Mobilna aplikacija za umrežavanje umjetnika

Novosel, Filip

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Filip Novosel

Mobilna aplikacija za umrežavanje umjetnika

Završni rad

Pula, rujan, 2022.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

Filip Novosel

Mobilna aplikacija za umrežavanje umjetnika

Završni rad

JMBAG: 0303061388, izvanredni student
Studijski smjer: Informatika

Predmet: Programsko inženjerstvo
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan, 2022.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani/a Filip Novosel, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj seminarski rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio seminarskog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student/ica

U Puli, 14.9.2022.

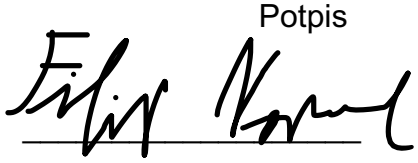
IZJAVA

o korištenju autorskog djela

Ja, Filip Novosel, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom „Mobilna aplikacija za umrežavanje umjetnika“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama. Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 14.9.2022.

Potpis



Mobilna aplikacija za umrežavanje umjetnika

Sažetak: Umjetnici današnjice uvelike se razlikuju od „klasičnih“ umjetnika o kojima svi pričaju, što je normalan proces razvoja društva i kulture. Današnji (mladi) umjetnici primorani su koristiti razne društvene mreže kako bi proširili svoj glas, bez centralnog tijela koje bi objedinilo umjetnike cijeloga svijeta. Cilj ovoga rada je ukazati na tu problematiku, te razraditi rješenje koje je pogodno za umjetnike, neovisno o njihovoj starosti, informatičkoj pismenosti, ili lokaciji.

Ključne riječi: ArtGallery, Flutter, Dart, Firebase, mobilna aplikacija, umjetnost, Android

Mobile application for connecting artists

Abstract: Artists of today are very different from „classic“ artists of whom everyone talks about, which is a normal process of cultural and social development. (Young) people of today are forced to use social networks to spread their voice, without any central entity which would unite them across the globe. The goal of this paper is to point out this issue, as well to create a solution that works for all artists, no matter their age, computer literacy, or location.

Keywords: ArtGallery, Flutter, Dart, mobile application, art, Android

Sadržaj

1. Uvod

Umjetnost je prisutna od samih početaka ljudi i civilizacije, na što ukazuju pronađeni dokazi grafika i slika u špiljama, arheološke iskopine, stari spisi, melodije i pjesme koje se provlače kroz generacije. Otisnuta (opipljiva) umjetnost ostavlja trajni pečat (u slučaju da nije uništena) iz kojega možemo iščitati kako su ljudi živjeli, dok je nedokumentirana ili neopipljiva umjetnost sklona interpretaciji.

Umjetnici su uvijek nalazili način da prošire glas o sebi kako bi drugi ljudi mogli uživati u njoj, time obogaćujući živote okoline. Razvojem tehnologije kroz generacije razvijala se i umjetnost, novi alati za slikanje, kiparenje, pa i fotografija.

Do početaka interneta, koristili su se analogni mediji za oglašavanje (ne samo umjetničkih izložbi) jer su to bili jedini načini. Kasnije dolazi radio, TV, telefon, pa i sam Internet koji generira problematiku ovog rada.

Tko dobro promotri umjetničku scenu, može primijetiti da je „stala“ te je i dalje jedan od naj jačih medija prijenosa informacija usmena predaja, ma koliko sadržaja bilo na internetu. Također, veliki problem predstavlja činjenica da su sve informacije o izložbama „razbacane“ na društvenim mrežama, portalima i blogovima bez ikakve centralne jedinice.

Ovaj rad će prikazati kako bi jedna takva aplikacija izgledala, kako bi umjetnici mogli iskoristiti takvu aplikaciju, te ono najbitnije, znali čija izložba se održava, kada i gdje. Cilj je pojednostavniti korištenje informacija u svrhu navigacije do izložbe, generiranje podsjetnika u kalendar te komunikacija sa drugim korisnicima kroz komentare.

Ovaj završni rad će u drugom poglavlju pobliže objasniti koje tehnologije su korištene pri izradi aplikacije. U trećem poglavlju razrađujemo samu aplikaciju, njen dizajn, implementaciju i korištenu arhitekturu.

Izvorni kod aplikacije nalazi se na: https://github.com/fico45/ArtGallery

2. Tehnologije i alati

Za ovaj rad korišten je Flutter za razvoj mobilne aplikacije, te Firebase koji nudi Backend te bazu podataka.

2.1. Flutter

Flutter je alat za izradu korisničkog sučelja baziran na Dart-u te je proizveden od strane Google-a, prvi puta predstavljen 2015. godine, a prva inačica je izdana 2017. godine. Flutter nudi opciju pisanja jednog koda (single-codebase) te kompajliranje na više platformi. U trenutku pisanja, službena dokumentacija nalaže da se Flutter može kompajlirati u:

- Android
- iOS
- Windows
- Web
- Google Fuchsia
- macOS (BETA)
- Linux (BETA) [20]

Kako bi postigao izvršavanje na više platformi, Flutter koristi Skia Graphics Engine (pisano u C++), te iako se kompajlira u nativni jezik platforme te koristi nativne biblioteke, iscrtavanje (render) se izvršava u vlastitom Canvas-u, koji stoji „iznad“ glavnog Activity-a, zvan „Flutter Activity“.

2.2. Dart

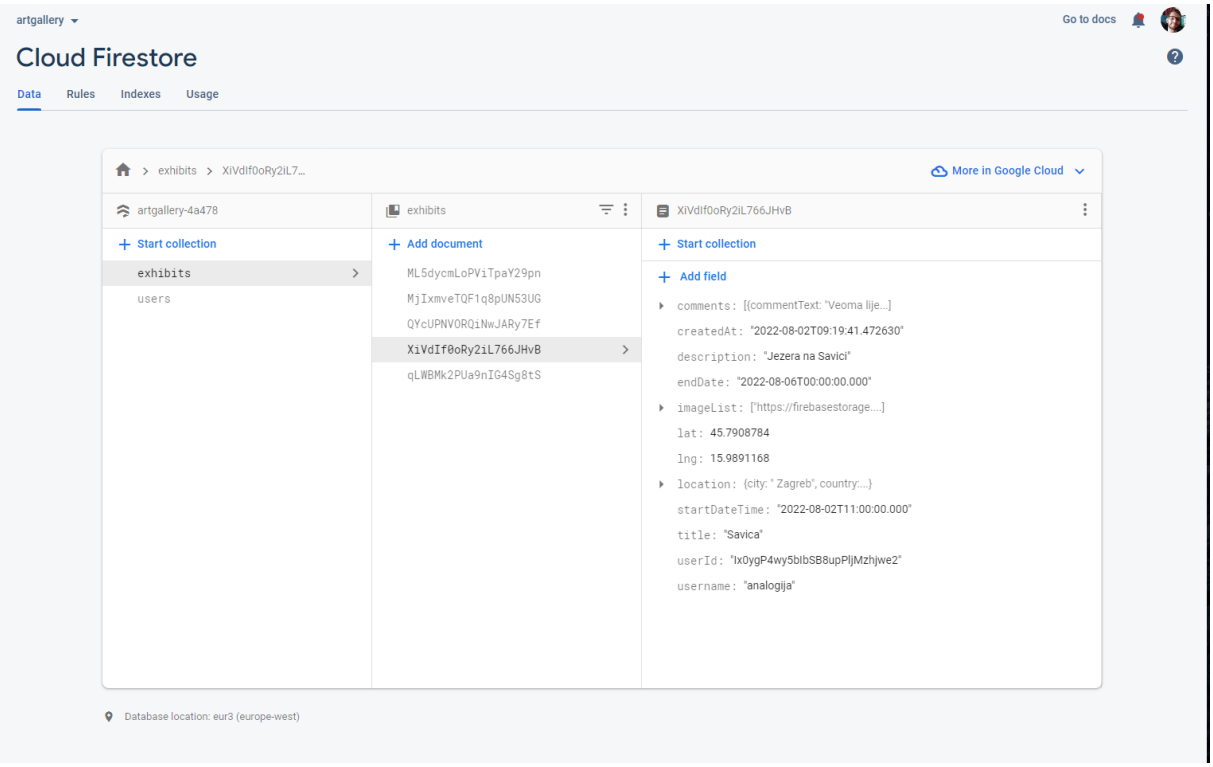
Dart je jezik kojeg je Google izdao za razvoj Client-side aplikacija. Također postoji mogućnost izrade server i desktop aplikacija u Dart-u, iako to nije njegova primarna namjena. Objektno-orijentirani je jezik, a sintaksa se ne razlikuje previše od C-a. Prvi je puta predstavljen 2011. godine u „GOTO“ konferenciji u Danskoj. Prvi verzija dolazi 2013. godine. Inicijalno, Google je imao plan usmjeriti Dart eksplicitno WEB

developmentu, u smislu kompajliranja u JavaScript, kako bi ostvario dominaciju na WEB tržištu; prisjetimo se da je Angular također Googleov proizvod usmjeren ka WEB developmentu.

2.3. Firebase

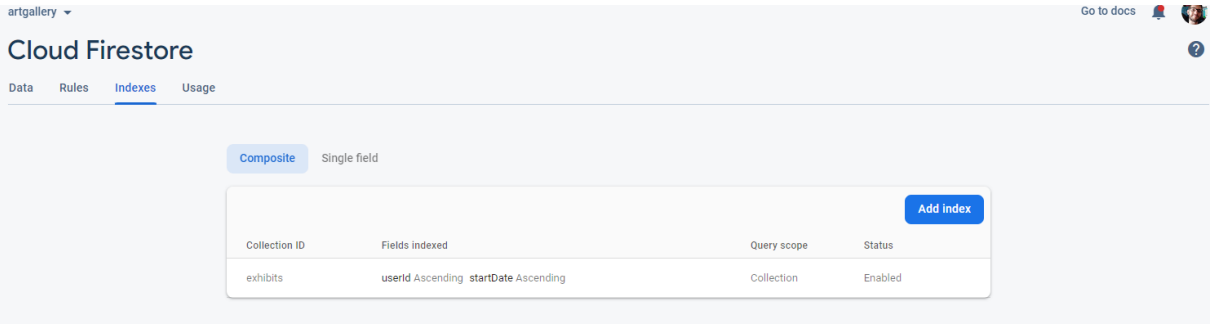
Firebase [1] je također Google-ov proizvod, te nudi širok spektar alata i usluga koji developerima olakšava život. Najčešće se implementira jako brzo i lako; importa se ekstenzija za jezik u kojem se razvija aplikacija, potom Firebase generira ispravan dokument za tu platformu sa potrebnim varijablama (API ključevi, storage bucket ID, informacije o Firebase projektu, itd.) kako bi aplikacija mogla komunicirati sa Firebase-om. Trenutno postoje dva naplatna plana, Spark (besplatan, manja količina dostupnih feature-ova) te Blaze (pay-as-you-go plan, besplatan do trenutka kada se premaši kvota). Za trenutni projekt, potrebni su bili Firebase Authentication [2] (implementacija prijave i registracije korisnika), Firebase Storage [3] (pohranjivanje fotografija profila te fotografija/slika izložbi), Firestore Database [4] (baza podataka aplikacije) te Analytics (radi prikupljanja raznih metrika o korisnicima).

Na slici 1 vidi se struktura baze podataka, te primjetiti da se ne radi o relacijskoj bazi podataka, već o NoSQL bazi, koja sadrži kolekcije i dokumente. Također, dokument unutar određene kolekcije također može sadržavati kolekciju. Time se omogućava skaliranje baze/aplikacije.



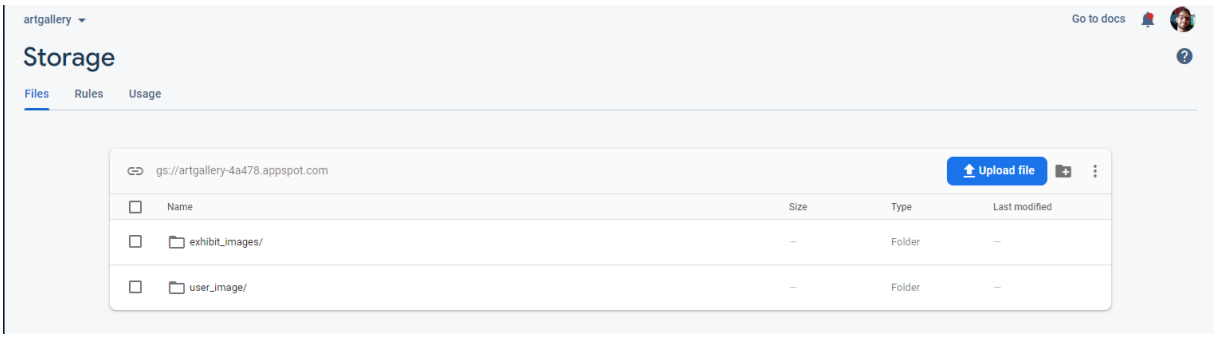
Slika 1: Firebase Firestore Database

Također, slika 2 prikazuje ručno napravljena pravila za indeksiranje te serviranje podataka. Pri kreiranju projekta, Firebase sam „zaključuje“ koje pretrage vršimo najčešće te ih automatski indeksira, no treba imati na umu da će to raditi samo i samo za jednostavne upite. U slučaju da želimo izvršiti neki kompleksniji upit, te u samome kodu definiramo upit, doći će do pogreške u runtime-u te će konzola napomenuti kako treba dodati ručno definirano pravilo na sam Firebase (također će i priložiti link kako bi se to jednostavno i odmah napravilo). Kroz koju minutu, kada Firebase „pripremi“ pravilo, upit će se uspješno izvršiti. Samim indeksiranjem podataka omogućava se veoma brza isporuka podataka klijentu.



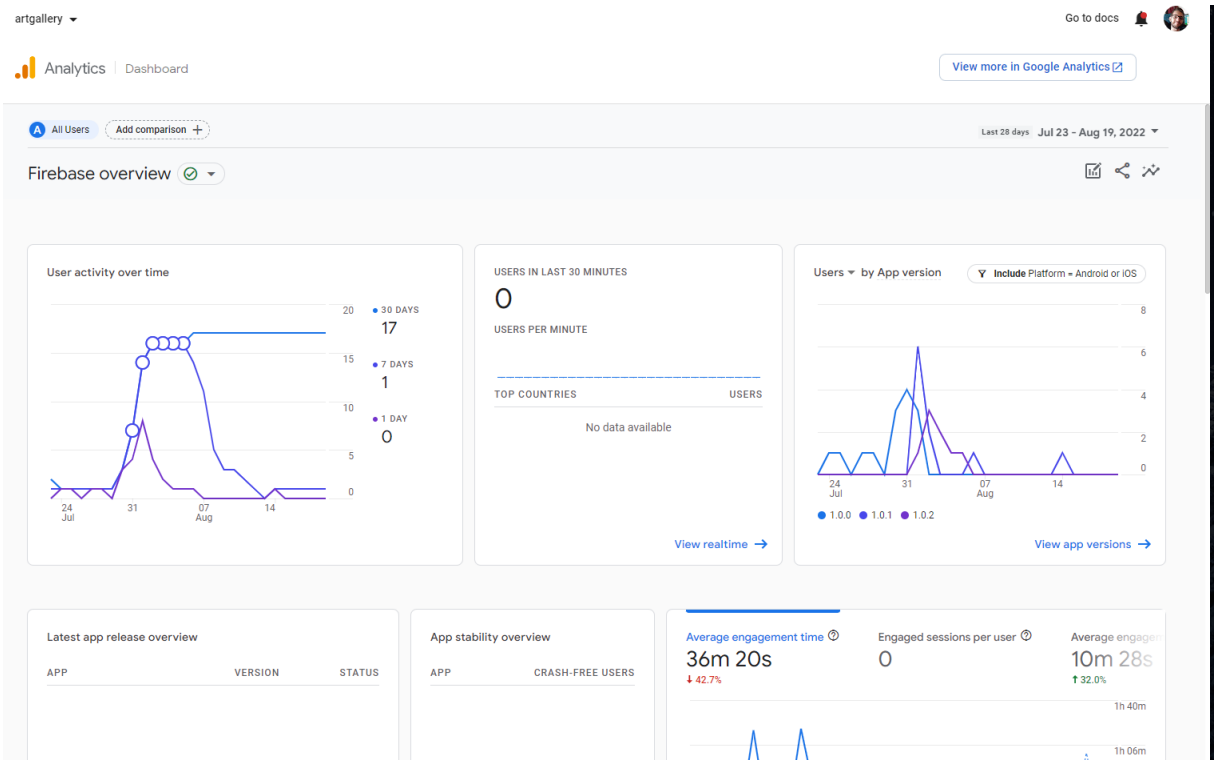
Slika 2: Firebase Firestore indexi

Ako je potrebno pohraniti neki dokument, a nije prikladno pohraniti ga u bazu podataka, koristiti će se Firebase Storage. Na slici 3 vidljiva je struktura Firebase Storage-a, koja ima definirani „bucket“ u koji se može pohraniti dokumenti. Za potrebe projekta, ovdje se pohranjuju slike izložbi te profilne fotografije korisnika.



Slika 3: Firebase Storage

Google Analytics je također moguće koristiti kroz Flutter, te će i to ići kroz Firebase. Na slici 4 možemo vidjeti dashboard analitike, te link na Google Analytics gdje možemo pobliže raditi sa prikupljenim podacima. Boljka Flutter-a i Firebase-a je što su poprilično integrirani; čak i sam Google predlaže korištenje Firebase-a pri razvijanju Flutter aplikacije. Nakon inicijalnog podešavnja (importanje ekstenzije te dokumenta sa potrebnim API ključevima) Firebase je već spreman za rad, te je u mogućnosti prikupiti analitiku o navigaciji korisnika kroz aplikaciju, učestalost njihovog korištenja, a po potrebi se mogu definirati i custom eventi. Dovoljno je samo pozvati jednu funkciju sa imenom eventa, te prepustiti Firebase-u da odradi ostalo.



Slika 4: Firebase Analytics

2.3.1 JSON seriazible i Freezed

U svrhu pojednostavljenja izrade modela podataka te ulaznih i izlaznih transformacija, koristi se paket Freezed [5], čija implementacija je vidljiva na slici 5, koji će dodati funkcionalnosti koje sam Dart nema, na primjer unije i *copyWith() [6]* metodu koja se koristi kada želimo kopirati postojeći objekt u novi uz određene izmjene podataka, a pritom se izvorni objekt ne mijenja. Dovoljno je dodati ključnu riječ *@freezed* ispred klase kako bi aplikacija prepoznala da treba koristiti Freezed paket u svrhu generiranja modela. Model podataka često sadrži, osim same definicije klase, ulazne i izlazne transformacije (*fromJson, fromDocument, toJson, toDocument*), *copyWith()* metodu, te (ponekad) i metode koje izvršavaju određene operacije sa objektom, a nisu povezane sa transformacijama. Freezed će, nakon što definiramo klasu, automatski generirati sve potrebne metode za primanje te slanje podataka izvan domene aplikacije, recimo na vanjski server preko API-a. Sama ekstenzija uključuje i JSON seriazible [7] i JSON annotation [8] ekstenziju kako bi dodatno pojednostavio transformacije, čija implementacija se također vidi na slici 5. To se postiže definiranjem

prilagođenih ključeva u JSON-u. Za „createdAt“ polje bilo je potrebno definirati posebnu pomoćnu metodu *_sentAtFromJson* radi manjka kompatibilnosti sa *DateTime* tipom podatka između Firebase-a i Dart-a. Time se osigurava da se *Timestamp* (kojeg Firebase koristi) bude transformiran u *DateTime*, sa kojim se mogu raditi operacije sortiranja, dodavanja, oduzimanja, itd.

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:freezed_annotation/freezed_annotation.dart';

part 'user_data_model.freezed.dart';
part 'user_data_model.g.dart';

DateTime _sendAtFromJson(Timestamp timestamp) => timestamp.toDate();

@freezed
abstract class UserModel implements _$UserModel {
  const UserModel._();

  const factory UserModel({
    required String email,
    required String image_url,
    required String username,
    required String firstName,
    required String lastName,
    required bool reviewable,
    @JsonKey(name: 'createdAt', fromJson: _sendAtFromJson)
        required DateTime createdAt,
    required String bio,
    required List<String> favorites,
  }) = _UserModel;

  factory UserModel.fromJson(Map<String, dynamic> json) =>
      _$UserModelFromJson(json);

  factory UserModel.fromDocument(DocumentSnapshot doc) {
    final data = doc.data()! as Map<String, dynamic>;
    return UserModel.fromJson(data);
  }
  Map<String, dynamic> toDocument() => toJson();
}
```

Slika 5: ArtGallery implementacija JSON Seriazible i Freezed

Kada se sve potrebno za model podataka definira, moguće je pokrenuti skriptu koja će generirati potrebne datoteke koje sadrže generirane funkcije. Freezed će generirati **.freezed.dart* datoteku, dok će JSON seriazible generirati **.g.dart datoteku*.

3. ArtGallery

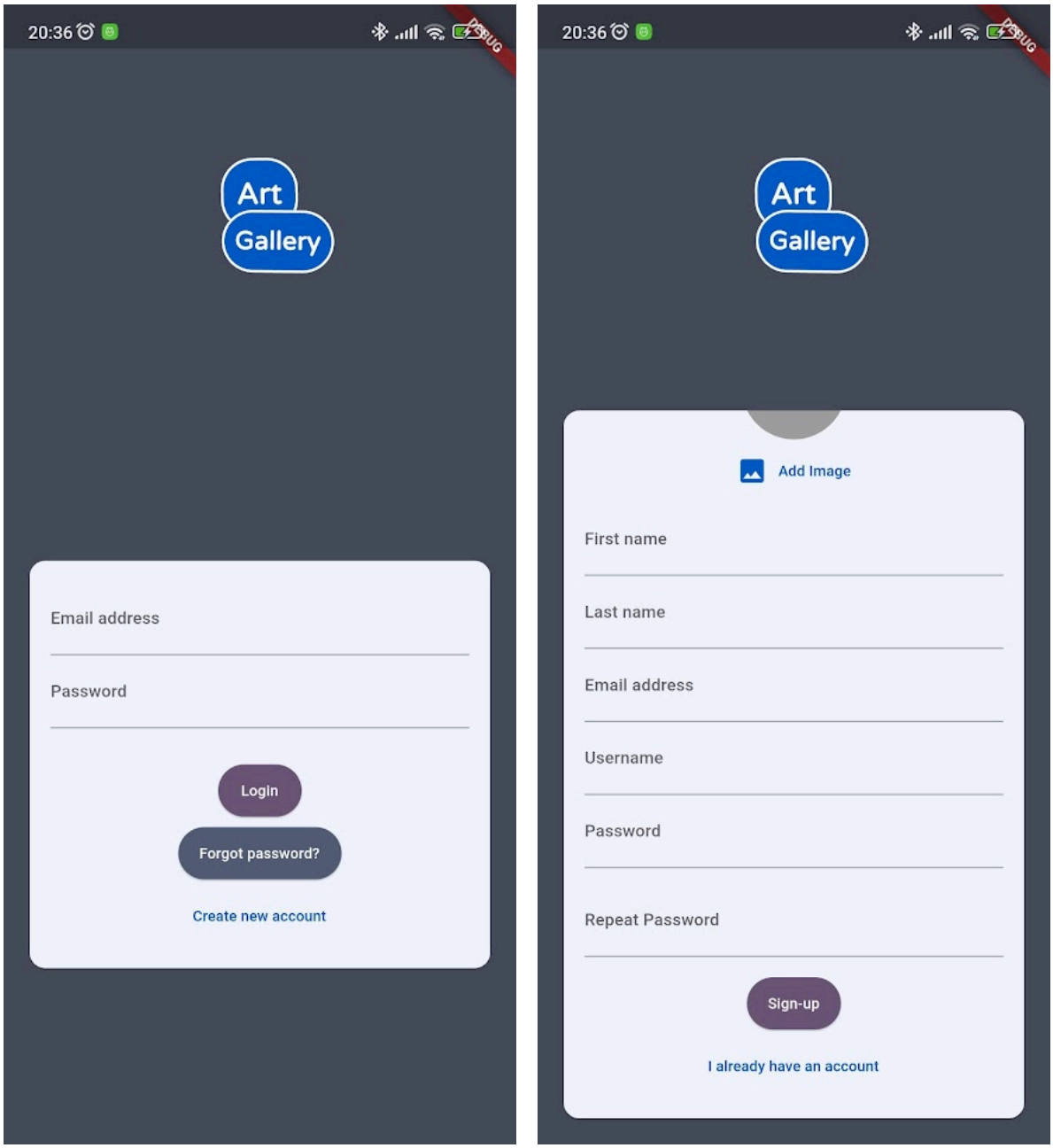
Kod izrade aplikacije, slijeđene su Material Design 3 (Material Design 3, n.d.) smjernice, koje uključuju i Material You (Material You, n.d.). Svi elementi u aplikaciji bazirani su na Android korisničkom sučelju, te će kao takvi biti prezentirani.

Sama aplikacija sastoji se od:

- Login/Registrer page-a
- Dashboard-a sa pregledom izložbi
- Prikaz detalja o izložbi
- Forme za dodavanje/uređivanje izložbe
- Prikaz profila korisnika
- Forma za uređivanje profila
- „Explore“; interaktivna karta sa lokacijama izložbi

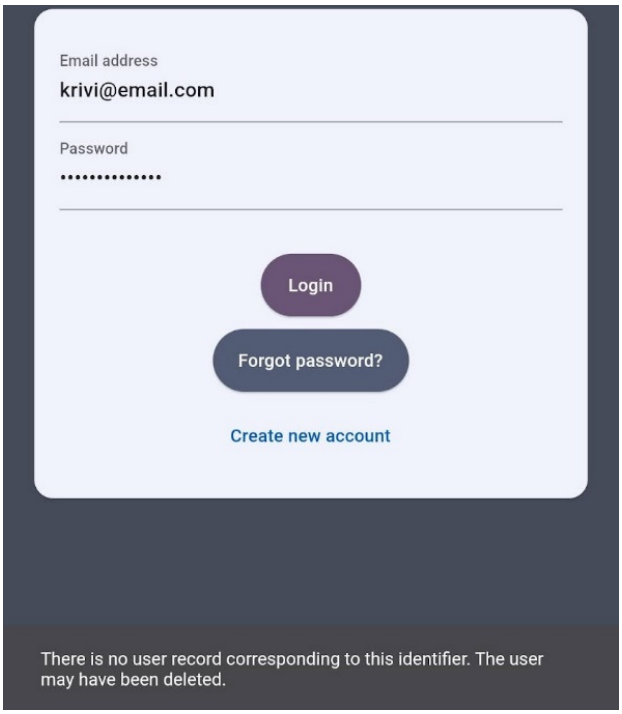
3.1. Autentikacija

Prilikom pokretanja aplikacije, dočekati će nas Authentication page, gdje se ujedno možemo i registrirati u slučaju da još nemamo korisnički račun, koje možemo vidjeti na slici 6. Login traži unos ispravne e-mail adrese i lozinke (lijevo), dok registracija zahtjeva unos e-mail adrese, korisničkog imena, lozinke, ponovljene lozinke, te fotografija koja će reprezentirati korisnika (desno).



Slika 6: Lijevo: ArtGallery Login page, desno: ArtGallery forma za registraciju

Ako se pokuša napraviti login sa nepostojećim računom, prikazati će se prigona poruka, što vidimo na slici 7.



Slika 7: ArtGallery login sa nepostojećim korisnikom

Za samu autentikaciju koristi se *signInWithEmailAndPassword* funkcija koju nudi Firebase Auth, te je implementacija prikazana na slici 8.

```
if (isLogin) {
  try {
    authResult = await _auth.signInWithEmailAndPassword(
        email: email, password: password);
    widget.callback(newRegistrationStatus: true);
  } on FirebaseAuthException catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(
      content: Text(e.message.toString()),
    )); // SnackBar
    setState(() {
      _isLoading = false;
    });
  }
} else {
```

Slika 8: ArtGallery funkcija za autentikaciju

Nakon uspješnog poziva na Firebase, vraća se instanca objekta koji je spremljen u „state“ (stanje) aplikacije. To će nam omogućiti Riverpod [9] provider (dalje u tekstu:

Provider) pomoću kojega je moguće distribuirati podatke, stanja ili tokove aplikacija. Bez Riverpoda, Flutter nije u mogućnosti obavljati takvu funkciju na globalnoj razini aplikacije, već unutar samih widget-a, te se stanja i podaci mogu prosljeđivati eksplicitno kroz parent-child odnos, što uvelike otežava skaliranje aplikacije, te sami pristup podacima na drugim mjestima aplikacije. Primjerice, na slici 9 se može vidjeti implementacija autentikacijskog repozitorija koji sadrži stanje autentikacije unutar aplikacije, te su podaci (stanje) dostupni kroz cijelu aplikaciju.

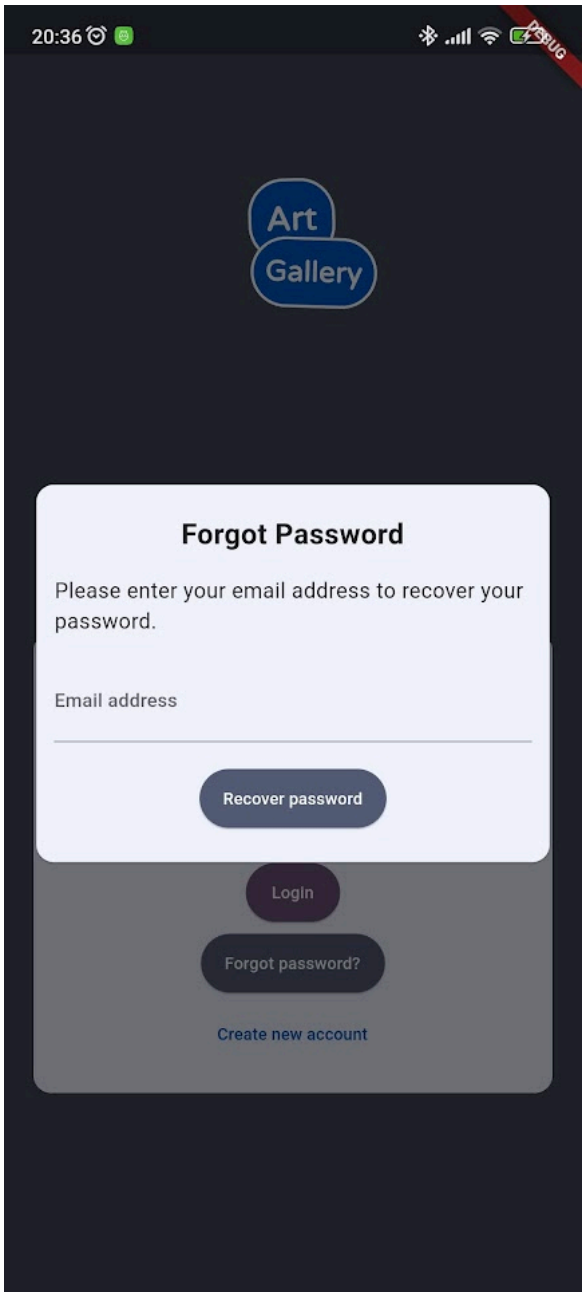
```dart
import 'package:artgallery/data/providers/general_providers.dart';
import 'package:artgallery/repositories/custom_exception.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_riverpod/flutter_riverpod.dart';

Filip Novosel, 5 months ago | 1 author (Filip Novosel)
abstract class BaseAuthRepository {
  Stream<User?> get authStateChanges;
  User? getCurrentUser();
  Future<void> signOut();
}

final authRepositoryProvider =
    Provider<AuthRepository>((ref) => AuthRepository(ref.read));

Filip Novosel, 5 months ago | 1 author (Filip Novosel)
class AuthRepository implements BaseAuthRepository {
  final Reader _read;

  AuthRepository(this._read);

  @override
  Stream<User?> get authStateChanges =>
      _read(firebaseAuthProvider).authStateChanges();

  @override
  User? getCurrentUser() {
    try {
      return _read(firebaseAuthProvider).currentUser;
    } on FirebaseAuthException catch (e) {
      throw CustomException(message: e.message);
    }
  }
      Filip Novosel, 5 months ago • Repository and .env introduced
  @override
  Future<void> signOut() async {
    try {
      await _read(firebaseAuthProvider).signOut();
    } on FirebaseAuthException catch (e) {
      throw CustomException(message: e.message);
    }
  }
}
```

Slika 9: ArtGallery autentikacijski repozitorij

Dovoljno je samo pozvati Provider koji je potreban, te čitati vrijednost (ili pozvati njegovu funkciju). Preko njega možemo čitati da li je trenutni korisnik prijavljen ili nije. Također, u njemu se nalaze i dvije funkcije, *getCurrentUser* i *signOut*. Prva će vratiti objekt sa podacima trenutno prijavljenog korisnika, dok će druga napraviti odjavu iz aplikacije, te preusmjeriti korisnika na formu za prijavu i registraciju. Pri registraciji korisnika će se pozvati funkcija sa slike 10. Prvo će se napraviti novi korisnik na Firebase Auth instanci, te pohraniti u Provider stanje prijavljenog korisnika. U slučaju da funkcija baci grešku, recimo, ako se korisnik pokušava registrirati sa već postojećom email adresom, odmah se izlazi iz funkcije te se odlazi u *catch* blok, koji odrađuje prikaz pogreške. Ako je uspješna registracija, tada se pohranjuje fotografija profila koju je korisnik odabrao u Firebase Storage, te se u Firebase Firestore pravi zapis korisnika sa potrebnim informacijama radi simplifikacije (Firebase Auth nema mogućnost pohraniti prilagođene podatke).

Slika 10: ArtGallery funkcija za registraciju korisnika

U slučaju da se korisnik ne sjeća lozinke, može zatražiti link za resetirane lozinke dodirom na „Forgot password?“ dugme, te unjeti svoj email u formi na slici 11.



Slika 11: ArtGallery forma za resetiranje lozinke

To će pozvati Firebase Auth funkciju za resetiranje lozinke, vidljiva na slici 12.

```
onPressed: () async {
  await _auth
      .sendPasswordResetEmail(email: _email)
      .then(
        (value) => showDialog(
```

Slika 12: Funkcija za resetiranje lozinke

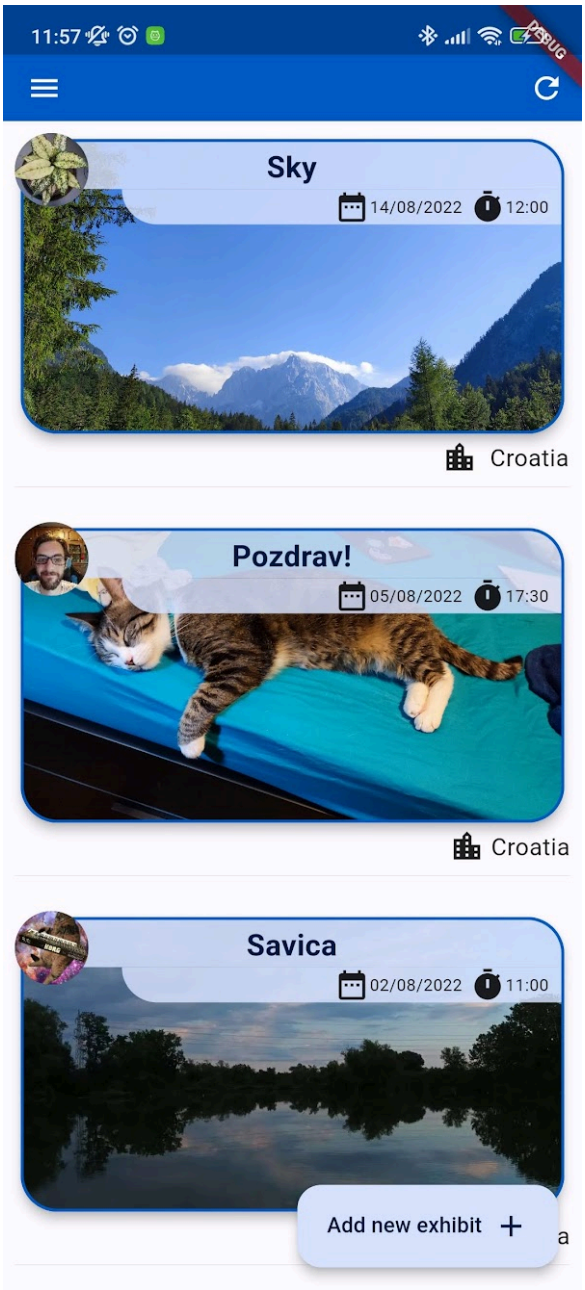
Glavna provjera stanja autentikacije korisnika odvija se u samom korijenu aplikacije, takozvanoj *main.dart* datoteci. Sama implementacija se nalazi na slici 13, te prikazuje *StreamBuilder*, widget koji čita tok podataka i servira ga aplikaciji. Unutar samog widgeta, nalazi se *builder* koji će prikazati na ekranu druge widgete ovisno o trenutnom stanju toka podataka. Prva provjera je *userSnapshot.hasData* koja će biti **TRUE** tek nakon uspješne autentikacije. Druga provjera (*registartionComplete)* je dodana kao pomoćna varijabla, te je ona inicijalizirana sa **TRUE**. Tek kada se pozove funkcija za registraciju (znači, nakon popunjene forme za registraciju) će se *registrationComplete* varijabla postaviti na **FALSE**, i biti će dok se cijela funkcija ne izvrši, kako ne bi došlo do automatske navigacije u Dashboard prije nego su se svi podaci poslali na Firebase Firestore, s obzirom da se FIrebase Auth funkcija za registraciju korisnika izvršava puno brže od samog procesa pohrane korisničke fotografije na Firebase Storage instanci i kreiranja korisničkog objekta na Firebase Firestore instanci. Kada su svi uvjeti zadovoljeni, može se pristupiti aplikaciji.

```
49    Widget build(BuildContext context) {
50      final authControllerState = ref.read(authControllerProvider.notifier);
51
52      return MaterialApp(
53        title: 'Flutter Demo',
54        theme: ThemeData(
55          useMaterial3: true,
56          colorScheme: ColorScheme.fromSeed(seedColor: Colors.blueAccent),
57          //ColorScheme.fromSeed(seedColor: Colors.green),
58        ), // ThemeData
59        home: StreamBuilder(      fico45, 14 months ago • Initial push
60          stream: authControllerState.stream,
61          builder: (ctx, userSnapshot) {
62            if (userSnapshot.hasData && registrationComplete == true) {
63              return CustomDrawer();
64            }
65            return AuthScreen(callback: _setRegistrationComplete);
66          },
67        ), // StreamBuilder
68        routes: {
69          NewExhibit.routeName: (context) => NewExhibit(),
70          ProfileView.routeName: (context) => ProfileView(),
71        },
72      ); // MaterialApp
73    }
74  }
```

Slika 13: ArtGallery implementacija logike provjere autentikacije korisnika

3.2. Dashboard

Nakon što smo se uspješno registrirali i prijavili u aplikaciju, možemo pogledati ponuđene izložbe. Njih ćemo vidjeti u Dashboard-u, gdje će izložbe biti kronološki sortirane, a to nam omogućuje Firebase indeksiranje upita. Prikaz Dashboard-a možemo vidjeti na slici 14.



Slika 14: ArtGallery Dashboard

Zapise o izložbama ćemo povući sa Firebase Firestore-a, i to pozivom preko repozitorija za izložbe. Slika 15 prikazuje definiciju repozitorija koji se koristi za CRUD

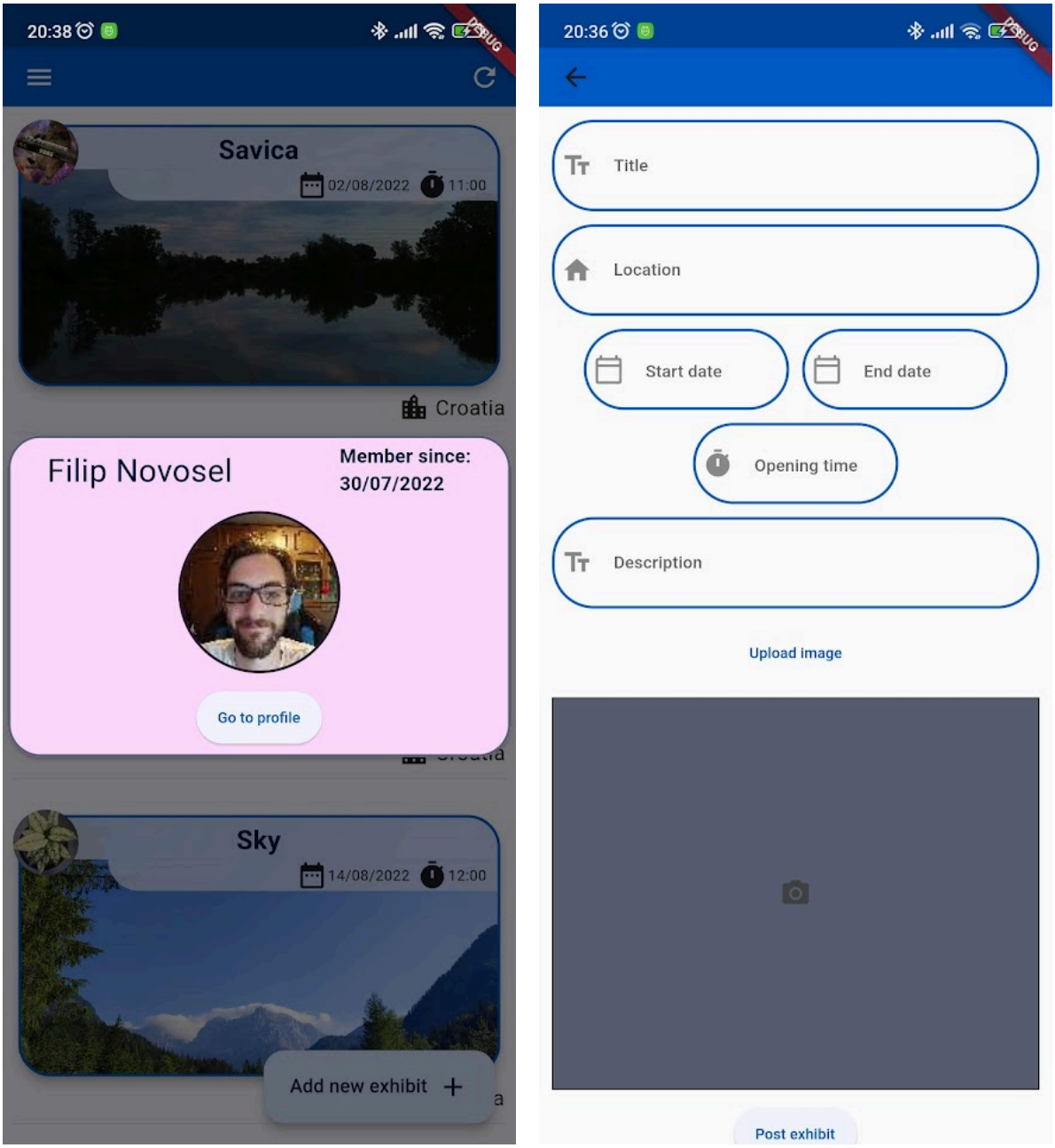
operacije izložbi. Implementacija se radi unutar same datoteke u *ExhibitRepository* klasi, gdje će biti implementirane sve funkcije koje su definirane u *BaseExhibitRepository* klasi. Između njih se nalazi i *exhibitRepositoryProvider* varijabla, koja pokazuje na Provider tipa *ExhibitRepository* kako bi se zadržalo stanje izložbi te koristilo kroz aplikaciju. Pozivom na *retrieveAllExhibits* dohvaća se lista objekata o izložbama, te se sortiraju silazno po vremenu početka (najnovije izložbe su prve).

```dart
abstract class BaseExhibitRepository {
  Future<List<Exhibit>> retrieveAllExhibits({required String userId});
  Future<List<Exhibit>> retrieveCurrentUserExhibits({required String userId});
  Future<String> createExhibit({required Exhibit exhibit});
  Future<void> updateExhibit(
      {required String userId, required Exhibit exhibit});
  Future<void> deleteExhibit(
      {required String userId,
      required String exhibitId,
      required List<String> imagesToDelete});
  Future<void> postExhibitComment(
      {required String exhibitId, required Comment comment});
}

final exhibitRepositoryProvider =
    Provider<ExhibitRepository>((ref) => ExhibitRepository(ref.read));

Filip Novosel, 2 weeks ago | 2 authors (Filip Novosel and others)
class ExhibitRepository implements BaseExhibitRepository {
  final Reader _read;

  const ExhibitRepository(this._read);

  @override
  Future<List<Exhibit>> retrieveAllExhibits({required String userId}) async {
    try {
      final snap = await _read(firebaseFirstoreProvider)
          .collection('exhibits')
          .orderBy('startDateTime', descending: true)
          .get();
      return snap.docs.map((doc) => Exhibit.fromDocument(doc)).toList();
    } on FirebaseException catch (e) {
      throw CustomException(message: e.message);
    }
  }
```

Slika 15: ArtGallery definicija i implementacija repozitorija za izložbe

Dodirom na profilnu ikonu korisnika, otvoriti će nam se mala kartica profila tog korisnika, što vidimo na slici 16 (lijevo). Ako dodirnemo karticu izložbe, biti ćemo

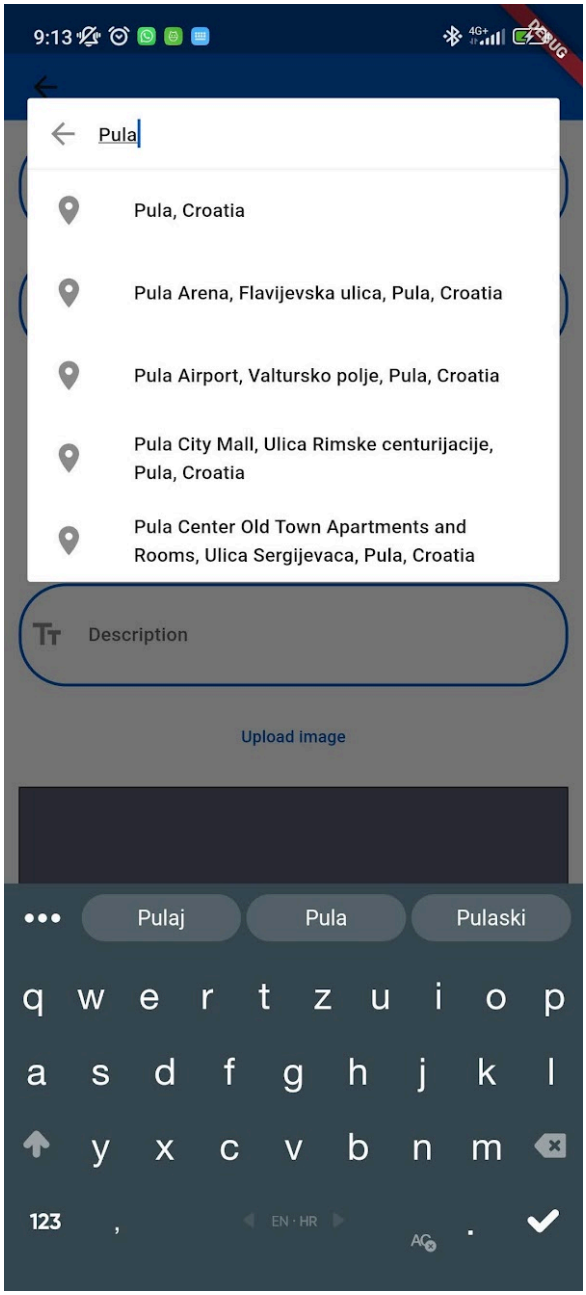
odvedeni na prikaz detalja o izložbi. Također, moguće je izraditi i novu izložbu dodirom na „Add new exhibit +“ dugme što otvara formu za dodavanje izložbe, vidljivo na slici 16 (desno).



Slika 16: Lijevo: ArtGallery kartica profila, desno: ArtGallery izrada nove izložbe

Title i Description su jedina polja koja primaju običan tekstualni input. Nakon što dodirnemo „Location“ polje, otvoriti će se Google Places Picker [10] koji će automatski predlagati adrese na temelju inputa. Ponuđene adrese su takozvani „Prediction“-i, te ako odaberemo jednu od ponuđenih adresa, Google vraća puni objekt sa detaljima te

adrese. Time se osigurava da ne dolazi do problema sa unosom adresa, da se poštuje forma (model podataka) adrese i olakšava se korisnicima unos same adrese, što se vidi na slici 17.



Slika 17: ArtGallery Google Address Picker

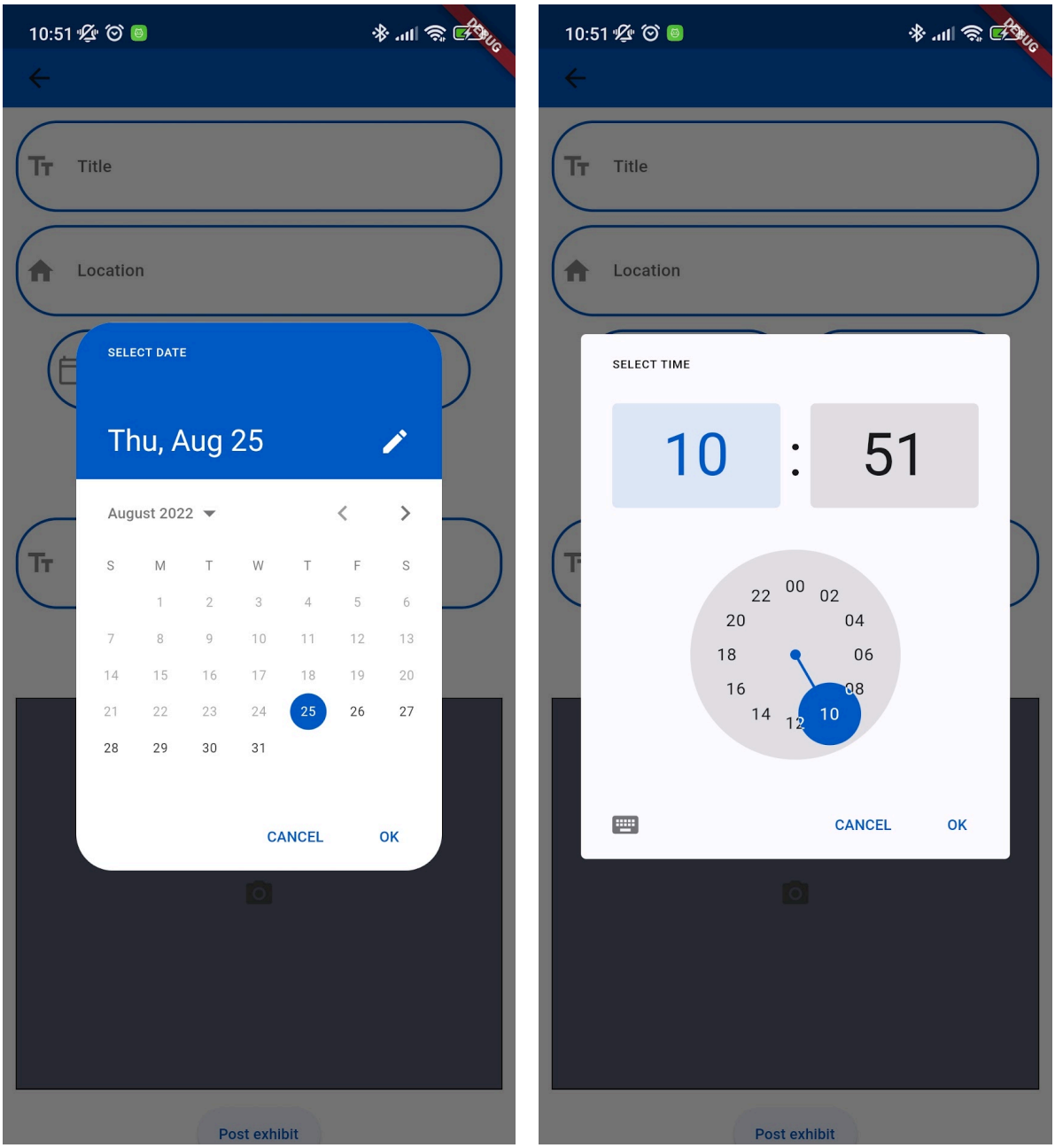
Treba napomenuti kako Google Places Picker ne može komunicirati sa Google-om bez API ključa koji se dohvaća sa Googleove konzole. S obzirom da se izvorni kod nalazi na javnom repozitoriju, nepoželjno je držati bilo kakve ključeve vidljive svijetu. Zato se koristi dotenv [11] ekstenziju koja čita ključeve iz .env datoteke, te koja se ne

šalje na repozitorij već se čuva lokalno na računalu. Primjer čitanja jednog ključa prikazan je na slici 18.

```
final _places = GoogleMapsPlaces(apiKey: dotenv.env['google-api-key']!);
```

Slika 18: dotenv čitanje ključa

Odabir datuma i vremena ostvaruje se pomoću Date Time Picker-a [12], čija se implementacija nalazi na slici 19. Start i End date (datumi početka i kraja izložbe) pozivaju DatePicker widget (lijevo) gdje korisnik može odabrati datum na interaktivnom kalendaru. Pritiskom na „OK“ promjene se spremaju. Samo vrijeme početka se može definirati dodirom na „Start Time“ što će otvoriti interaktivni widget za odabir vremena početka izložbe, tj. promocije (desno).



Slika 19: Lijevo: ArtGallery Date Picker, desno: ArtGallery Time picker

Za upload slika/fotografija, korisnik može dodirnuti dugme „Upload images“ što će pozvati Image Picker [13] widget, te dozvoliti korisniku pohranu slike iz galerije svoga telefona. Sam poziv ostvaruje se putem *pickImage* funkcije koja će korisniku prikazati

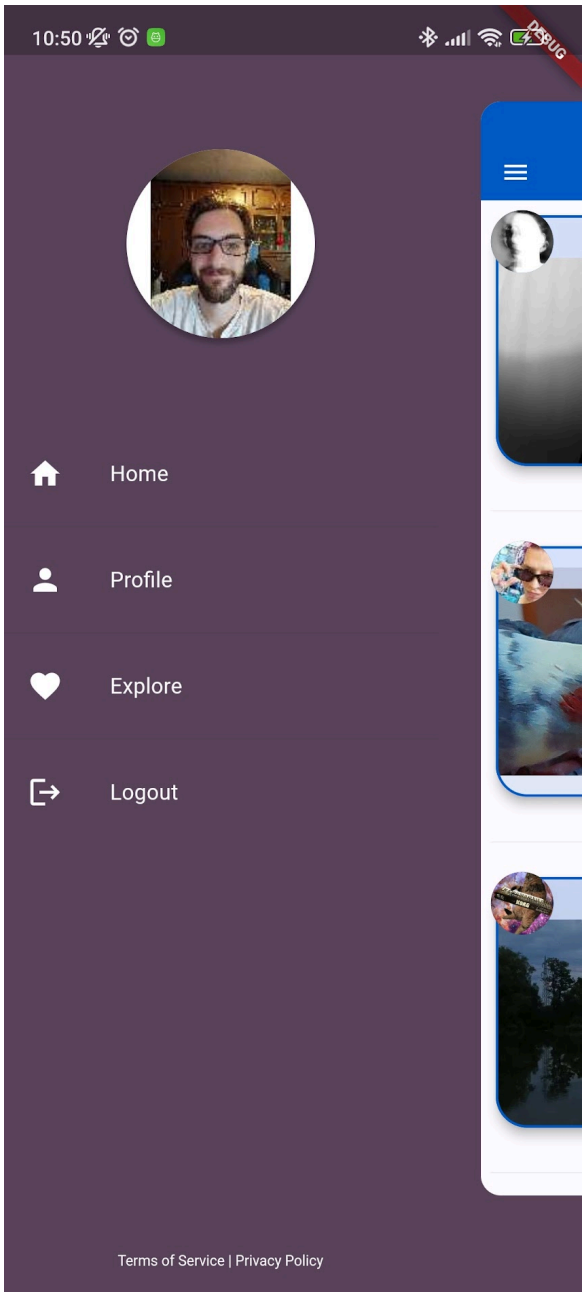
sučelje sa sadržajem galerije, gdje korisnik može odabrati sliku koju će pohraniti uz izložbu, čija implementacija je vidljiva na slici 20.

```
TextButton(
  onPressed: () async {
    String imageUrl;
    XFile? newPicker = await _picker.pickImage(
        source: ImageSource.gallery, imageQuality: 60);
    if (newPicker != null) {
      imageUrl =
          await uploadExhibitImage(currentUser!, newPicker);
      setState(() {
        _images.add(imageUrl);
        _unsubmittedImages.add(imageUrl);
      });
    }
  },
  child: Text('Upload image'),
), // TextButton
```

Slika 20: Poziv Image Picker-a

3.3. Ladica za navigaciju (navigation drawer)

Kroz aplikaciju je moguće navigirati koristeći Flutter Advanced Drawer [14] (dalje u tekstu: ladica); ladicu koja se izvlači sa lijeve strane te koja sadrži rute (navigaciju) na određene segmente aplikacije (slika 21). Navigacijska ladica je prisutna samo na „prvoj razini“ navigacije, te joj se ne može pristupiti u rutama koje su iznad prve razine, npr. prikaz detalja izložbe, uređivanje profila, dodavanje nove izložbe.

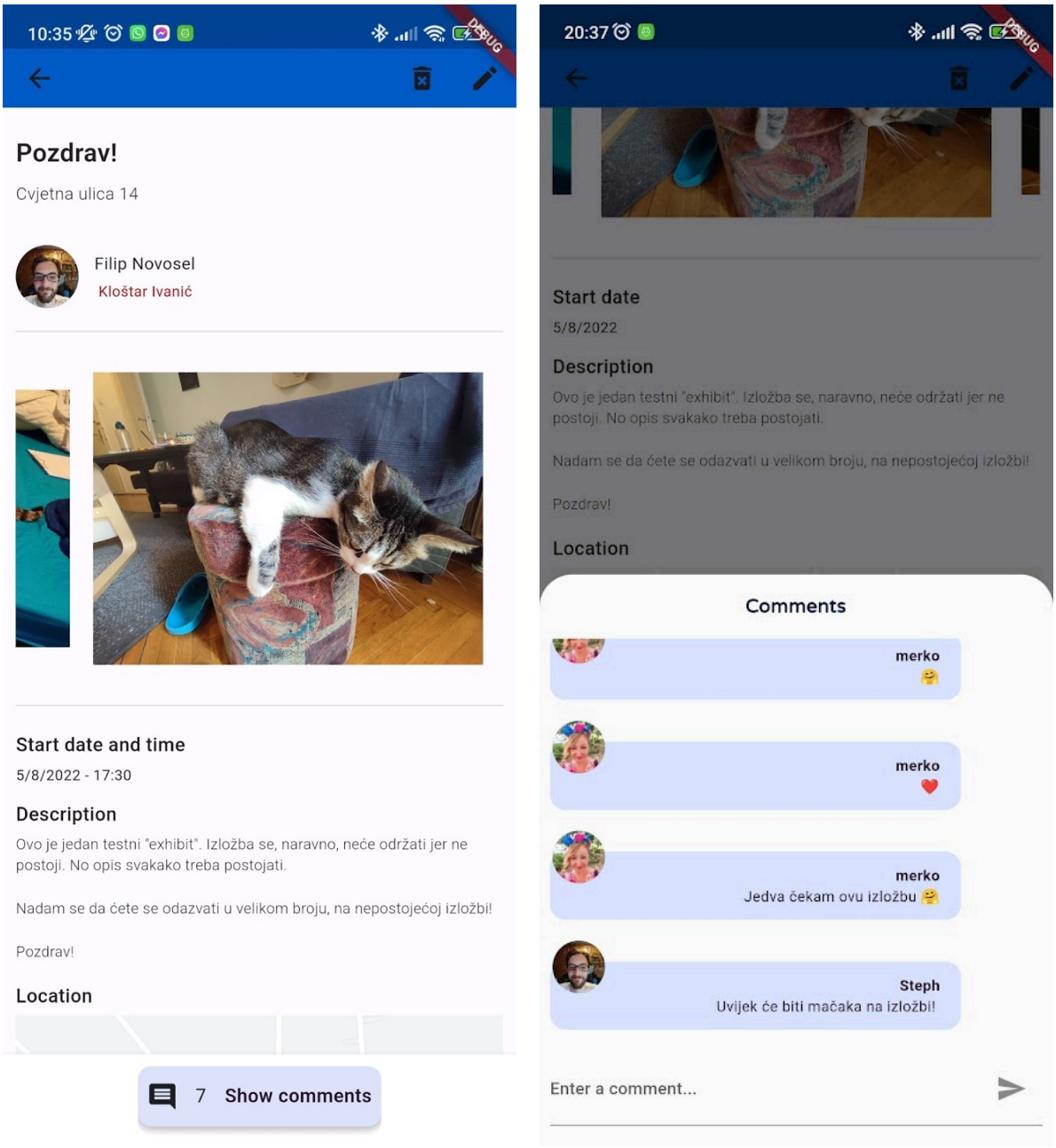


Slika 21: Flutter Advanced Drawer ladica za navigaciju

3.4. Prikaz detalja izložbe

Ovdje možemo pogledati sve detalje vezane uz izložbu; naslov izložbe, adresu, vlasnika izložbe, datum i vrijeme početka izložbe. Fotografije se nalaze unutar „carousell“-a, te se automatski mijenjaju. Prikaz detalja izložbe je vidljiv na slici 22 (lijevo).

Također, na samome dnu se nalazi dugme „Show comments“ sa brojem komentara koji izložba sadrži. Dodirom na gumb, otvara se Sheet koji sadrži sve komentare napisane na izložbu, također vidljivo na slici 22 (desno). U slučaju da još nema komentara, prikazuje se poruka na sredini sheeta „No comments yet!“, Ovdje je također moguće dodirnuti profilnu fotografiju korisnika, te pozvati malu karticu profila. Ako je otvorena izložba koju je sam korisnik kreirao, biti će ponuđena i opcija uređivanja te brisanja exhibita. Pri dodiru ikone za brisanje (ikona kante za smeće) prikazat će se Alert prozor, koji traži da se potvrdi brisanje. Pri dodiru ikone za uređivanje (ikona olovke), otvara se forma za dodavanje novog exhibita (kao na slici 12), no tada je forma popunjena inicijalnim podacima te izložbe. Nakon uređivanja informacija, moguće je opet spremiti tu izložbu.



Slika 22: Lijevo: ArtGallery prikaz detalja izložbe, desno: ArtGallery "sheet" sa komentarima

Također će dugme „Post exhibit“ tada biti „Update exhibit“, čija logika se nalazi na slici 23. Ako je proslijeđena izložba u widget, tada se smatra da se izložba uređuje, no ako nije, tada se smatra da se izrađuje nova izložba.

```
ElevatedButton(
  onPressed: () {
    _postExhibit(context, currentUser!);
  },
  child: widget.exhibit == null
      ? Text('Post exhibit')
      : Text('Update exhibit'),
), // ElevatedButton
```

Slika 23: ArtGallery logika teksta gumba

Slike izložbe u prikazu detalja izložbe je moguće je pregledavati tako što se dodirne i pomakne prstom u lijevo ili desno. Slike se također automatski listaju, što omogućava Carousell Slider [15]. Njegova implementacija vidi se na slici 24.

```
child: CarouselSlider(
  items: generateImages(widget.exhibit.imageList),
  options: CarouselOptions(
    height: 250,
    aspectRatio: 16 / 9,
    initialPage: 0,
    enableInfiniteScroll: true,
    reverse: false,
    autoPlay: true,
    autoPlayInterval: Duration(seconds: 3),
    autoPlayAnimationDuration:
        Duration(milliseconds: 800),
    autoPlayCurve: Curves.fastOutSlowIn,
    enlargeCenterPage: true,
    scrollDirection: Axis.horizontal,
  ), // CarouselOptions
), // CarouselSlider
```

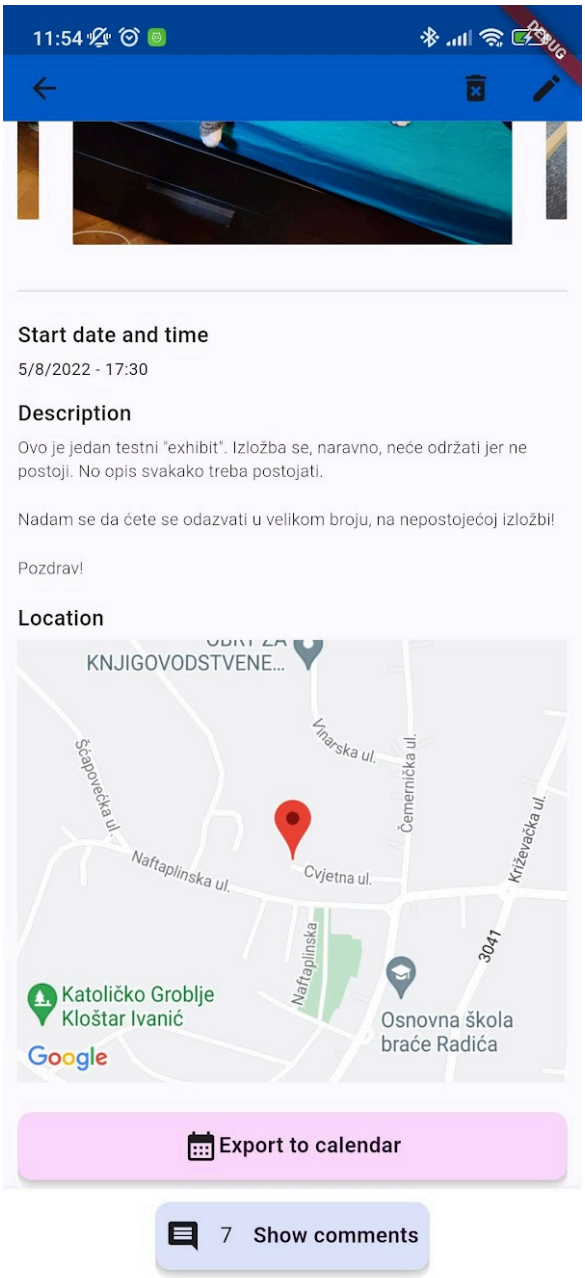
Slika 24: ArtGallery CarouselSlider implementacija

Ako je brisanje potvrđeno, poziva se funkcija na slici 25. Funkcija će primiti ID korisnika kako bi se autoriziralo brisanje izložbe, ID izložbe, te listu slika koje treba obrisati, s obzirom da Storage i Firestore nisu povezani. Nakon što se obrišu sve slike iz liste, briše se i sama izložba. Time će se osigurati da ne preostaju slike izložbi koje su obrisane, i održavati prostor za pohranu čistim.

```
@override
Future<void> deleteExhibit({
  required String userId,
  required String exhibitId,
  required List<String> imagesToDelete,
}) async {
  try {
    imagesToDelete.forEach(
        (element) => FirebaseStorage.instance.refFromURL(element).delete());
    await _read(firebaseFirstoreProvider)
        .collection('exhibits')
        .doc(exhibitId)
        .delete();
  } on FirebaseException catch (e) {
    throw CustomException(message: e.message);
  }
}
```

Slika 25: ArtGallery funkcija za brisanje izložbe

Na dnu ekrana se nalazi i interaktivna karta sa lokacijom izložbe, te je moguće dodirnuti pribadaču na karti te zatražiti navigaciju do lokacije, što je prikazano na slici 26. Ispod karte, nalazi se „Export to calendar“ dugme koje će generirati event na defaultnom kalendaru uređaja, te ga je moguće pohraniti. Tako možemo biti adekvatno obaviješteni prije početka izložbe.



Slika 26: ArtGallery adresa izložbe i dugme za export u kalendar

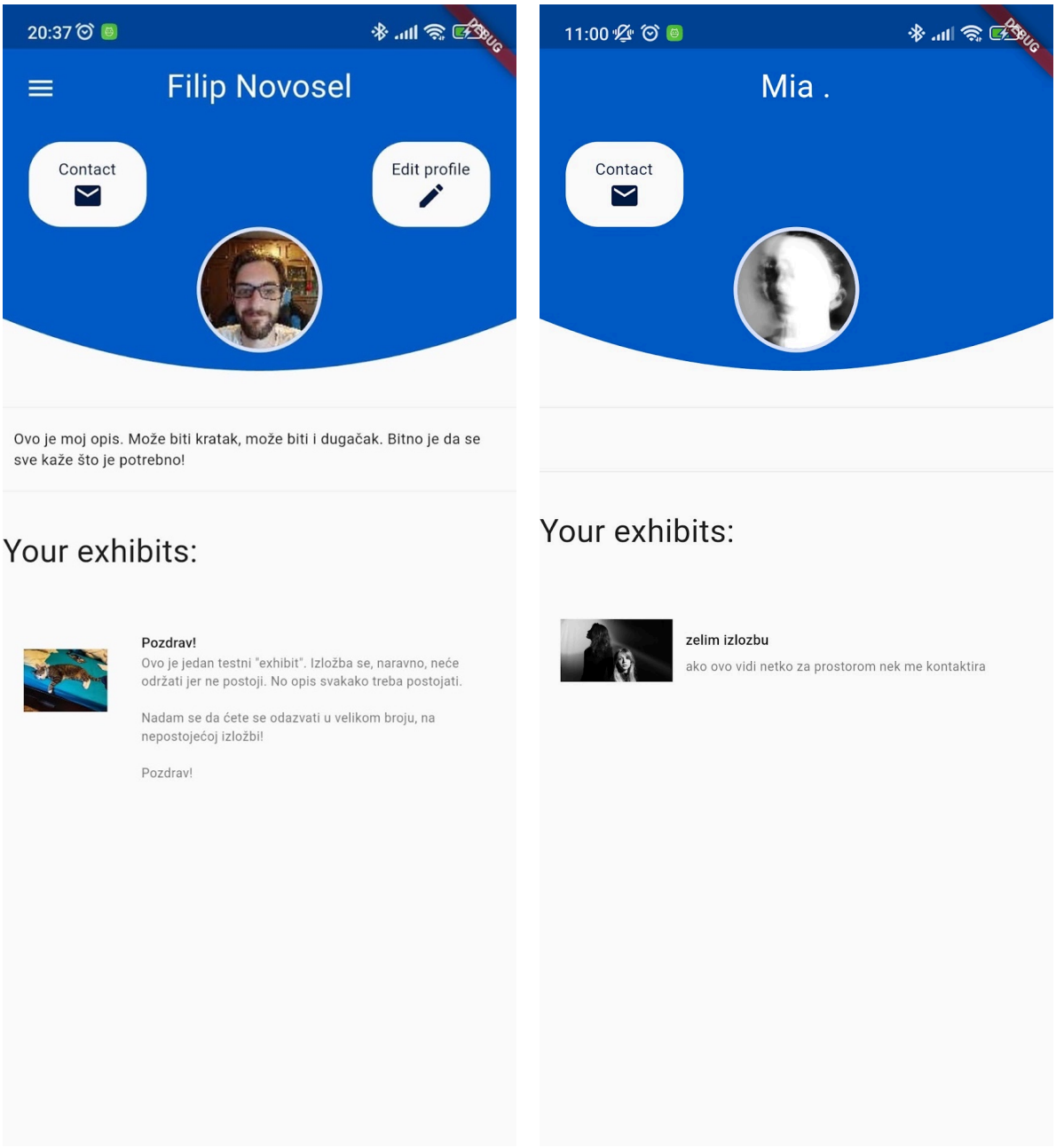
Pri dodiru na „Export to calendar“ dugme, poziva se Add to Calendar [16 i 17] funkcija koja generira iCal event te ga prosljeđuje u zadanu aplikaciju kalendara telefona. Sama implementacija nalazi se na slici 27. Prvo je potrebno definirati objekt Event, kojeg se potom može proslijediti u funkciju *addEvent2Cal.* Ta će funkcija obaviti poziv izvan Flutter-a, te proslijediti hipervezu, koju će uređaj prepoznati kao event za upisivanje novog događaja u kalendar.

```
void exportExhibitToCalendar({
  required Exhibit exhibit,
}) async {
  try {
    final Event event = Event(
      title: exhibit.title,
      description: exhibit.description,
      location:
          "${exhibit.location.street}, ${exhibit.location.city}, ${exhibit.location.country}",
      startDate: exhibit.startDateTime,
      endDate: exhibit.endDate,
    );

    Add2Calendar.addEvent2Cal(event);
  } catch (e) {}
}
```

Slika 27: ArtGallery implementacija Add2Calendar

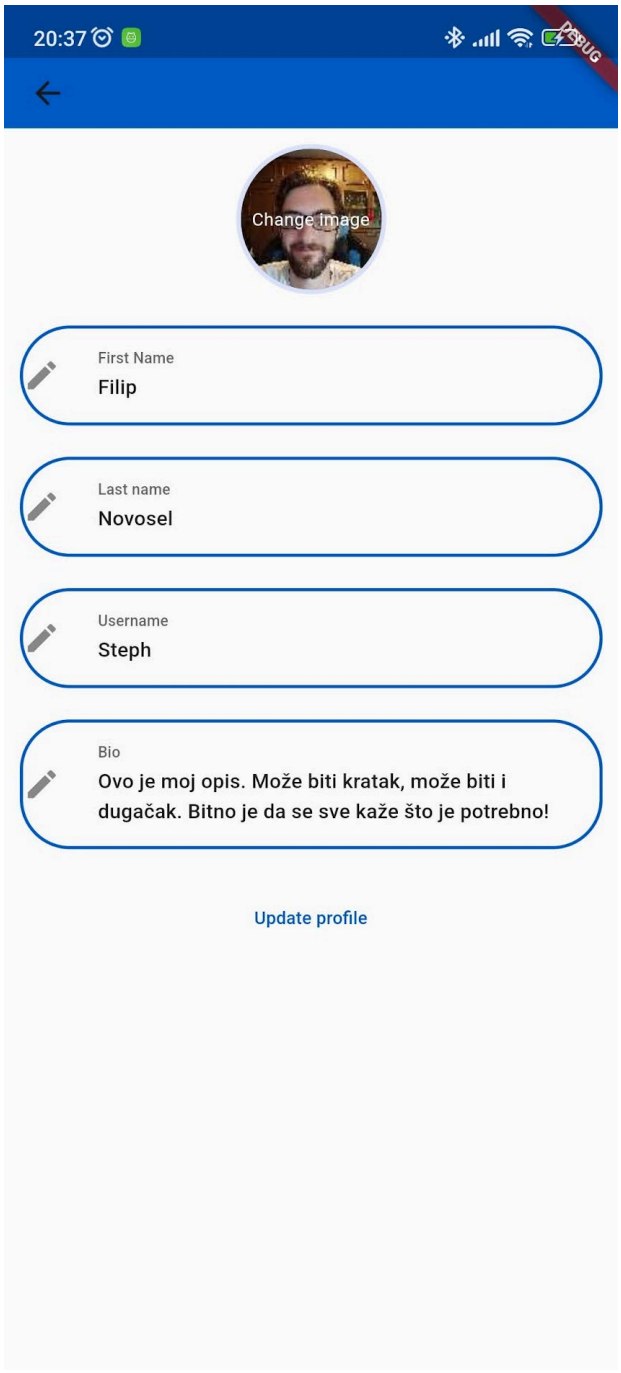
3.5. Profil korisnika

Svaki korisnik može pogledati svoj, te tuđi profil. Ako korisnik gleda svoj profil, također je vidljivo i dugme „Update profile“ sa kojim možemo urediti svoj profil. Na slici 28 vidljivi su ekrani vlastitog profila (lijevo) te profila drugih korisnika (desno).



Slika 28: Lijevo: ArtGallery prikaz vlastitog profila, desno: ArtGallery prikaz tuđeg profila

Ako dodirnemo „Edit Profile“ dugme, tada će nas aplikacija odvesti na ekran sa formom gdje možemo urediti informacije o svom profilu. Forma će biti popunjena sa podacima koji su spremljeni u bazi podataka. Forma za uređivanje profila nalazi se na slici 29. Ako dodirnemo dugme „Contact“ poziva se URL launcher [15] koji ima namjenu otvaranja hiperveze kroz operativni sustav, koji tada čita protokol te otvara prigodnu aplikaciju.



Slika 29: ArtGallery forma za uređivanje profila

Nakon što smo zadovoljni promjenama, možemo ih pohraniti dodirom na dugme „Update profile“. Time će se pozvati funkcija sa slike 30, koja će prvo dohvatiti objekt korisnika, potom pohraniti novu sliku profila (ako ju je korisnik odabrao za promjenu),

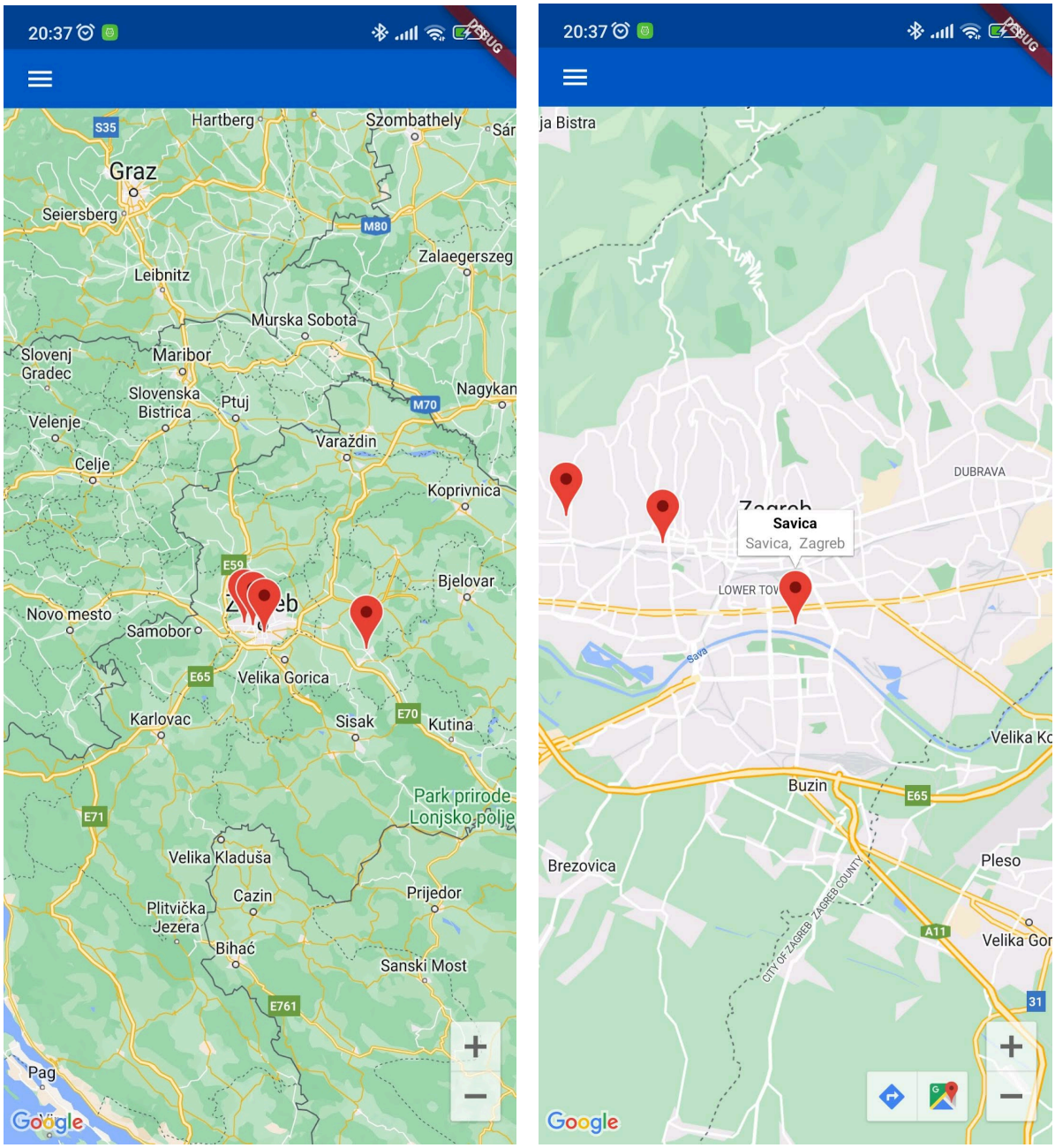
spremiti unesene podatke, te kada se uspješno sve pohrani, zatvoriti će se ekran sa formom za uređivanje profila.

```
TextButton(
  onPressed: () async {
    var currentAuthUser = ref.read(authControllerProvider);
    if (newImage != null) {
      /* await FirebaseStorage.instance
          .refFromURL(_userImageUrl)
          .delete(); */
      final ref = FirebaseStorage.instance
          .ref()
          .child('user_image')
          .child(currentAuthUser!.uid + '.jpg');
      await ref.delete();
      await ref.putFile(File(newImage!.path));
      _userImageUrl = await ref.getDownloadURL();
    }

    await ref
        .read(userControllerProvider.notifier)
        .updateUserData(
          currentAuthUser!.uid,
          UserModel(
            email: _email,
            image_url: _userImageUrl,
            username: _username,
            firstName: _firstName,
            lastName: _lastName,
            reviewable: _reviewable,
            bio: _bio,
            favorites: _favorites,
            createdAt: _createdAt,
          ),
        );
    Navigator.pop(context);
  },
  child: Text('Update profile'),
),
```

Slika 30: ArtGallery funkcija za spremanje promjena uređenog profila korisnika

3.6. „Explore“

Na slici 31 (lijevo) možemo vidjeti Explore ekran sa interaktivnom kartom na kojoj se nalaze pribadače sa lokacijama izložbi koje su unijeli korisnici. To omogućuje *Google Maps Flutter* [19] ekstenzija. Dodirom na pribadaču, javlja se mali skočni prozor sa detaljima (naslov izložbe i adresa), koji je opet moguće dodirnuti te ćemo biti odvedeni na prikaz detalja izložbe, te je prikazan na slici 31 (desno).



Slika 31: Lijevo: ArtGallery Explore ekran, desno: ArtGallery skočni prozor sa detaljima

4. Zaključak

Cilj ovog rada bio je prikazati implementaciju jedne mobilne aplikacije za umjetnike. Sama izrada bila je odviše zanimljiva radi kombinacije tehnologija koje su korištene, no evidentno je da ne pokriva sva potrebna područja potrebna umjetnicima. Ovaj je rad samo zagrebao površinu, a to je oglašavanje samih izložbi, njihovo pregledavanje, društvenu interakciju uz komentare, generiranje obavijest te pregled i navigacija do mjesta izložbe.

Aplikacija je prošla kroz omanji broj mobilnih telefona ljudi različitih zanimanja, starosti i lokacija, te je sama ideja bila više nego prihvaćena. Velika kritika korisnika je nemogućnost odabira više fotografija iz galerije pri izradi izložbe, što otvara vrata za poboljšanje. Javlja se i problem kod objave komentara, gdje komentar nestane iz korisničkog sučelja pri zatvaranju tipkovnice, te nedostatak vizualnog indikatora osvježavanja izložbi u Dashboard-u. Iz samoga rada proizlazi potencijal za napretkom, možda i izrada kompletne centralne jedinice koja neće omogućiti samo oglašavanje izložbi, već i prijave na natječaje za izložbene prostore, odjeljak za ponudu/potražnju poslova (od ukrašavanja i crtanja zidova spavaće sobe djeteta, pa do izrade skulpture), što samo po sebi već olakšava izradu portfolija umjetnika i olakšava njegov napredak i rast.

Velika prednost je brzina prijenosa konkretnih i specifičnih informacija između umjetnika, onih koji to tek žele postati, te ljudi koji su zainteresirani za umjetnost. Uz dodavanje događanja na kalendar mobilnog telefona, te podešavanja podsjetnika, teško će se zaboraviti na izložbu.

Kako je tehnologija promijenila sve naše živote, od najmanjih pa tako do najvećih stvari; oglašavanje poslova, pregled interaktivnog sadržaja, zabava, prodaja, pregled recepata. Ako jedan informatičar, dizajner ili programer može iz udobnosti svoga doma pronaći, odraditi i naplatiti svoj posao, ne postoji razlog zašto i umjetnici ne bi mogli uživati u istim blagodatima Interneta.

Uz vrlo brzi tehnološki napredak, krajnje je vrijeme da se i ostale djelatnosti (makar nekima bile i samo hobi) umreže u jednu zajednicu za koju znaju da ih neće sputavati algoritmima i time inducirati osjećaj manje vrijednosti.

Literatura

1. *Firebase Core* (n.d.). Firebase.google.com. verzija 1.4.0, Preuzeto : 5.9.2022.: https://pub.dev/packages/firebase_core
2. *Firebase Auth* (n.d.). Firebase.google.com. verzija 3.0.1, Preuzeto : 5.9.2022.: https://pub.dev/packages/firebase_auth
3. *Firebase Storage* (n.d.). Firebase.google.com. verzija 10.0.3, Preuzeto : 5.9.2022.: https://pub.dev/packages/firebase_storage
4. *Cloud Firestore* (n.d.). Firebase.google.com. verzija 3.1.1, Preuzeto : 5.9.2022.: https://pub.dev/packages/cloud_firestore
5. *Freezed* (n.d.). Dash-overflow.net. verzija 2.0.3+1, Preuzeto : 5.9.2022.: https://pub.dev/packages/freezed
6. *Copy with extension gen* (n.d.). oleksandrkirichenko.com. verzija 4.1.0, Preuzeto : 5.9.2022.: https://pub.dev/packages/copy_with_extension_gen
7. *JSON Serializable* (n.d.). Google.dev. verzija 6.1.5, Preuzeto : 5.9.2022.: https://pub.dev/packages/json_serializable
8. *JSON Annotation* (n.d.). Google.dev. verzija 4.4.0, Preuzeto : 5.9.2022.: https://pub.dev/packages/json_annotation
9. *Flutter Riverpod* (n.d.). Dash-overflow.net. verzija 1.0.4, Preuzeto : 5.9.2022.: https://pub.dev/packages/flutter_riverpod
10. *Flutter Google Places* (n.d.). Unvertified uploader. verzija 0.3.0, Preuzeto : 5.9.2022.: https://pub.dev/packages/flutter_google_places
11. *Flutter Dotenv* (n.d.). Unverified uploader. verzija 5.0.2, Preuzeto : 5.9.2022.: https://pub.dev/packages/flutter_dotenv
12. *Date Time Picker* (n.d.). M3uzz.com. verzija 2.1.0, Preuzeto : 5.9.2022.: https://pub.dev/packages/date_time_picker
13. *Image Picker* (n.d.). Flutter.dev. verzija 0.8.5+3, preuzeto: 5.9.2022. sa: https://pub.dev/packages/image_picker
14. *Flutter Advanced Drawer* (n.d.). Alexmelnyk.io. verzija 1.3.0, preuzeto 5.9.2022. sa: https://pub.dev/packages/flutter_advanced_drawer
15. *Carousel Slider* (n.d.). Serenader.me. verzija 4.1.1, preuzeto 5.9.2022. sa: https://pub.dev/packages/carousel_slider

16. *Add 2 Calendar* (n.d.). unverified uploader. verzija 2.1.3, preuzeto 5.9.2022. sa: https://pub.dev/packages/add_2_calendar
17. *Device Calendar* (n.d.). builttoroam.com. verzija 4.2.0, preuzeto 5.9.2022. sa: https://pub.dev/packages/device_calendar
18. *URL Launcher* (n.d.). Flutter.dev. verzija 6.1.0, preuzeto 5.9.2022. sa: https://pub.dev/packages/url_launcher
19. *Google Maps Flutter*. (n.d.). Flutter.dev. verzija 2.1.8, preuzeto 5.9.2022. sa: https://pub.dev/packages/google_maps_flutter
20. *Podržane platforme*. (n.d.). Flutter. Preuzeto 29.8.2022. sa https://docs.flutter.dev/development/tools/sdk/release-notes/supported-platforms
21. *Material Design 3*. (n.d.). Google. Preuzeto 29.8.2022. sa https://m3.material.io/
22. *Material You*. (n.d.). Google. Preuzeto 29.8.2022. sa https://material.io/blog/announcing-material-you

Popis slika