Izrada višeplatformskih mobilnih aplikacija - flutter / dart

Marinović, Hrvoje

VELEUČILIŠTE
s pravom javnosti
BALTAZAR ZAPREŠIĆ
Zaprešić

Preddiplomski stručni studij
Informacijske tehnologije

HRVOJE MARINOVIĆ

IZRADA VIŠEPLATFORMSKIH MOBILNIH APLIKACIJA - FLUTTER / DART

PREDDIPLOMSKI ZAVRŠNI RAD

Zaprešić, 2023. godine

VELEUČILIŠTE
s pravom javnosti
BALTAZAR ZAPREŠIĆ
Zaprešić

Preddiplomski stručni studij
Informacijske tehnologije

PREDDIPLOMSKI ZAVRŠNI RAD

IZRADA VIŠEPLATFORMSKIH MOBILNIH APLIKACIJA - FLUTTER / DART

Mentor:
dr. sc. Vladimir Mateljan, red. prof.
Naziv kolegija:
Mobilne aplikacije

Student:
Hrvoje Marinović
JMBAG studenta:
0234061205

SADRŽAJ

SAŽETAK

Cilj ovog diplomskog rada je izrada višeplatformskih mobilnih aplikacija koristeći Flutter i Dart. U uvodu su objašnjeni pojam višeplatformskih mobilnih aplikacija te su istaknute njihove prednosti. U nastavku su opisane karakteristike Fluttera i Darta, te njihova upotreba u izradi mobilnih aplikacija. U središnjem dijelu rada prikazan je postupak izrade višeplatformskih aplikacije na različitim platformama. U četvrtom poglavlju je predstavljen primjer uspješno realizirane višeplatformske mobilne aplikacije. U zaključku je istaknuta važnost višeplatformskih mobilnih aplikacija, te su prednosti korištenja Fluttera i Darta u njihovoj izradi.

Ključne riječi: Višeplatformske mobilne aplikacije, Flutter, Dart, Mobilna aplikacija

DEVELOPMENT OF MULTI-PLATFORM MOBILE APPLICATIONS - FLUTTER / DART

ABSTRACT

This thesis aims to develop cross-platform mobile applications using Flutter and Dart. The introduction explains the concept of cross-platform mobile applications and highlights their advantages. The characteristics of Flutter and Dart are described, along with their use in mobile application development. The central part of the thesis presents the process of creating a cross-platform mobile application, from setting up the development environment to implementing functionality and testing the application on different platforms. The fourth chapter provides example of successfully developed cross-platform mobile application. In conclusion, the importance of cross-platform mobile applications and the advantages of using Flutter and Dart in their development are emphasised.

Key words: Cross-platform mobile applications, Flutter, Dart, Mobile application.

1. UVOD

U današnje vrijeme mobilne aplikacije igraju ključnu ulogu u našim svakodnevnim životima. Njihova popularnost neprestano raste, a kako bi iskoristili sve svoje prednosti koje nam ove aplikacije pružaju, razvoj mobilnih aplikacija postaje sve složeniji i zahtjevniji. Jedan od glavnih izazova s kojima se programeri suočavaju je razvijanje mobilnih aplikacija za različite platforme, kao što su iOS i Android. Zbog toga je koncept višeplatformskog razvoja mobilnih aplikacija ključan.

Jedan od najnovijih i najpopularnijih alata za razvoj višeplatformskih mobilnih aplikacija je Flutter. Flutter je open-source alat koji omogućuje razvoj mobilnih aplikacija za različite platforme uz korištenje jednog zajedničkog koda. Za razvoj u Flutter-u koristi se programski jezik Dart, koji je razvijen s ciljem rješavanja problema u razvoju mobilnih aplikacija.

Fokus ovog rada je razvoj višeplatformskih mobilnih aplikacija korištenjem Flutter-a i Dart-a. Cilj rada je pružiti detaljan pregled procesa razvoja višeplatformskih mobilnih aplikacija uz upotrebu Flutter-a i Dart-a. Također bit će prikazan primjer uspješno razvijene mobilne aplikacije korištenjem ove tehnologije.

Kao što je jednom rekao Eric Sevareid*, "The chief cause of problems is solutions"*, što u prijevodu znači *"Glavni uzrok problema su rješenja" (Eric Savareid, na vijestima CBS-a, 29. prosinca 1970).*

Stoga je važno pronaći pravi alat za razvoj mobilnih aplikacija kako bi se riješili problemi koji se mogu pojaviti. Upravo je to ono što Flutter i Dart nude – alat za razvoj mobilnih aplikacija koji je jednostavan za korištenje, a pruža fleksibilnost u razvoju mobilnih aplikacija za različite platforme.

2. FLUTTER I DART

2.1 Flutter kao Framework za razvoj višeplatformskih mobilnih aplikacija

Flutter je otvoreni izvorni kod i korisničko sučelje (UI) toolkit koji se koristi za razvoj mobilnih aplikacija, desktop aplikacija i web aplikacija. Flutter je razvije od strane Google-a te je prvotno objavljen 2017. godine. Flutter je popularan među programerima jer omogućuje jednostavan i brz razvoj aplikacija za više platformi (iOS, Android, Web, Desktop) s istim izvornim kodom.

Flutter koristi programski jezik Dart, koji je također razvijen od strane Goolgle-a. Dart je prvobitno razvijen za korištenje u web aplikacijama, no s dolaskom Fluttera, postao je važan programski jezik za razvoj mobilnih aplikacija.

Jedna od glavnih prednosti Fluttera je što omogućuje brz razvoj aplikacija bez gubitka performansi ili kvalitete aplikacije. To se postiže korištenjem vlastitog grafičkog sustava tzv. Flutter Engine, koji omogućuje prikazivanje aplikacije s visokom brzinom osvježavanja. Osim toga, Flutter ima bogatu kolekciju widgeta koji se koriste za izgradnju korisničkog sučelja.

Još jedna značajna mogućnost koju Flutter pruža je Hot Reload, što programerima omogućuje da brzo vide promjene koje su napravili u aplikaciji. To znači da se izmjene u izvornom kodu odmah prikazuju u aplikaciji, što je vrlo korisno za testiranje i razvoj.

Flutter također ima aktivnu i veliku zajednicu programera te podršku tvrtke Goolge. Što znači da programeri mogu pronaći rješenja za probleme s kojima se su, kao i pristupiti raznim resursima za učenje i razvoj.

Uz sve navedeno, Flutter je također besplatan i otvorenog koda, što ga čini privlačnim za male i velike tvrtke koje žele razviti mobilnu aplikaciju na brz i učinkovit način.

2.2 Dart kao programski jezik koji se koristi za pisanje koda u Flutteru

Dart je programski jezik koji je razvijen od strane Google-a i koji se koristi za pisanje koda u Flutteru. To je moderni, objektno orijentirani jezik, koji je prvenstveno dizajniran za izgradnju velikih i kompleksnih aplikacija. Dart je vrlo sličan programskim jezicima poput Jave, JavaScript-a i C++.

Dart ima mnoge karakteristike koje ga čine idealnim za izradu mobilnih aplikacija. Ovaj programski jezik ima jednostavnu sintaksu, brzo izvođenje koda, veliku podršku za asinkrone operacije, te također omogućava razvoj aplikacija koje rade na iOS-u i Androidu.

Također, Dart ima veliku podršku za izradu i korištenje widgeta, što čini integraciju s Flutterom vrlo jednostavnom i intuitivnom.

Dart se izvršava na Dart virtualnom stroju (DartVM), koji omogućava brzo izvođenje koda i nisku latenciju. Osim toga, Dart omogućava pretvorbu koda koji u Javascript, što omogućava izradu web aplikacija pomoću Dart-a.

Prednosti programskog jezika Dart:

- Jednostavna sintaksa
- Brzo izvršavanje koda
- Velika podrška za widgete i UI
- Izvrsna podrška za asinkrone operacije
- Statički tipovi podataka
- Velika podrška za objektno orijentirano programiranje

Ukratko, Dart je brzi, moderni, objektno orijentirani programski jezik koji se koristi za pisanje koda u Flutter-u. Dart ima mnoge prednosti koje ga čine idealnim za izradu mobilnih aplikacija i veliku podršku za widgete i UI.

2.3 Osnovne značajke Fluttera i Darta

Flutter:

- Flutter je otvoreni izvorni kod framework za izradu mobilnih aplikacija, web aplikacija i desktop aplikacija.
- Flutter je razvijen od strane Google-a i prvi put je predstavljen na Dart developer summitu u 2017. godini.
- Flutter se temelji na konceptu widgeta koji se koriste za izgradnju korisničkog sučelja aplikacije. Ovo omogućava razvoj visoko prilagodljivih aplikacija koje se mogu koristiti na različitim platformama.
- Flutter podržava Hot reload, što znači da se promjene u kodu mogu odmah vidjeti u aplikaciji.
- Flutter ima veliku i aktivnu zajednicu programera koji su spremni pomoći i podržati novajlije u razvoju aplikacija.

- Flutter je jedan od najbrže rastućih alata za razvoj mobilnih aplikacija na tržištu.

Dart:

- Dart je programski jezik koji se koristi za razvoj aplikacija koje se temelji na Flutter-u.
- Dart je razvijen od strane Google-a i prvi put predstavljen u 2011. godini.
- Dart je jednostavan za učenje i koristi se u različitim projektima, od izrade web aplikacija do server-side programiranja.
- Dart je statički tipiziran jezik što omogućava programerima da otkriju greške u kodu prije pokretanja aplikacije.
- Dart ima ugrađenu podršku za asinkrone operacije što je korisno u razvoju aplikacije koje zahtijevaju obradu velikih količina podataka.
- Dart također podržava JIT (Just-In-Time) i AOT (Ahead-of-Time) kompilaciju što pomaže u brzom razvoju i optimalizaciji performansi aplikacija.

Uz ove osnovne značajke, Flutter i Dart nude i druge napredne mogućnosti koje se mogu koristiti u razvoju mobilnih aplikacija. Stoga, ovi alati predstavljaju vrlo dobru opciju za razvoj modernih i prilagodljivih mobilnih aplikacija.

3. KREIRANJE VIŠEPLATFORMSKE MOBILNE APLIKACIJE KORISTEĆI FLUTTER I DART

3.1 Postavljanje razvojnog okruženja za Flutter i Dart

Prije početka izrade višeplatformske mobilne aplikacije koristeći Flutter i Dart, potrebno je postaviti razvojno okruženje koje će omogućiti razvoj, testiranje i izgradnju aplikacije.

Za rad s Flutter-om i Dart-om potrebno je instalirati slijedeće komponente:

- Flutter SDK – sadrži alate za razvoj, testiranje i izgradnju aplikacije.
- Visual Strudio Code ili Android Studio – razvojno okruženje za izradu Flutter aplikacija
- Dart plugin – plugin koji omogućuje rad s Dartom u odabranom razvojnom okruženju

Flutter SDK se može preuzeti s službene stranice Fluttera, a nakon preuzimanja i instalacije potrebno je dodati putanju do Flutter alata u PATH varijablu okruženja.

Za razvoj Flutter aplikacija preporuča se korištenje Android Studija ili Visual Studio Code-a. Nakon instalacije odabranog razvojnog okruženja, potrebno je instalirati Dart plugin.

Nakon uspješne instalacije potrebnih komponenti, može se započeti sa izradom višeplatformske mobilne aplikacije.

3.2 Kreiranje osnovne strukture višeplatformske mobilne aplikacije

Samo kreiranje osnovne strukture višeplatformske mobilne aplikacije uključuje postavljanje projekta i organizaciju datoteka u Flutter okruženju.

Prije samog početaka izrade aplikacije, potrebno je odabrati ime projekta i stvoriti novi projekt u odabranoj IDE (Integrated Development Environment) aplikaciji. Zatim, potrebno je odabrati vrstu aplikacije koja će se kreirati primjerice za iOS, Android ili oboje platforme.

Nakon što je projekt kreiran, slijedi organizacija datoteka i direktorija unutar projekta. Flutter projekt sastoji se od nekoliko ključnih datoteka i direktorija, među kojima su 'lib/' direktorij, 'pubspec.yaml' datoteka i 'main.dart' datoteka,

'lib/' direktorij sadrži izvorni kod aplikacije, koji se sastoji od Dart datoteka koje definiraju sami izgled i funkcionalnost aplikacije. U 'pubspec.yaml' datoteci, definiraju se vanjske biblioteke i paketi potrebni za izgradnju aplikacije.

Glavna Dart datoteka, 'main.dart', definira strukturu i sadržaj aplikacije, uključujući glavni widget aplikacije, kao i funkcije koje opisuju ponašanje i logiku aplikacije.

Prilikom kretanja osnovne strukture aplikacije, važno je razmotriti i organizaciju koda, kako bi se olakšala buduća održivost i proširivost aplikacije. Preporučljivo je organizirati kod u odvojene module i widgete, te koristiti dobro poznate Flutter arhitekture poput BLoC (Business Logic Component), Provider, i druge, kako bi se osigurala čitljivost i skalabilnost aplikacije.

3.3 Korištenje različitih Flutter widgeta za izradu sučelja aplikacije

Korištenje različitih Flutter widgeta za izradu sučelja aplikacije predstavlja jednu od osnovnih karakteristika Fluttera kao frameworka za razvoj višeplatformskih mobilnih aplikacija. Flutter dolazi sa širokim izborom predefiniranih widgeta koji se mogu koristiti za izradu grafičkog sučelja aplikacije. Uz navedeno, Flutter također omogućava kreiranje potpuno prilagođenih widgeta, tako da je izrada grafičkog sučelja u potpunosti fleksibilna i prilagodljiva potrebama aplikacije.

Najčešće korišteni Flutter widgeti uključuju:

- Text wiget – koristi se za prikazivanje jednostavnog teksta
- Container widget – omogućava grupiranje drugih widgeta i definiranje njihovog izgleda (veličina, rubovi, pozadinska boja itd.)
- Column i Row widgeti – koriste se za organiziranje drugih widgeta u vertikalne i horizontalne redove
- Image widget – koristi se za prikazivanje slika
- Button widgeti – primjerice ElevatedButton i FlatButton, koji se koriste za kreiranje gumba i njegove funkcionalnosti
- Input widgeti – primjerice TextField i DropdownButton, koji se koriste za prikupljanje korisničkog unosa.
- ListView widget – koristi se za prikazivanje popisa stavki

Osim navedenih, postoji i mnogo drugih widgeta koje Flutter nudi, poput Animation widgeta koji se koristi za animiranje korisničkog sučelja, ili primjerice Table widgeta kako bi se organizirale widgeti u tablice.

Važno je napomenuti da se Flutter widgeti mogu prilagođavati kroz različite parametre, što omogućava još veću fleksibilnost u izradi grafičkog sučelja. Primjerice, Text widget se može

prilagoditi veličini, boji i fontu te još mnogim drugim svojstvima, dok se Container widget može prilagoditi dimenzijama, pozadinskoj boji, marginama te još drugim svojstvima.

Korištenje različitih Flutter widgeta za izradu sučelja aplikacije olakšava i ubrzava proces razvoja, omogućava veću fleksibilnost i prilagodljivost te rezultira modernim i atraktivnim grafičkim sučeljem za višeplatformsku mobilnu aplikaciju.

3.4 Implementacija funkcionalnosti aplikacije koristeći Dart programski jezik

U ovoj sekciji fokusirati ćemo se na implementaciju funkcionalnosti aplikacije korištenjem Dart programskog jezika u Flutter okruženju.

Dart je objektno orijentirani programski jezik koji se koristi za pisanje koda u Flutter okruženju. Dart je dizajniran kako bi bio jednostavan za učenje, učinkovit i brz. Jedna od glavnih karakteristika Darta je to da je to kompilirani jezik, što znači da se programski kod prevodi u strojni jezik prije izvođenja, što pridonosi performansama.

Implementacija funkcionalnosti aplikacije započinje definiranjem klasa koje opisuju objekte i njihova svojstva. Na primjer, klasa koja opisuje korisnika aplikacije mogla bi sadržavati svojstva kao što su ime, prezime, e-mail adresa, itd. Definiranje klasa u Dart-u vrlo je jednostavno te je slično definiranju klasa u drugim programskim jezicima.

Jedna od ključnih značajki Dart programskog jezika je tzv. 'async/await' sintaksa koja omogućuje asinkrono izvršavanje koda, što je važno za mobilne aplikacije koje moraju interaktirati s različitim izvorima podataka, kao što su mrežni API-ji ili lokalna pohrana podataka.

Flutter dolazi s mnogim ugrađenim widgetima koji olakšavaju implementaciju funkcionalnosti aplikacije. Na primjer, FlatButton widget se može koristiti za definiranje gumba sučelja aplikacije, a TextFromField widget za unos teksta. Flutter također nudi podršku za različite tipove unosa, poput izbornika, potvrde, lozinka s višestrukim izborima i drugih.

Nakon definiranja funkcionalnosti aplikacije, možemo koristiti Dart za obradu podataka koji se unose na sučelju. To može uključivati provjeru valjanosti unesenih podataka, spremanje podataka u bazu podatak ili slanje podataka putem mreže. Dart također omogućuje interakciju s drugim servisima i API-ima putem HTTP zahtjeva.

Konačno, Dart i Flutter zajedno omogućuju implementaciju svih funkcionalnosti mobilne aplikacije, uključujući sučelje, logiku i interakciju s podacima. Zahvaljujući bogatstvu dostupnih widgeta i alata, izrada mobilne aplikacije u Flutteru i Dartu može biti brza i učinkovita.

3.5 Testiranje aplikacije na različitih platformama

Kada je aplikacija završena, potrebno ju je testirati na različitim platformama kako bi se osigurala njena funkcionalnost i kvaliteta na svakoj platformi. Flutter omogućuje testiranje na tri glavne platforme a to su: Android, iOS i Web.

Kako bi se testiralo na Android i iOS platformama, potrebno je koristiti odgovarajuće emulatore ili uređaj. Flutter dolazi sa integriranim emulatorima za ove dvije platforme, čime omogućuje jednostavno testiranje aplikacije. Potrebno je izvršiti nekoliko osnovnih testova kako bi se provjerilo funkcioniranje aplikacije, poput provjere može li se aplikacija pokrenuti i da li se funkcionalnosti aplikacije izvršavaju bez problema.

Za testiranje na Webu, potrebno je objaviti aplikaciju na nekom od web poslužitelja. Postoji više mogućnosti za objavljivanje aplikacije, uključujući Firebase i GitHub Pages. Nakon što se objavimo aplikaciju, potrebno ju je testirati na različitim preglednicima kako bi se osigurala njena funkcionalnost i kvaliteta.

Uz osnovne testove, preporučljivo je provesti i nekoliko testova koji se fokusiraju na performanse aplikacije. To uključuje testiranje brzine učitavanja aplikacije, vrijeme odaziva te ukupno vrijeme potrebno za izvršavanje različitih funkcionalnosti aplikacije. Ovi testovi pomažu u osiguravanju da će aplikacija raditi glatko i brzo na svakoj platformi na kojoj se izvršava.

U zaključku, testiranje je ključni korak u razvoju bilo koje aplikacije, posebno višeplatformskih mobilnih aplikacija. Testiranje na različitim platformama osigurava da će aplikacija pravilno funkcionirati na svakoj platformi i da će korisnici dobiti visokokvalitetno iskustvo korištenja aplikacije.

4. PRIMJER VIŠEPLATFORMSKE MOBILNE APLIKACIJE KREIRANE KORIŠTENJEM FLUTTERA I DARTA

4.1 Primjer : Aplikacija za fotogaleriju

Za izradu aplikacije za fotogaleriju važne su četiri glavne komponente:

- Izrada početne stranice
- Stvaranje organizacijske značajke
- Usmjeravanje na indeks slike
- Izrada navigacijske trake unutar ruta

4.1.1 Izrada početne stranice

Prije nego što započnemo s izradom početne stranice, potrebno je provesti nekoliko koraka u stvaranju novog Flutter projekta. Slijedite sljedeće korake:

1. Otvorite terminal i stvorite novi direktorij za projekt pomoću naredbe: **mkdir ime_mape** (gdje ime_mape zamijenite željenim nazivom mape).
2. Promijenite trenutni direktorij u novo stvoreni direktorij pomoću naredbe: **cd ime_mape**.
3. Izradite novi Flutter projekt pomoću naredbe: **flutter create ime_projekta** (gdje ime_projekta zamijenite željenim nazivom projekta).

Nakon što ste stvorili novi projekt, otvorite **main.dart** datoteku i zamijenite postojeći kod sljedećim kodom:

```
import 'package:flutter/material.dart';
import 'homepage.dart';

void main() {
  runApp(const MyApp());
}
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  static const String _title = 'Gallery App';
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) => const MaterialApp(
```

```
        title: _title,
        debugShowCheckedModeBanner: false,
        home: MyHomePage(title: 'Gallery'),
      );
}
```

Gornji kod jednostavno instancira klasu **MyHomePage**. Sada trebamo stvoriti novu klasu **MyHomePage** koju ćemo smjestiti u zasebnu Dart datoteku nazvanu **homepage.dart**. Otvorite **homepage.dart** datoteku i zamijenite postojeći kod sljedećim kodom:

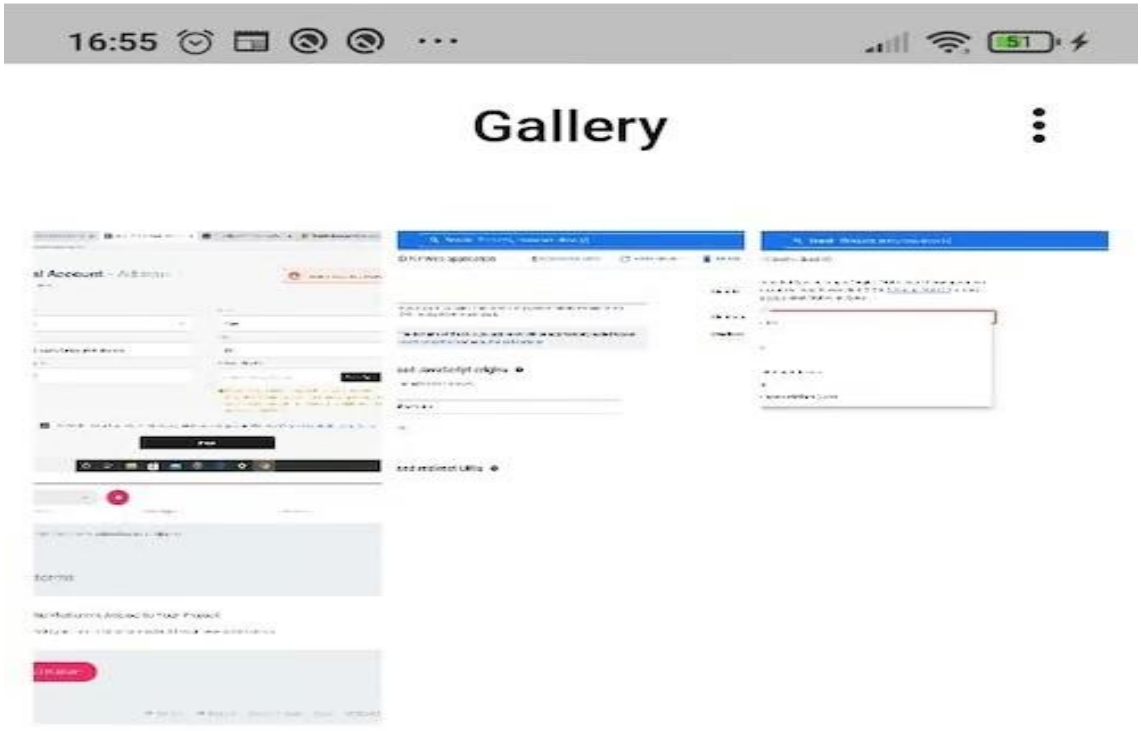
```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

import 'gallerywidget.dart';

const whitecolor = Colors.white;
const blackcolor = Colors.black;
class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key, required this.title}) : super(key: key);
  final String title;
  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
  final urlImages = [
    'assets/images/a.jpg',
    'assets/images/b.jpg',
    'assets/images/c.jpg',
    'assets/images/d.jpg',
  ];
  var transformedImages = [];

  Future<dynamic> getSizeOfImages() async {
    transformedImages = [];
    for (int i = 0; i < urlImages.length; i++) {
      final imageObject = {};
      await rootBundle.load(urlImages[i]).then((value) => {
            imageObject['path'] = urlImages[i],
            imageObject['size'] = value.lengthInBytes,
          });
      transformedImages.add(imageObject);
    }
  }
  @override
  void initState() {
    getSizeOfImages();
    super.initState();
```

```
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        elevation: 0,
        backgroundColor: whitecolor,
        centerTitle: true,
        title: Text(
          widget.title,
          style: const TextStyle(color: blackcolor),
        ),
        iconTheme: const IconThemeData(color: blackcolor),
      ),
      // Body area
      body: SafeArea(
          child: Column(
        crossAxisAlignment: CrossAxisAlignment.stretch,
        children: <Widget>[
          Expanded(
              child: Container(
                  padding:
                      const     EdgeInsets.symmetric(horizontal:     10,
vertical: 20),
                  decoration: const BoxDecoration(
                    color: whitecolor,
                  ),
                  child: GridView.builder(
                    gridDelegate:
                        const SliverGridDelegateWithFixedCrossAxisCount(
                      crossAxisCount: 3,
                      crossAxisSpacing: 5,
                      mainAxisSpacing: 5,
                    ),
                    itemBuilder: (context, index) {
                      return RawMaterialButton(
                        child: InkWell(
                          child: Ink.image(
                            image:
AssetImage(transformedImages\[index\]['path']),
                            height: 300,
                            fit: BoxFit.cover,
                          ),
                        ),
                        onPressed: () {
                            Navigator.push(
                              context,
```

```
                                MaterialPageRoute(
                                    builder: (context) => GalleryWidget(
                                          urlImages: urlImages,
                                          index: index,
                                        )));
                            },
                        );
                    },
                    itemCount: transformedImages.length,
                )))
            ],
        )),
    );
  }
}
```

U gornjem kodu definiramo klasu **_MyHomePageState**, koja je odgovorna za prikaz galerije slika. Metoda **initState** poziva metodu **getSizeOfImages**, koja dohvaća dimenzije slika i stvara transformirane slike koje se zatim dodaju u listu **transformedImages**. U metodi **build** prikazujemo **Scaffold** widget koji sadrži **AppBar** i **GridView** za prikaz slika.

Ovo je rezultat gornjeg koda:



Slika 1 Prikazuje izgled početne stranice aplikacije
Izvor: https://blog.logrocket.com/photo-gallery-app-flutter/#building-photo-gallery-application

4.1.2 Stvaranje organizacijske značajke

Za stvaranje organizirane značajke koja će sortirati raspored slika na temelju veličine i naziva, potrebno je unijeti sljedeći kod unutar datoteke **homepage.dart**, iznad funkcije **initState**:

```
Future<dynamic> sortImagesByIncreseSize() async {
  transformedImages.sort((a, b) => a['size'].compareTo(b['size']));
}
Future<dynamic> sortImagesByDecreseSize() async {
  transformedImages.sort((b, a) => a['size'].compareTo(b['size']));
}
Future<dynamic> sortImagesByNamesIncrease() async {
  transformedImages.sort((a, b) => a['path'].compareTo(b['path']));
}
Future<dynamic> sortImagesByNamesDecrease() async {
  transformedImages.sort((b, a) => a['path'].compareTo(b['path']));
}
```

Gore navedene funkcije će se pozvati nakon što kliknemo na odgovarajući gumb koji ćemo povezati s njima.

Zatim želimo koristiti te funkcije povratnog poziva. Stvorit ćemo svojstvo **actions** unutar **AppBar**-a s četiri **TextButton**-a: sortiranje po nazivu uzlazno, sortiranje po nazivu silazno, sortiranje po veličini uzlazno i sortiranje po veličini silazno.

Unesite sljedeći kod za gore navedeno:

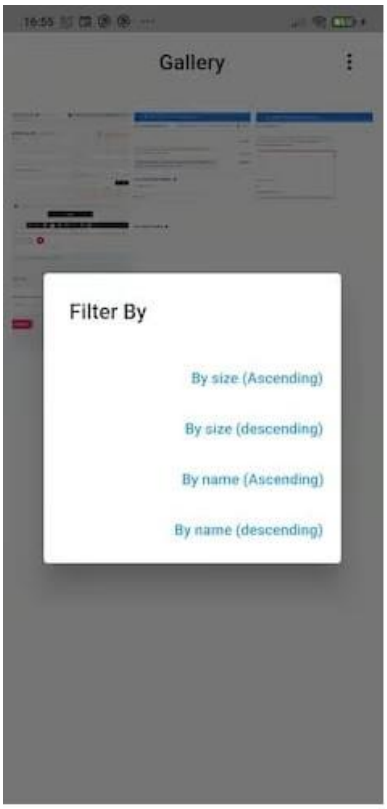
```
actions: <Widget>[
  GestureDetector(
    onTap: () {
      // show the dialog
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: const Text("Filter By"),
            // content: const Text("This is my message."),
            actions: [
              TextButton(
                child: Column(
                  children: const [
                    Text('By size (Ascending)'),
                  ],
                ),
                onPressed: () {
                  sortImagesByIncreseSize()
```

```
                    .then((value) => setState(() {}));
                Navigator.pop(context);
              },
            ),
            TextButton(
              child: Column(
                children: const [
                  Text('By size (descending)'),
                ],
              ),
              onPressed: () {
                sortImagesByDecreseSize()
                    .then((value) => setState(() {}));
                Navigator.pop(context);
              },
            ),
            TextButton(
              child: Column(
                children: const [
                  Text('By name (Ascending)'),
                ],
              ),
              onPressed: () {
                sortImagesByNamesIncrease()
                    .then((value) => setState(() {}));
                Navigator.pop(context);
              },
            ),
            TextButton(
              child: Column(
                children: const [
                  Text('By name (descending)'),
                ],
              ),
              onPressed: () {
                sortImagesByNamesDecrease()
                    .then((value) => setState(() {}));
                Navigator.pop(context);
              },
            ),
          ],
        );
      },
    );
  },
  child: Container(
    margin: const EdgeInsets.only(right: 20),
```

```
      child: const Icon(Icons.more_vert),
    ),
  )
],
```

Gore navedeni kod dodaje **GestureDetector** koji otvara dijaloški okvir kada se klikne na ikonu za više opcija u **AppBar**-u. Dijaloški okvir sadrži četiri **TextButton**-a koji će pokrenuti odgovarajuće funkcije sortiranja. Nakon sortiranja, koristimo **setState** kako bismo osvježili prikaz s novim rasporedom slika.

Slijedeća slika prikazuje trenutni izgled naše aplikacije:



Slika 2 Prikazuje organizacijske značajke u aplikaciji
Izvor:https://blog.logrocket.com/photo-gallery-app-flutter/#building-photo-gallery-application

4.1.3 Usmjeravanje na indeks slike

Cilj ovog dijela je naučiti kako prikazati svaku sliku nakon što je kliknemo. Kako bismo to postigli, koristit ćemo **onTap** svojstvo unutar **GridView.builder** widgeta.

Nakon što kliknemo sliku, želimo da nas odvede na novu stranicu s **GalleryWidget** klasom. Stvorit ćemo novu klasu **GalleryWidget** unutar nove datoteke pod nazivom **gallerywidget.dart**.

Unutar **GalleryWidget** klase, želimo imati mogućnost pregledavanja naših slika i dobivanja slika pomoću njihovog indeksa nakon što ih kliknemo. Da bismo to postigli, stvorit ćemo **pageController** koji ćemo koristiti za kontrolu indeksa stranica.

Zatim ćemo kreirati **build** metodu unutar koje ćemo koristiti **PhotoViewGallery** widget iz **photo_view** paketa koji smo prethodno instalirali.

Unutar **build** metode, koristit ćemo **itemBuilder** svojstvo **PhotoViewGallery** widgeta kako bismo izgradili svaku sliku koristeći indeks. Svaka slika bit će omotana u **PhotoViewGalleryPageOptions** widget koji dolazi iz **photo_view** paketa, a slika će se postaviti pomoću **AssetImage** klase.

Na kraju, koristit ćemo **GestureDetector** unutar **GridView.builder** kako bismo dodali funkcionalnost **onTap**. Kada se slika klikne, koristit ćemo **Navigator** da bismo otvorili novu stranicu s **GalleryWidget** klasom.

Kod koji je potrebno unijeti za prethodno opisano:

```
import 'package:flutter/material.dart';
import 'package:photo_view/photo_view_gallery.dart';

import 'homepage.dart';

class GalleryWidget extends StatefulWidget {

  final List<String> urlImages;
  final int index;
  final PageController pageController;
  // ignore: use_key_in_widget_constructors
  GalleryWidget({
    required this.urlImages,
    this.index = 0,
  }) : pageController = PageController(initialPage: index);
  @override
  State<GalleryWidget> createState() => _GalleryWidgetState();
}
class _GalleryWidgetState extends State<GalleryWidget> {
  var urlImage;
  @override
  void initState() {
    provider = AssetImage(widget.urlImages[widget.index]);
    super.initState();
  }
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        elevation: 0,
        backgroundColor: whitecolor,
        centerTitle: true,
        title: const Text(
          'Gallery',
          style: TextStyle(color: blackcolor),
        ),
        iconTheme: const IconThemeData(color: blackcolor),
        leading: IconButton(
            onPressed: () => Navigator.of(context).pop(),
            icon: const Icon(Icons.arrow_back)),
      ),
      body: Column(
        children: <Widget>[
          Expanded(
            child: PhotoViewGallery.builder(
              pageController: widget.pageController,
              itemCount: widget.urlImages.length,
              builder: (context, index) {
                urlImage = widget.urlImages[index];
                return PhotoViewGalleryPageOptions(
                  imageProvider: AssetImage(urlImage),
                );
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

4.1.4 Izrada donje navigacijske trake

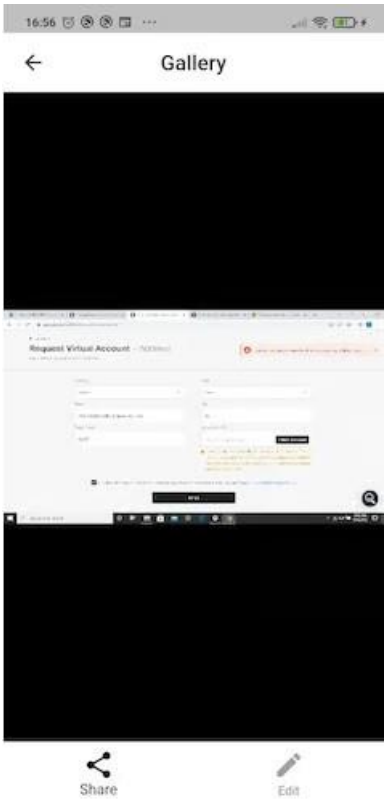
Sada ćemo stvoriti donju navigacijsku traku. Kako bismo to postigli, koristit ćemo **bottomNavigationBar** widget pozicioniran ispod tijela aplikacije. Unutar tog widgeta, definirat ćemo **onTap** funkciju koju ćemo koristiti za navigaciju na novu stranicu za uređivanje. Želimo stvoriti **BottomNavigationBarItem** s oznakom "Edit" (Uredi).

Vratimo se na **onTap** svojstvo, jer ćemo trebati dobiti odabranu stavku navigacijske trake. To možemo postići postavljanjem indeksa u **_GalleryWidgetState** klasi.

Za implementaciju funkcionalnosti, koristit ćemo sljedeći kod:

```
int bottomIndex = 0;
bottomNavigationBar: BottomNavigationBar(
  onTap: (e) {
    setState(() {
      bottomIndex = e;
      if (e == 1) {
        Navigator.push(
            context,
            MaterialPageRoute(
                builder: (context) => EditScreen(image: urlImage)));
      }
    });
  },
  currentIndex: bottomIndex,
  backgroundColor: Colors.white,
  iconSize: 30,
  selectedItemColor: Colors.black,
  unselectedIconTheme: const IconThemeData(
    color: Colors.black38,
  ),
  elevation: 0,
  items: const <BottomNavigationBarItem>[
    BottomNavigationBarItem(icon:      Icon(Icons.share),      label:
'Share'),
    BottomNavigationBarItem(
      icon: Icon(Icons.edit),
      label: 'Edit',
    ),
  ],
),
```

Prikaz kako izgleda rezultat koda:



Slika 3 Prikazuje usmjeravanje na indeks slike
Izvor:https://blog.logrocket.com/photo-gallery-app-flutter/#building-photo-gallery-application

4.1.5 Stvaranje značajke za uređivanje

Započet ćemo stvaranjem nove Dart datoteke nazvane "edit_screen.dart". Unutar te datoteke, definirat ćemo klasu "EditScreen", koja će biti povezana s našom navigacijskom trakom. U ovoj klasi ćemo stvoriti dvije značajke za uređivanje slika: rotaciju i okretanje.

Kada korisnik klikne gumb za uređivanje, bit će preusmjeren na stranicu za uređivanje, gdje će vidjeti sliku na cijelom zaslonu i imati mogućnost odabira opcija za uređivanje slike. Također ćemo stvoriti funkciju za vraćanje slike na zadanu vrijednost. Dodatno, definirat ćemo dvije funkcije: **_flipHorizon** i **_flipVert**, koje će se koristiti za okretanje slike vodoravno i okomito. Na kraju, implementirat ćemo funkciju za rotaciju slike i postaviti vrijednost funkcije na naziv "handleOption".

Ovdje je implementacija prethodno opisanog:

```
import 'dart:convert';
import 'dart:typeddata';
import 'package:flutter/material.dart';
```

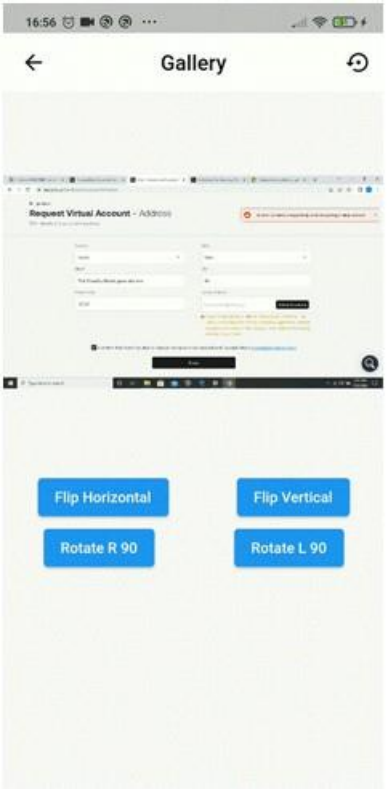
```dart
import 'package:flutter/services.dart';
import 'package:imageeditor/imageeditor.dart';
import 'package:imagesizegetter/imagesize_getter.dart';
import 'homepage.dart';

class EditScreen extends StatefulWidget {
  String image;
  EditScreen({required this.image, Key? key}) : super(key: key);
  @override
  State<EditScreen> createState() => _EditScreenState();
}
class _EditScreenState extends State<EditScreen> {
  ImageProvider? provider;
  bool horizon = true;
  bool vertic = false;
  int angle = 1;
  int angleConstant = 90;
  @override
  void initState() {
    provider = AssetImage(widget.image);
    super.initState();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        elevation: 0,
        backgroundColor: whitecolor,
        centerTitle: true,
        title: const Text(
          'Gallery',
          style: TextStyle(color: blackcolor),
        ),
        iconTheme: const IconThemeData(color: blackcolor),
        actions: <Widget>[
          IconButton(
            icon: const Icon(Icons.settings_backup_restore),
            onPressed: restore,
            tooltip: 'Restore image to default.',
          ),
        ],
      ),
      body: Column(
        children: <Widget>[
          if (provider != null)
            AspectRatio(
              aspectRatio: 1,
```

```
              child: Image(
                image: provider!,
              ),
            ),
          Expanded(
            child: Scrollbar(
              child: SingleChildScrollView(
                child: Column(
                  children: [
                    Row(
                      mainAxisAlignment:
MainAxisAlignment.spaceAround,
                      children: <Widget>[
                        ElevatedButton(
                          child: const Text('Flip Horizontal'),
                          onPressed: () {
                            _flipHorizon(FlipOption(
                                horizontal:    horizon,    vertical:
vertic));
                          },
                        ),
                        ElevatedButton(
                          child: const Text('Flip Vertical'),
                          onPressed: () {
                            _flipVert(FlipOption(
                                horizontal:    vertic,    vertical:
horizon));
                          },
                        ),
                      ],
                    ),
                    Row(
                      mainAxisAlignment:
MainAxisAlignment.spaceAround,
                      children: [
                        ElevatedButton(
                          child: const Text('Rotate R 90'),
                          onPressed: () {
                            _rotate(RotateOption(angleConstant     *
angle!));
                            setState(() {
                              angle = angle + 1;
                              if  ((angleConstant  *  angle)  >  360)
angle = 1;
                            });
                          },
```

```
                        ),
                        ElevatedButton(
                          child: const Text('Rotate L 90'),
                          onPressed: () {
                            _rotate(
                                RotateOption(360 - (angleConstant *
angle!)));
                            setState(() {
                              angle = angle + 1;
                              if ((angleConstant * angle) > 360)
angle = 1;
                            });
                          },
                        )
                      ],
                    )
                  ],
                ),
              ),
            ),
          ),
        ],
      ),
    );
  }
  void setProvider(ImageProvider? provider) {
    this.provider = provider;
    setState(() {});
  }
  void restore() {
    setProvider(AssetImage(widget.image));
  }
  Future<Uint8List> getAssetImage() async {
    final ByteData byteData = await rootBundle.load(widget.image!);
    return byteData.buffer.asUint8List();
  }
  Future<void> _flipHorizon(FlipOption flipOption) async {
    handleOption(<Option>[flipOption]);
    setState(() {
      horizon = !horizon;
    });
  }
  Future<void> _flipVert(FlipOption flipOption) async {
    handleOption(<Option>[flipOption]);
    setState(() {
      horizon = !horizon;
    });
```

```
  }
  Future<void> _rotate(RotateOption rotateOpt) async {
    handleOption(<Option>[rotateOpt]);
  }
  Future<void> handleOption(List<Option> options) async {
    final ImageEditorOption option = ImageEditorOption();
    for (int i = 0; i < options.length; i++) {
      final Option o = options[i];
      option.addOption(o);
    }
    option.outputFormat = const OutputFormat.png();
    final Uint8List assetImage = await getAssetImage();
    final                          srcSize                          =
ImageSizeGetter.getSize(MemoryInput(assetImage));
    print(const                                JsonEncoder.withIndent('
').convert(option.toJson()));
    final Uint8List? result = await ImageEditor.editImage(
      image: assetImage,
      imageEditorOption: option,
    );
    if (result == null) {
      setProvider(null);
      return;
    }
    final resultSize = ImageSizeGetter.getSize(MemoryInput(result));
    print('srcSize: $srcSize, resultSize: $resultSize');
    final MemoryImage img = MemoryImage(result);
    setProvider(img);
  }
}
```

Ovdje je prikaz rezultata gornjeg koda:



Slika 4 Prikazuje značajke za uređivanje u aplikaciji
Izvor:https://blog.logrocket.com/photo-gallery-app-flutter/#building-photo-gallery-application

4.1.6 Zaključak primjera

Ovaj primjer prikazuje izradu višeplatformske mobilne aplikacije za fotogaleriju pomoću Fluttera i Darta. Aplikacija se sastoji od četiri glavne komponente: izrada početne stranice, stvaranje organizacijske značajke, usmjeravanje na indeks slike i izrada navigacijske trake unutar ruta.

U prvom dijelu opisan je proces izrade početne stranice i definiranje klase "MyHomePage" u Dart datoteci. U ovoj klasi, aplikacija prikazuje popis slika koristeći "GridView.builder" widget, pri čemu se putanje do slika dohvaćaju iz mape "assets" unutar projektnog direktorija. Također, koristi se funkcija "getSizeOfImages" za dobivanje veličine svake slike. Scaffold widget sadrži appBar i grid za prikaz slika.

U drugom dijelu opisano je stvaranje organizacijske značajke za sortiranje rasporeda slika na temelju njihove veličine i naziva. Unutar datoteke "homepage.dart" definirane su četiri funkcije povratnog poziva Future<dynamic> za sortiranje, a TextButton widgeti su dodani u appBar i povezani s tim funkcijama.

U trećem dijelu opisano je usmjeravanje na indeks slike kako bi se prikazala odabrana slika nakon što se klikne na nju. To se postiže dodavanjem onPressed svojstva za svaku sliku

unutar "GridView.builder" widgeta. Kroz ovaj primjer stekli smo osnovno razumijevanje korištenja Fluttera i Darta za izradu višeplatformske mobilne aplikacije za fotogaleriju.

5. ZAKLJUČAK

5.1 Sažetak važnosti višeplatformskih mobilnih aplikacija

Višeplatformske mobilne aplikacije postaju sve popularnije i popularnije u današnjem digitalnom svijetu jer omogućuju korištenje jedne aplikacije na različitim platformama i uređajima. Flutter i Dart su programski jezici koji omogućuju razvoj višeplatformskih mobilnih aplikacija koje su efikasne, brze i vizualno privlačne.

U ovom radu smo detaljno razmotrili kreiranje višeplatformske mobilne aplikacije korištenjem Fluttera i Darta te smo predstavili primjer aplikacije koja je razvijena korištenjem ovih tehnologija. Taj primjer je dokazao da su Flutter i Dart idealan izbor za kreiranje višeplatformskih mobilnih aplikacija, s obzirom na jednostavnost razvoja, brzinu izrade i izvrsne performanse.

Ukratko, višeplatformske mobilne aplikacije su važne jer omogućuju korisnicima pristup aplikacijama na različitim platformama i uređajima. Flutter i Dart su moćni alati za razvoj takvih aplikacija, što je dokazano primjerom iz prakse. Stoga, njihova važnost i primjena u razvoju mobilnih aplikacija očekuju se da će rasti u sve većem broju u budućnosti.

5.2 Prednosti korištenja Fluttera i Darta u izradi višeplatformskih mobilnih aplikacija

Flutter i Dart donose brojne prednosti u izradi višeplatformskih mobilnih aplikacija. Najvažnije prednosti su:

- Brzo razvojno vrijeme: Flutter omogućuje brzu izgradnju aplikacija putem svojih „Hot reload“ funkcija. Razvojni timovi mogu vidjeti promjene u stvarnom vremenu što značajno ubrzava proces razvoja aplikacije.
- Jedinstven kod: Flutter koristi jedinstven kod koji se može koristiti na različitim platformama, što uvelike smanjuje vrijeme i troškove koje je potrebno za izradu aplikacije za svaku platformu posebno.
- Modularni dizajn: Flutter omogućuje razvoj modularne aplikacije koja se sastoji od mnogih manjih widgeta, čime olakšava razvoj i održavanje aplikacije.
- Brzina izvođenja: Dart je visoko performantan jezik koji pruža brzo izvođenje aplikacija i efikasno korištenje resursa.
- Bogata biblioteka: Flutter ima bogatu biblioteku widgeta koji se mogu koristiti za izradu aplikacija za različite platforme, uključujući iOS, Android, web i desktop.

- Snažna mogućnost prilagođavanja: Flutter omogućuje prilagođavanje izgleda i osjećaja aplikacije za različite platforme. Svojom mogućnosti „Hot reload“ programeri mogu vidjeti promjene u stvarnom vremenu i tako lakše prilagođavati izgled i ponašanje aplikacije.

- Stabilna i pouzdana aplikacija: Flutter i Dart pružaju stabilan i pouzdan okvir za izradu aplikacija, što povećava kvalitetu i pouzdanost aplikacija koje se razvijaju.

Sve u svemu, Flutter i Dart nude mnoge prednosti u izradi višeplatformskih mobilnih aplikacija. Brzina razvoja, jedinstven kod, modularni dizajn, brzina izvođenja, bogata biblioteka widgeta, snažne mogućnosti prilagođavanja i pouzdanost samo su neke od prednosti koje omogućuju razvoj visokokvalitetnih aplikacija za različite platforme.

5.3 Mogućnosti za daljnji razvoj višeplatformskih mobilnih aplikacija koristeći Flutter i Dart

Flutter i Dart su relativno novi alati za razvoj višeplatformskih mobilnih aplikacija, stoga postoji mnogo mogućnosti za daljnji razvoj u budućnosti. Ovi alati su prvobitno stvoreni za razvoj mobilnih aplikacija, no već su proširili svoju upotrebu i na druga područja, poput web i desktop aplikacija.

Jedna od mogućnosti za daljnji razvoj je unaprjeđenje performansi aplikacija. Flutter je poznat po svojoj brzini i fluidnosti, no uvijek postoji prostor za poboljšanje. Razvijatelji mogu nastaviti raditi na optimizaciji koda i smanjenju vremena potrebnog za učitavanje aplikacija.

Druga mogućnost je podrška za još više platformi. Trenutno Flutter podržava iOS, Android, web, macOS, Windows i Linux, no moguće je da će se u budućnosti dodati podrška za druge platforme.

Također, očekuje se pojava novih widgeta i biblioteka koje će olakšati razvoj mobilnih aplikacija. Flutter već ima mnogo predložaka i widgeta koje razvijatelji mogu koristiti kako bi ubrzali razvoj aplikacija, no uvijek postoji mogućnost stvaranja novih i poboljšanja postojećih widgeta.

Dodatno, Dart je uključen i u mnoge druge projekte izvan Fluttera, kao što su Google Ads i Google Assistant, što ukazuje na to da će se daljnji razvoj ovog jezika vjerojatno proširiti i na druga područja.

Konačno, Flutter i Dart su open source projekti, što znači da se svi mogu uključiti u razvoj i doprinijeti razvoju tih alata. To omogućuje zajednici razvijatelja da doprinese razvoju ovih alata i učini ih boljima u budućnosti.

IZJAVA

Izjava o autorstvu završnog rada i akademskoj čestitosti

Ime i prezime studenta: Hrvoje Marinović

Matični broj studenta: 0234061205

Naslov rada: Razvoj višeplatformskih mobilnih aplikacija Flutter/Dart

Pod punom odgovornošću potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada.

Potvrđujem da je elektronička verzija rada identična onoj tiskanoj te da je to verzija rada koju je odobrio mentor.

Datum Potpis studenta

_____ _____

6. POPIS LITERATURE

6.1 KNJIGE I ČLANCI

1) Felker, D. (2019). Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter and Dart 2. Birmingham, UK: Packt Publishing.

2) Shotts, W. (2020). Flutter Cookbook: Recipes for building fast, responsive, and native mobile applications. Sebastopol, CA: O'Reilly Media.

3) Thapa, R. (2021). Dart Programming for Flutter: Build App, Games and Websites. New York, NY: Independently published.

6.2 INTERNETSKI IZVORI

1) Flutter - Beautiful native apps in record time. (2023). Flutter. https://flutter.dev/

2) Dart. (2023). Dart. https://dart.dev/

3) LogRocket https://blog.logrocket.com/photo-gallery-app-flutter/#building-photo-gallery-application

4) Flutter Community. (2023). Medium. https://medium.com/flutter-community

5) Flutter Widgets. (2023). Flutter.dev. https://flutter.dev/docs/development/ui/widgets

6) FlutterFire. (2023). Firebase. https://firebase.flutter.dev/

7. POPIS SLIKA