soluție simplă care integrează pe de o parte editarea imaginilor și crearea de colaje, iar pe de altă parte gestionarea fotografiilor într-un mod plăcut și creativ, dar și dorința de a facilita procesul de editare a imaginilor și crearea de colaje pentru utilizatorii de toate nivelurile de experiență.

În această aplicație am adus laolaltă diverse tehnologii pentru a realiza o aplicație capabilă să satisfacă o gamă de cerințe de editare a imaginilor. Prin combinarea funcționalităților de bază, cum ar fi ajustarea luminii și culorilor, cu funcționalități mai avansate precum adăugarea de filtre și crearea de colaje, aplicația oferă utilizatorilor posibilitatea de a personaliza complet imaginile lor. În plus, opțiunea de salvare a imaginilor în colecții oferă un instrument util pentru organizarea și revizuirea fotografiilor.

Consider că aplicația poate fi îmbunătățită atât pe partea vizuală cât și pe partea funcționalităților. Viitoare îmbunătățiri ar putea include adăugarea de noi instrumente de editare și filtre, îmbunătățirea interfeței cu utilizatorul și integrarea cu alte platforme și aplicații de partajare a imaginilor.

În concluzie, această lucrare a reușit să atingă obiectivele initiale pe care ni le-am propus, realizarea unei aplicații care să facă editarea imaginilor un proces simplu și intuitiv pentru toți utilizatorii.

Bibliografie

- [1] Maria Petrou, Costas Petrou, *Image Processing: The Fundamentals*, Wiley, 2010.
- [2] Sandipan Dey, *Hands-On Image Processing with Python*, Packt, 2018.
- [3] Bryan Sills, Brian Gardner, Kristin Marsicano, Chris Stewart, *Android Programming: The Big Nerd Ranch Guide*, Addison-Wesley Professional, 5th edition, 2022
- [4] Miguel Grinberg, *Flask Web Development: Developing Web Applications with Python*, O'Reilly Media, 2nd edition, 2018
- [5] Michael J. Hernandez, *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design*, Addison-Wesley Professional, 2nd edition, 2003
- [6] Myers Jason, *Essential Sqlalchemy: Mapping Python to Databases*, O'Reilly Media, 2015
- [7] John Horton, *Android Programming for Beginners*, Ingram short title, 2nd edition, 2018
- [8] AWS S3 <https://www.javatpoint.com/aws-s3>
- [9] What is Client-Server Architecture? Everything You Should Know: <https://www.simplilearn.com/what-is-client-server-architecture-article>