Aplicatie crossplatform de editare foto

Dunica David-Gabriel

May 27,2022

Cuprins

1 Introducere **2**

2 Suport tehnic **2**

3 Prezentare tehnica **3**
3.1 Flutter SDK 3
3.1.1 Prezentare Framework 3
3.1.2 Exemplificare implementare UI 4
3.1.3 Biblioteci utilizate 5
3.2 Firebase backend 10
3.2.1 Authentication 11
3.2.2 Cloud Firestore 11
3.2.3 Firebase Storage 13

4 Mod de utilizare **13**

5 Concluzii **15**
5.1 Obiective 15
5.2 Utilitate 16
5.3 Performante 16

6 Referinte bibliografice **17**

1 Introducere

Tema abordata de mine in cadrul acestui proiect este o aplicatie de editare foto ce suporta si interactiuni intre utilizatori prin intermediul bazei de date atasate proiectului. Aplicatia cuprinde autentificare (pentru a defini clar profilul fiecarui utilizator), editorul foto, posibilitatea de a incarca fotografiile in baza de date, astfel postarile sunt accesibile tuturor utilizatorilor si un sistem de cautare prin care utilizatorii pot cauta si interactiona cu alti useri. Totodata utilizatorii pot sa aprecieze orice postare fiind introdus un sistem de aprecieri (like).

Considerand obiectivul initial: realizarea unei aplicatii crossplatform cu autentificare de editare foto si incarcare in baza de date, consider ca l-am indeplinit, dar chiar mai mult am implementat si sistemul de interactiune intre utilizatori, format din cautare + vizitarea profilului altor utilizatori + sistemul de aprecieri.

2 Suport tehnic

Avand in vedere ca am realizat o aplicatie mobile aceasta cuprinde mai multe nivele de interactiune: astfel avem interfata grafica (propriu zis interactiunea userului cu aplicatia), apoi avem partea de backend care cuprinde urmatoarele aspecte: stocarea interna a datelor necesare in aplicatie, stocarea externa centralizata intr-o baza de date si nu in ultimul rand API-urile ce faciliteaza conexiunea dintre baza de date si backendul local al aplicatiei.

Pentru acest proiect am optat sa folosesc IDE-ul celor de la Google: Android Studio pe care am instalat pluginul de Flutter (aceasta faciliteaza developmentul crossplatform Android - IOS). Partea de backend local este realizata cu ajutorul principiului de Redux (acesta confera o viziunea globala a tuturor variabilelor stocate local in momentul utilizarii aplicatiei), Flutter implementeaza aceasta biblioteca, astfel utilizarea ei este prielnica. Biblioteca Redux faciliteaza cu succes gestionarea si centralizarea de stari ale unei aplicatii, astfel cel mai mare avantaj al acesteia fiind usurinta cu care se poate modifica un proiect structurat cu redux. Partea de backend ce reprezinta accesul la baza de date, stocarea si gestiunea datelor este reusita cu ajutorul Google Firebase; acesta este un serviciu de cloud de la Google care pentru costuri minimale (in cazul meu nule) confera servicii de gestiune si stocare a datelor asemanatoare unei baze de date. Printre serviciie utilizate de mine

in acest proiect se regasesc: Authentification (pentru realizarea loginului), Cloud Firebase (o baza de date non-SQL utilizata pentru stocarea datelor utilizatorilor) si Storage (utilizata pentru a stoca structurat pozele utilizatorilor).

3 Prezentare tehnica

3.1 Flutter SDK

3.1.1 Prezentare Framework

Flutter este un SDK pentru crearea de user-interface open-source creat de Google. Asa cum reiese din sursa [1], este utilizat pentru crearea de aplicatii crossplatform printre care amintim: iOS, Android, Linux, macOS, Windows, web, etc. Engineul Flutter este scris in C++ si ofera suport de nivel inalt de randare . In cadrul proiectului meu am utilizat Flutter pentru implementarea crossplatform a unei aplicatii de mobile cu suport Android si iOS.

Pentru a crea un proiect Flutter trebuie scris in consola urmatoarea comanda:

```
flutter create [project]
```

Pentru a crea APK-ul pentru android se utilizeaza urmatoarea comanda[2]:

```
flutter build apk --plit-per-abi
```

Pentru a crea proiectul iOS pentru lansare se executa urmatoarea comanda[3]:

```
flutter build ipa --release --export-options-plist=$HOME/export_options.plist
```

Limbajul de implementare al acestui framwork este Dart (extensia .dart), fiind foarte asemanator cu Java. In Flutter toate widgeturile ce compun ecranul sunt de fapt niste clase, care sunt interpretate de catre engineul facut in C++, care mai departe il compileaza ca randare. Un proiect in Flutter poate fi constituit ca un arbore de widgeturi.

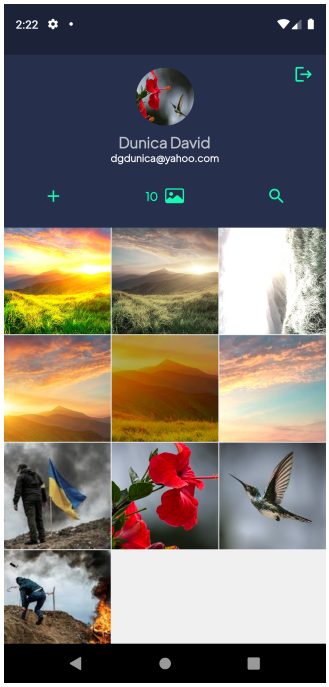
Pentru a lucra intr-un proiect flutter se modifica folderul **lib**; acesta contine si fisierul main.dart care se se ruleaza la build. In acest fisier intalnim metoda void main() in care se apeleaza metoda runApp() care are ca

parametru un Widget; aceasta metoda faciliteaza incarcarea acelui widget pe ecran. Astfel widgetul/ clasa **MyApp** se randeaza si devine automat baza arborelui de widgeturi. Clasa MyApp are implementata o metoda ce returneaza tot un widget, build(), pentru a crea o aplicatie trebuie sa returnam MaterialApp() aceastei clase ii putem da initial valoare prin constructor pentru mai multe atribute esentiale in aplicatie, precum numele aplicatiei, theme (tema de fonturi si culori folosite) si routes care tine evidenta tuturor ecranelor ce compun aplicatia.

3.1.2 Exemplificare implementare UI

Asa cum am precizat mai sus pentru a compune un user interface ne trebuie mai multe widgeturi. La baza fiecarui screen sta un **Scaffold** widget; acesta implementeaza material designul intr-un model visual structurat. In compunerea acestei aplicatii am folosit diverse widgeturi precum: Container, Text, Image, Icon, GridView, Row, Column, Stack, etc. Pentru a intelege mai bine modalitatea de creare a unui screen am considerat ca exemplu HomeScreen-ul, cel care reprezinta profilul utilizatorului.

In acest exeplu am considerat ca cel mai adanc nivel din aceasta ramura a aplicatiei sa fie un **Stack**; acest widget primeste o lista de widgeturi ca parametru pe care le aseaza pe ecran una peste alta, asemenea unei stive. Astfel avem un **IconButton**; cel din stanga sus si un **Column**. Coloana are de asemenea o lista de doua widgeturi: in primul rand avem un **Container** inalt cat 30% din ecran si lat cat ecranul deviceului, de culoare albastru inchis, iar apoi avem alt **Container** inalt cat 70% din ecran; acest container are ca child un widget de tip **GridView.builder** care efectiv genereaza imaginile asemenea unei liste; este de precizat ca inaltimea, aspect-ratio-ul si numarul de poze de pe linie sunt selectate de mine ca parametrii ai widgetului. In cazul in care lista de poze a userului este goala **Containerul** de jos isi schimba widgetul copil din **GridView** intr-un widget de tip **Text**.



Revenind la partea superioara a screenului **Containerul** albastru are ca child un widget **Col-**

umn, care pune in coloana urmatoarele widgeturi: Containerul rotund, care are la randul sau un copil **Image** (acesta afiseaza imaginea de profil a userului), apoi are doua widgeturi **Text** care afiseaza numele apoi adresa de email a userului, iar in final este un **Row**, ce pune in linie 3 **Containere** care au ca child un widget **Icon** si un **Text** care reda numarul de postari. Alinierea widgeturilor intre ele se face prin introducerea unor parametrii de tip padding sau margin asemenea cu cele din limbajul HTML, sau chiar prin adaugarea de **Containere** goale si transparente care sa ocupe locul.

O ultima precizare la partea de constructie a UI-ului. Pentru interactiunea utilizatorului cu ecranul am folosit widgetul **GestureDetector**. Acesta actioneaza o functie atunci cand pe suprafata Widgetului parinte se petrece vreun gest, de exemplul in cazul meu **onTap**; astfel cand apasam pe iconita + se executa o metoda care schimba ecranul de pe acesta pe cel de incarcare a unei noi imagini.

3.1.3 Biblioteci utilizate

In cadrul acestui proiect am uitlizat un numar de biblioteci open source, care sa ma ajute in indeplinirea cerintelor dorite. In Flutter bibliotecile se adauga in fisierul **pubspec.yaml** sub liniile de cod **dependencies** sau **dev_dependencies** dupa modelul urmator:

```
[package_name]: [version ex: ^1.0.0]
```

3.1.3.1 Biblioteci pentru editare foto + rationament

Am utilizat 3 biblioteci pentru partea de editare a imaginii; siteul de documentatie al acestor biblioteci este [4]:

```
image_picker: ^0.8.4+11
```

Acesta biblioteca faciliteaza incarcarea in aplicatie a fotografiilor din galerie prin metoda **ImagePicker().getImage(source: [sursa de unde se ia poza ex: ImageSource.gallery])**. Imaginile incarcare din galerie sunt de tip **PickedFile**, iar pentru a o putea manipula avem nevoie sa o facem de tip **File**, lucru simplu avand in vedere ca putem sa facem o variabila de tip **File(pickedFile.path)**.

```
extended_image: ^6.0.2+1
image_editor: ^1.0.2
```

Aceste doua biblioteci au fost necesare pentru a efectua anumite elemente de editare precum crop, flip si rotate care sunt deja implementate in biblioteca **extended_image**. Pentru rotire avem implementata metoda **editorKey.currentState?.rotate(right: [BOOL VALUE])**, iar pentru flip avem **editorKey.currentState?.flip()**.

Pe langa aceste functionalitati am implementat si 3 filtre ce ajusteaza nivelul **RGB** al imaginii; am realizat acest lucru cu ajutorul bibiliotecii standard din Flutter **dart:ui**. Astfel am folosit matricea **ColorFilter.matrix** din cadrul widgetului **ColorFiltered**, ce aplica matricea ca filtru imaginii selectate de utilizator. Ecuatiile utilizate pentru aceasta ajustare au fost preluate din [5].

In cod fiind vorba despre o lista de valori double **defaultColorMatrix**. Matricea de valori am reprezentat-o in felul urmator:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Aceasta matrice reprezentand primele patru linii din matricea A de aflare a vectorului de valori:

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

Pentru ajustarea nivelului de saturatie am considerat o variabila care default are valoarea 1, iar userul poate sa o modifice intre **[0,2]**. Aceasta variabila numita **saturation** se scade din 1 si obtinem **invSat**. Aceasta noua variabila este utila pentru aflarea noilor valori red, green si blue astfel:

$$R = 0.213 * invSat;$$

$$G = 0.715 * invSat;$$

$$B = 0.072 * invSat;$$

Dupa aflarea acestor valori se completeaza matricea de filtru pe primele 3 pozitii ale primelor 3 colane(acestea avand valorile relevante RBG), iar pe diagonala principala valorilor li se adauga nivelul saturatiei pentru a incrementa accentul de noanta pe pixeli.

$$\begin{bmatrix} R + saturation & G & B \\ R & G + saturation & B \\ R & G & B + saturation \end{bmatrix}$$

Pentru filtrul de contrast am considerat o variavila **contrast** care vine initial tot cu valoarea 1, iar userul poate sa o modifice in intervalul [**0, 4**]. In ceea ce priveste editarea acestui filtru lucrurile sunt ceva mai simple; trebuie modificate numai valorile de pe diagonala principala cu valoarea contrastului.

$$\begin{bmatrix} contrast & 0 & 0 \\ 0 & contrast & 0 \\ 0 & 0 & contrast \end{bmatrix}$$

Pentru cel de-al treilea filtru, si anume brightness, am considerat o variabila **bright**. Aici nu a mai fost cazul sa aduc nicio modificare in matricea, deoarece in cadrul widgetului **ExtendedImage** pot modifica parametrul color, care aplica un astfel de filtru. Rangeul de bright este [**-1,1**], iar pentru o valoare sub 0 parametrul color ia valoarea **Colors.black.withOpacity(-bright)** sau in caz contrar **Colors.white.withOpacity(bright)**.

3.1.3.2 Biblioteci pentru elemente de UI

Pentru realizarea interfetei grafice am utilizat 2 biblioteci, printre care una este standart:

```
dart:ui
```

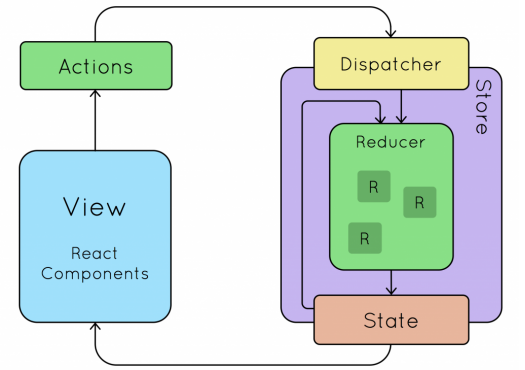
Aceasta este amintita si mai sus la sectiunea anterioare; aceasta biblioteca este standard in Flutter si se ocupa cu gestiunea widgeturilor.

```
cupertino_icons: ^1.0.4
```

Aceasta biblioteca este utilizat pentru widgetul **CupertinoIcons**, care include o colectie de Icons utilizate pe iOS.

3.1.3.3 Biblioteci pentru structurarea aplicatie + Redux

Redux reprezinta o alternativa fiabila pentru managerierea si centralizarea starilor prin care trece o aplicatie, asa cum este prezentat in [6]. Acest tool functioneaza dupa urmatorul principiu: exista un **Store** global care incapsulea variabilele utilizare in reprezentarea UI-ului. UI-ul prin interactiunea userului creeaza un **Action**. O actiune la randul ei creeaza un **Dispatcher** in Store. Aici intervine **Epics**; acest epics leaga actiunea de rezultatul ei care poate sa fie de success sau error (in cazul de utilizare a apiurilor, daca nu se pot prelua datele necesare prin api). Daca actiunea este successful atunci se trece in starea de **Reducer**; in Reducer se pot face modificari la elementele din Store, astfel se creeaza aceasta incapsulare de date. Dupa ce se modifica in **Reducer** se modifica statul si se trece iar in starea de UI unde se modifica afisajul. Se poate intelege acest concept si prin reprezentarea urmatoare:



[fig 1.0] Model automat redux furnizata de la [7]

Pentru utilizarea tool-ului Redux am avut nevoie de urmatoarele pachete:

```
redux: ^5.0.0

flutter_redux: ^0.9.0

redux_epics: ^0.15.1
```

Totodata cu aceste pachete am mai utilizat 6 pachete care sa ma ajute in generarea de cod pentru clasele si actiunile componente automatului Redux. Prin executarea comenzii urmatoare in consola aceste pachete, prin tool-urile lor, identifica fisierele din folderele /models (pentru clase) si /actions (pentru actiuni) si genereaza in **index.g.dart**, **serializers.g.dart** si **index.freezed.dart** clasele cu tot cu serializare si deserializare.

```
flutter pub run build_runner watch --delete-conflicting-outputs
```

Pachetele pentru generare clase sunt:

```
built_value: ^8.1.4

built_collection: ^5.1.1

build_runner: ^2.1.8

built_value_generator: ^8.1.4
```

Iar cele pentru generare actiuni sunt:

```
freezed: ^1.1.1

freezed_annotation: ^1.1.0
```

3.1.3.4 Biblioteci pentru backend si Firebase

Pentru utilizarea backendului Firebase si a serviciilor furnizare de acestia am avut nevoie de importat 3 pachete:

```
firebase_auth: ^3.3.12
```

Firebase auth se ocupa cu partea de inregistrare, acesta include metode precum: **_auth.createUserWithEmailAndPassword(email: email, password: password)** sau **_auth.signInWithCredential(credential)**.

```
cloud_firestore: ^3.1.11
```

Firebase cloud leaga aplicatia de baza de date in cloud. Astfel se pot utiliza metode precum:

```
 await _firestore
     .doc('users/${user.uid}')
     .set(user.json);
```

Acest exemplu seteaza in baza de date un nou user la register.

```
firebase_storage: ^10.2.10
```

Firebase storage este utilizat pentru serviciul de storage de la Firebase. Am utilizat acest pachet, deoarece pozele utilizatorilor sunt stocate in aceasta baza de date. Un exemplu de stocare a unei poze:

```
await _storage
        .ref('users/$uid/$index')
        .putFile(File(path));
```

Un ultim pachet utilizat este http; acesta permite interactionarea cu Stringuri ca linkuri. L-am utilizat deoarece pozele nu se descarca local, ci sunt vazute via network.

```
  http: ^0.13.4
```

3.2 Firebase backend

Asa cum am mentionat mai sus partea de backend a aplicatiei este realizata in Firebase. Acest serviciu este furnizat de Google si reprezinta o alternativa fiabila pentru aplicatii cu trafic de date redus. Am folosit trei servicii Firebase: Authentication, Cloud Firestore si Storage.

Pentru a conecta aplicatia la baza de date Firebase se completeaza pe siteul Firebase Android package name-ul aplicatiei (acesta se gaseste in folderul *android/build.gradle*). Dupa acest pas se introduce in folderul *app* fisierul *google-services.json*, descarcat de pe siteul Firebase, iar in final tot la pathul *android/build.gradle* se adauga urmatoarele 4 linii:

```
apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'

dependencies  {
  implementation platform('com.google.firebase:firebase-bom:29.3.1')
  implementation 'com.google.firebase:firebase-analytics'
}
```

Acum ca avem aplicatia legata la baza de date trebuie sa initializam instante catre cele 3 servicii, prin liniile urmatoare de cod:

```
  await Firebase.initializeApp();

  final FirebaseAuth auth = FirebaseAuth.instance;
  final FirebaseFirestore firestore = FirebaseFirestore.instance;
  final FirebaseStorage storage = FirebaseStorage.instance;
```

3.2.1 Authentication

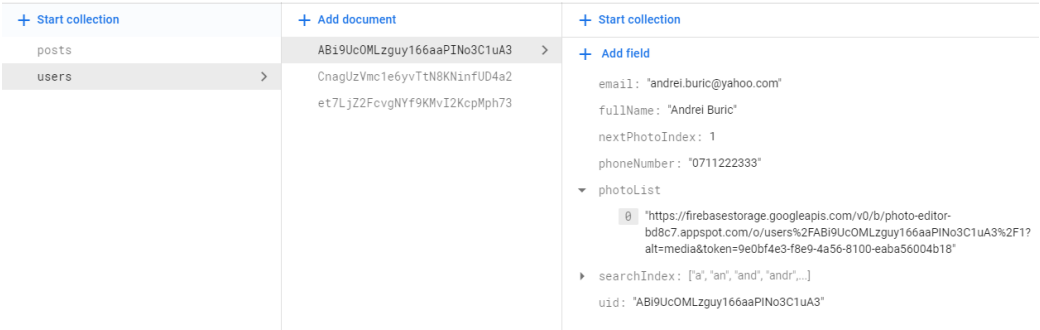
Acest serviciu se ocupa cu autentificarea pe aplicatie, astfel am utilizat acest serviciu pentru login, register si verificarea daca userul este logat sa intre direct in aplicatie fara sa isi mai introduca credentialele[8].

Pentru a te inregistra se foloseste functia **signInWithCredential** care va introduce in baza de date noul user. Acest sistem de autentificare introduce si prezenta unui token care se creeaza automat de fiecare data cand userul se autentifica pe aplicatie. Acest token se retine atat local cat si in baza de date, fiind util atunci cand utilizatorul intra pe aplicatie, astfel acesta nu este nevoit sa isi introduca credentialele de fiecare data cand intra in aplicatie. Pentru a verifica acest token se executa functia **currentUser** atunci cand aplicatia se porneste. In cazul in care exista token activ aplicatia porneste de pe **MainPage**, iar in caz contrar se porneste de pe **LoginPage**.

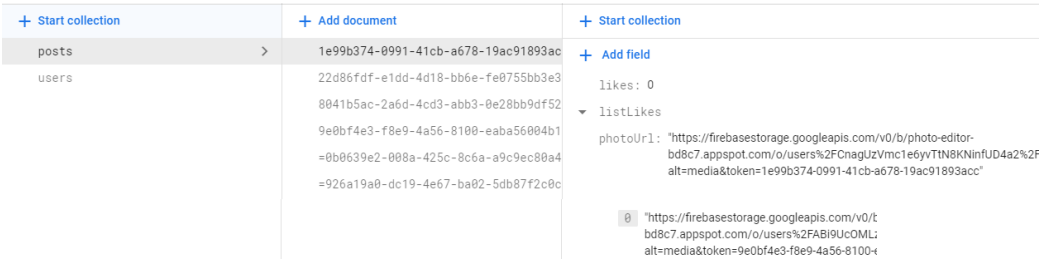
Pentru a se autentifica cu un cont deja existent userul trebuie sa isi introduca adresa de email si parola. Aceste doua credentiale se introduc in functia **LoginInWithCredential** ce returneaza un User in cazul in care datele sunt corecte. Authentification cuprinde si alte servicii gratuite precum verificarea emailului si schimabarea parolei. Aceste doua functionalitati planuiesc sa le introduc in etapa urmatoare a aplicatiei, pentru a imbunatati experienta utilizatorilor si a preintampina diferite erori ulterioare.

3.2.2 Cloud Firestore

In Cloud Firestore stochez propriu zis datele utilizatorilor si postarile[9].Baza de date este structurata in felul urmator: in colectia *users* intalnesc mai multe documente cu numele id-ul userului, iar fiecare document are inregistrat informatiile userilor. In colectia *posts* am mai multe documente cu tokenul fiecarei postari (tokenul fiind ultimele 26 de caractere din linkul catre imaginea stocata), iar in fiecare document se intalnesc informatiile unei postari. Se poate observa ca o baza de date Cloud Firestore este nonSQL, astfel aceasta baza de date este strucurata asemenea unor fisiere de date.



[fig 2.0] Structura users



[fig 2.1] Structura posts

Interactiunea cu baza de date se face prin functiile clasice si pentru SQL, si anume: **get, set, update, delete**. In interactiunea cu aceasta baza de date am folosit toate cele patru variante. Spre exemplu pentru set am utilizat aceasta functie atunci cand am vrut sa introduc un nou user in baza de date. Am folosti get spre exemplu atunci cand am vrut sa scot datele unui user cautat din baza de date. Este de mentionat ca atunci cand scoti din baza de date se creeaza un fisier json care se descarca automat, iar in functie de atributele clasei pe care vreau sa o completez se selecteaza din acel json doar elementele mapei care corespund cu numele si tipul de date furnizat in clasa.

Pentru functia de update am considerat incrementarea numarului de likeuri care creste sau scade atunci cand un user interactioneaza cu o postare. In cele din urma am utilizat functia de delete atunci cand userul doreste sa isi stearga poza de pe profil, astfel se sterge din colectia userului linkul spre Cloud Storage.

Utilizand acest serviciu de Cloud Firestore zilnic pot inregistra 20.000 scrieri/zi , 50.000 citiri/zi si 20.000 stergeri/zi, ceea ce pentru moment reprezinta o bariera de peste 5000 de useri activi/zi care sa aiba un consum de date moderat. Totusi preturile utilizarii unei astfel baze de date sunt reduse

deoarece 100.000 citiri/zi in plus costa 0.06 dolari, iar un gigabyte de memorie stocata pe luna reprezinta un cost de 0.18 dolari, serviciul avand din start un gigabyte de memorie lunara gratis. Avand in vedere ca in acest model de baza de date se stocheaza numai variabile, un gigabyte reprezinta un volum considerant de date

$$1GB = 2^{30} biti$$

Astfel aceasta varianta de backend este, dupa parerea mea, convenabila pentru un proiect care nu are resurse financiare pentru o echipa numeroasa.

3.2.3 Firebase Storage

Firebase storage este asemanator cu Cloud Firestore, insa scopul utilizarii acestui serviciu este de a stoca pozele utilizatorilor. Astfel exista un director *users* in care exista mai multe directoare cu numele id-ul userilor care stocheaza toate pozele acestora.

Exemplu de interactiune cu storage este de a folosi functia **putFile** care introduce la adresa **ref** un fisier de tip File. De indata ce se introduce in baza de date se creeaza automat un url pentru a descarca poza in baza de date; practic acest url este contacul apicatiei cu pozele stocate in aceasta baza de date. Pentru a furniza acest url se acceseaza functia **getDownloadURL**. Pentru a sterge pozele din Firebase Storage se utilizeaza functia delete. Este de mentionat ca atunci cand utilizatorul sterge o poza pe de-o parte se sterge Url-ul din Cloud Firestore si pe de alta parte se sterge poza propriu zis din Firebase Storage.

De asemenea ca la Cloud Firestore acest serviciu este contra cost, insa preturile sunt in1deajuns de mici incat sa fie nesimnificative pentru o alicatie care scaleaza. Acest serviciu iti furnizeaza 5 GB gratuiti lunar de stocare, iar pentru fiecare GB in plus se plateste 0.026 dolari. Sunt 50.000 descarcari/zi gratuite si 0.004 dolari/10k, iar pentru incarcari sunt 20.000 gratuite pe zi si 0.05 dolari/10k.

4 Mod de utilizare

Aplicatia porneste initial pe pagina de **login**; acolo userul are doua optiuni: ori isi introduce credentialele (email si parola), ori daca este nou pe aplicatie apasa pe TextButton-ul pe care scrie "Register". In cazul in care apasa pe "Register" se va deschide o noua pagina, si anume pagina de **Register**,

unde userul trebuie sa isi completeze datele si sa apese pe SignIn. Dupa ce userul apasa Login sau Signin, daca conexiunea la internet este valida si credentialele sunt corecte aplicatia deschide pagina **Main**, care teoretic este pagina principala a userului unde poate adauga poze si sa isi vada profilul. Pe acesta pagina **Main** userul poate sa se deconecteze, apasand pe butonul dreapta sus, poate sa adauge o postare, daca apasa pe Iconul cu plus, poate sa caute alti useri daca apasa pe Iconul cu lupa sau poate sa isi vada pozele apasand pe ele. Userul isi poate schimba imaginea de profil apasand pe cercul cu iconita de user de pe pagina sa principala.

Daca userul apasa pe iconita cu plus este redirectionat in galerie pentru a selecta o poza. Dupa ce a ales o poza ecranul se muta pe screenul **Edit Image**. Acolo userul poate sa editeze poza dupa modurile explicate la sectiunea 3.1.3.1. In partea de sus sunt doua iconite in partea dreapta. Prima este pentru a reseta filtrele, iar a doua pentru a posta propriu zis imaginea.

Daca userul doreste sa caute alti useri apasa pe iconita cu lupa. Acesta redirectioneaza userul spre screenul de **search**. Pe acesta pagina userul introduce in partea de sus usernameul, iar aplicatia da trigger la fiecare 0.5 secunde la actiunea de cautare in baza de date a userilor cu acea combinatie in nume. Dupa ce se gaseste in lista userul dorit se poate apasa pe el pentru a fi redirectionat catre pagina acelui user.

Userul poate da click pe orice poza. Daca se apasa pozele se deschide **Post** screenul, unde userul poate sa aprecieze postarea si in cazul in care este postarea sa, sa o stearga.

Este de mentionat ca butoanele care pornesc actiuni cu impact mai mare precum SignOut sau DeleteImage pornesc un **AlertDialog** in urma caruia userul trebuie sa confirme dorinta acelei actiuni.

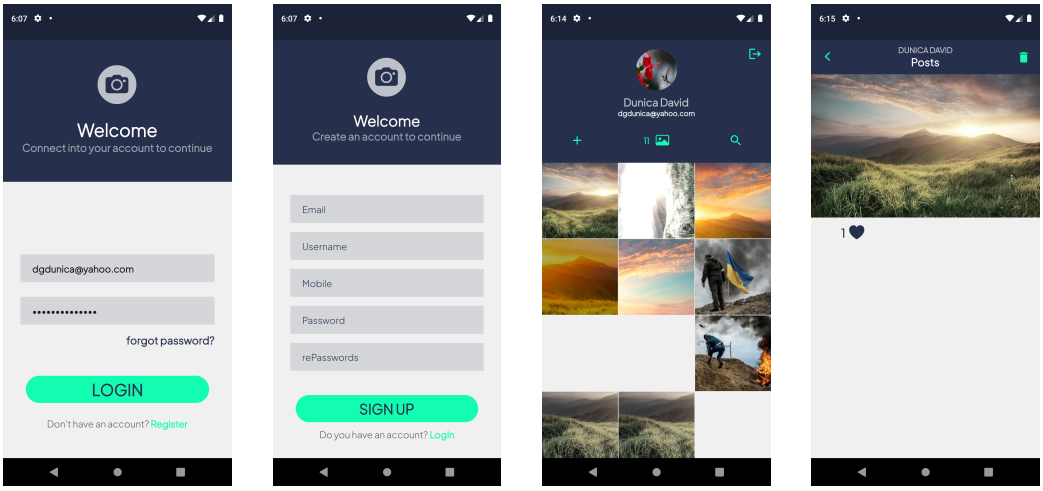


Figure 1: Login Page & Register Page & Main Page & Post Page

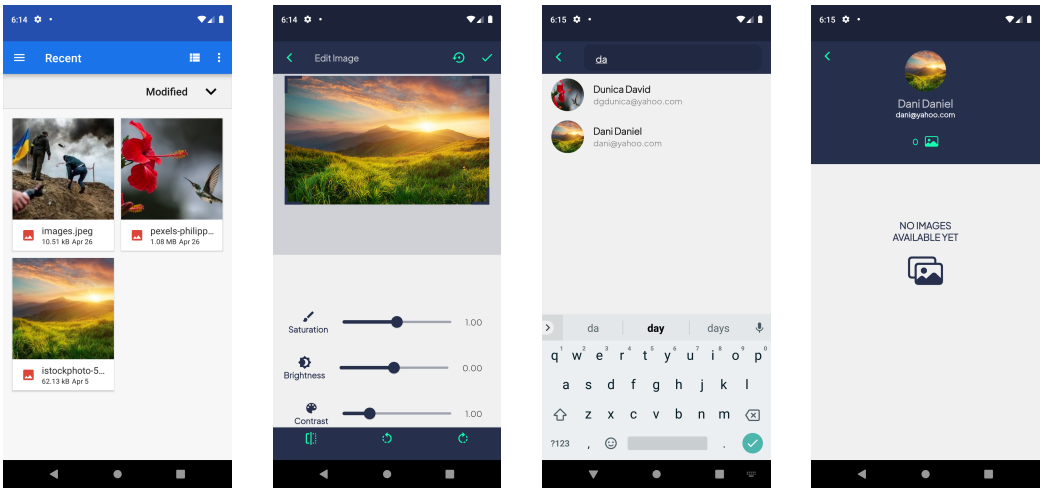


Figure 2: Galery Page & Edit Image Page & Search Page & Result Page

5 Concluzii

5.1 Obiective

Din punct de vedere al obiectivelor aplicatiei, personal ma declar multumit. Am reusit sa indeplinesc toate obiectivele pe care mi le-am pus initial. Mai mult de atat am reusit sa mai implementez in timpul ramas si alte func-

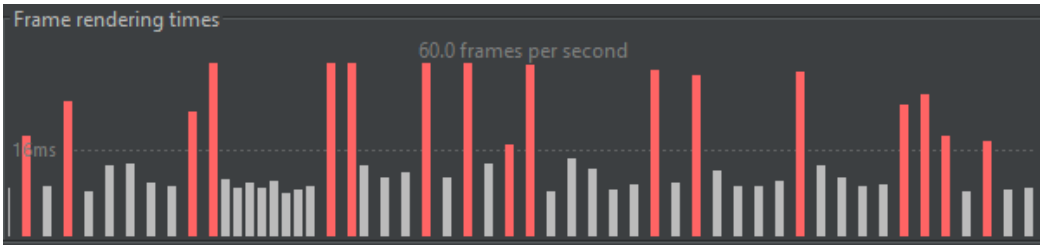
tionalitati aplicatiei precum: un sistem de search intre useri, posibilitatea de interactiune intre acestia prin aprecierea postarilor. Din punct de vedere al editarii imaginii cele 3 pachete utilizate mi-au conferit o usurinta in implementarea; pachetele au fost foarte bine documentate si exemplificate. Firebase a reprezentat in aceast proiect o alternativa pentru anumite dificultati tehnice precum un sistem de login securizat care sa implementeze si un token de autentificare atunci cand userul vrea sa intre din nou pe aplicatie.

5.2 Utilitate

Din punct de vedere al utilitatii, aplicatie reprezinta un demo al unei aplicatii mult mai ample asemanatoare cu Instagram, dar care pune accentul mai mare pe crearea de arta fotografica. Pentru a intari acest concept intentionez sa adaug pe viitor mai multe functionalitati precum comentarii la postari, notificari si mult mai multe filtre de editare a imaginii(hue,vibration,etc.).

5.3 Performante

Performantele aplicatiei, in punctul actual sunt bune pentru o aplicatie cross-platform. Este evident ca o aplicatie non nativa nu va fi niciodata la fel de "performanta" precum aplicatiile native, insa Flutter se aproprie foarte tare de cifrele scoase de aplicatiile native. Aplicatia mea reuseste sa scoata 60 frames per second, ceea ce este mai mult decat necesar pentru o aplicatie de mobile. Din punct de vedere al memory usage aplicatia foloseste undeva intre 55 si 75 MB, ceea ce este cat se poate de rezonabil pentru specificatiile telefoanelor actuale. Se pot observa in capturile urmatoare performantele in timpul rularii aplicatiei:



[fig 5.1] Graficul frame-rateului

[fig 5.2] Graficul Memory Usage / timp

6 Referinte bibliografice

1 https://flutter.dev/multi-platform/mobile

2 https://docs.flutter.dev/deployment/android

3 https://docs.flutter.dev/deployment/ios

4 https://pub.dev/packages/image_editor_pro

5 https://www.kindacode.com/article/flutter-colorfiltered/

6 https://www.esri.com/arcgis-blog/products/3d-gis/3d-gis/react-redux-building-

7 https://www.esri.com/arcgis-blog/wp-content/uploads/2017/09/react-redux-overview.png

8 https://firebase.flutter.dev/docs/auth/email-link-auth

9 https://firebase.flutter.dev/docs/firestore/usage/