Razvoj višeplatformskih mobilnih aplikacija korištenjem Flutter platforme

Vranjić, Petra

VELEUČILIŠTE U ŠIBENIKU

ODJEL POSLOVNA INFORMATIKA

PREDDIPLOMSKI STRUČNI STUDIJ POSLOVNA INFORMATIKA

Petra Vranjić

RAZVOJ VIŠEPLATFORMSKIH MOBILNIH APLIKACIJA KORIŠTENJEM FLUTTER PLATFORME

Završni rad

Šibenik, 2021.

VELEUČILIŠTE U ŠIBENIKU

ODJEL POSLOVNA INFORMATIKA

PREDDIPLOMSKI STRUČNI STUDIJ POSLOVNA INFORMATIKA

RAZVOJ VIŠEPLATFORMSKIH MOBILNIH APLIKACIJA KORIŠTENJEM FLUTTER PLATFORME

Završni rad

Kolegij: Razvoj mobilnih aplikacija

Mentor(ica): dr.sc. Frane Urem, prof. v. š.

Student(ica): Petra Vranjić

Matični broj studenta(ice): 0243099604

Šibenik, kolovoz 2021.

TEMELJNA DOKUMENTACIJSKA KARTICA

Veleučilište u Šibeniku
Odjel Poslovna informatika
Preddiplomski stručni studij Poslovna informatika

Završni rad

RAZVOJ VIŠEPLATFORMSKIH MOBILNIH APLIKACIJA KORIŠTENJEM FLUTTER PLATFORME

PETRA VRANJIĆ
pvranjic@vus.hr

Rad opisuje Flutter razvojni alat i Dart, programski jezik na kojem je on zasnovan. Nudi pregled njihove povijest i razvoja, te današnjeg stanja. Osvrće se na njihovo konkuriranje na tržištu mobilnih aplikacija u vrijeme kada su one sveprisutne. Nudi se i kratki uvod u Dart programski jezik. Rad opisuje proces postavljanja i korištenja Flutter-a, te u konačnici kratko dokumentira izradu jednostavne aplikacije.

(31 stranica / 25 slika / 0 tablica / 6 literaturnih navoda / jezik izvornika: hrvatski)
Rad je pohranjen u digitalnom repozitoriju Knjižnice Veleučilišta u Šibeniku
Ključne riječi: Flutter, Dart, aplikacije
Mentor(ica): dr.sc. Frane Urem, prof. v. š.
Rad je prihvaćen za obranu dana:

BASIC DOCUMENTATION CARD

Polytechnic of Šibenik
Batchelor/Graduation Thesis
Department of Business IT
Professional Undergraduate/Graduate Studies of Business IT

MULTIPLATFORM MOBILE APPLICATION DEVELOPMENT USING FLUTTER

PETRA VRANJIĆ
pvranjic@vus.hr

The thesis describes the Flutter software development tool kit, and Dart, the programming language upon which Dart is based. It offers a review of their history and development, as well as their current state. It refers to how well they compete in both the web and mobile app market, at a time when those are omnipresent. It also offers a short introduction into the Dart programming language. The thesis finally describes the process of setting up and using Flutter, as well as briefly documenting the process of creating a simple application.

(31 pages / 25 figures / 0 tables / 6 references / original in Croatian language)
Thesis deposited in Polytechnic of Šibenik Library digital repository
Keywords: Flutter, Dart, applications
Supervisor: dr.sc. Frane Urem, prof. v. š.
Paper accepted:

SADRŽAJ

1. UVOD

Evolucijom mobilnog telefona u pametni telefon u ranim 2000-ima, individualne aplikacije polako su našle svoje sad već nezamjenjivo mjesto u našoj svakidašnjici. Svaki suvremeni potrošački uređaj može sadržavati čak stotine aplikacija, od prvotnih unaprijed instaliranih osnovnih aplikacija proizvođača operacijskog sustava. namijenjenih za komunikaciju putem GSM mreže ili interneta, do praktički beskonačnog broja nezavisno razvijenih aplikacija koje ispunjavaju bilo koje funkcijske potrebe u čitavom spektru između produktivnosti i razonode.
Iz ovih razloga, danas postoji čitav niz razvojnih okolina za mobilne aplikacije, svaki optimiziran za različite moguće potrebe razvojnog tima, ovisno o dostupnim resursima i znanju pojedinaca koji ga sačinjavaju.
Primarna svrha ovog rada je razrada funkcionalnosti Flutter SDK-a[1] i njegovih sastavnih dijelova, sa svrhom isticanja njihovih prednosti za potrebe višeplatformskog razvoja aplikacija za Android, iOS, i web platforme. Iako su i Android i iOS uređaji zasnovani na ARM procesorskoj arhitekturi, njihovi operacijski sustavi zahtijevaju posve drugačije skupove vještina u području programiranja.
U ranim danima izbijanja pametnih uređaja na ovim dvjema platformama, mnogi razvojni programeri morali su birati između razvoja aplikacije u potpunosti za samo jednu od njih, ili podjele svog vremena između razvoja dvaju zasebnih aplikacija, stvarajući potrebu za činjenjem različitih kompromisa i žrtvovanjem kvalitete. Iz tih razloga, nastao je niz različitih razvojnih okolina sa mogućnošću istovremenog razvoja jedne aplikacije optimizirane za dvije ili više platformi. Flutter je do svoje popularnosti među njima došao zahvaljujući ne samo podrškom Google-a i njegovih programera, već kombinacijom te prednosti sa primjenom filozofije potpune otvorenosti koda, tzv. *Open Source*, što znači da je njegov izvorni kod u svojoj potpunosti javno dostupan za modifikaciju. Danas je Flutter, upravo zahvaljujući podršci *Open Source* zajednice, mnogo bolje optimiziran kao alat, te omogućava posve besprijekornu integraciju stvorenih aplikacija kako u Android tako i u iOS. Ovo je postignuto takozvanim paketima koje zajednica može razvijati neovisno o Google-ovom Flutter razvojnom timu, a koje će ovaj rad pobliže opisati u nastavku.

[1] Eng. *Software development kit*, skup alata za razvoj softvera

2. POSTAVLJANJE FLUTTER-A

2.1. Flutter SDK

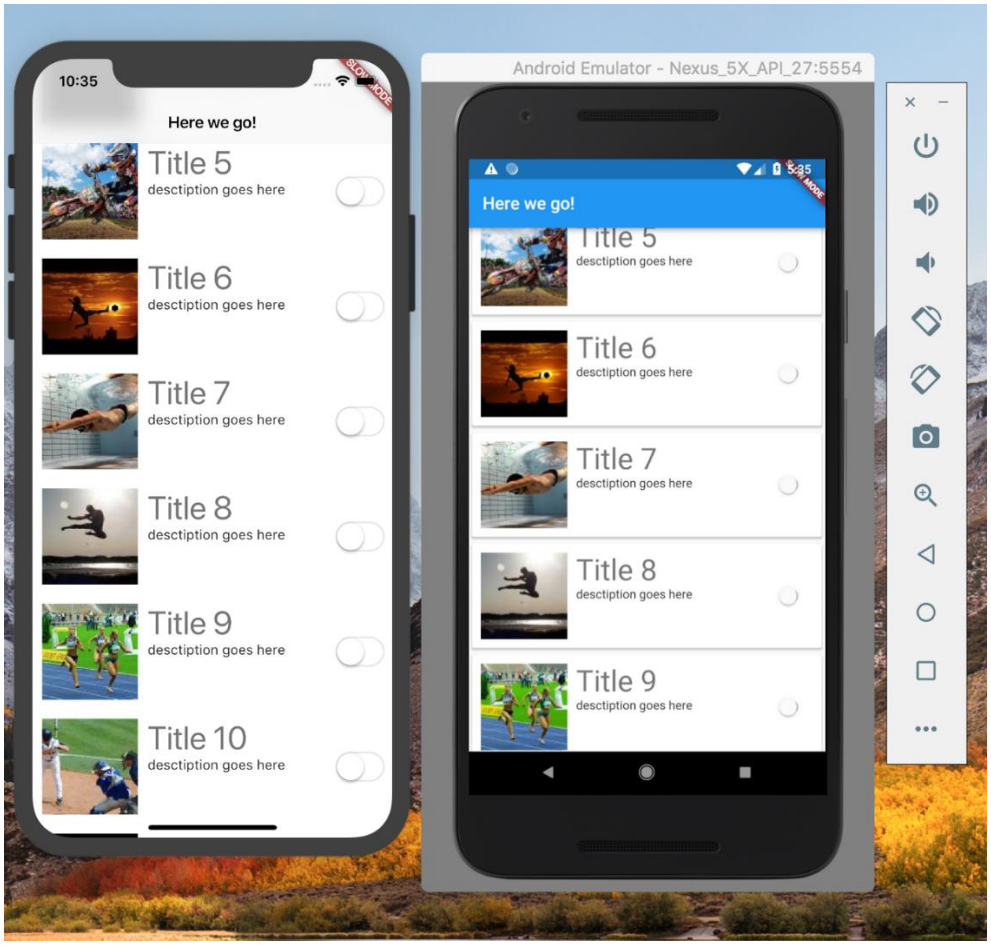
Svaki SDK, kako sam naziv predlaže, sastoji se od niza alata i drugih programskih komponenti, okupljenih iz različitih izvora sa svrhom stvaranja najbolje moguće razvojne okoline. Iako Flutter snažno potiče rad *Open Source* zajednice, to ne znači da već u početnoj instalaciji ne dolazi spreman za stvaranje više od samo osnovnih aplikacija.

Osnovna sastavnica Flutter SDK-a je Dart programski jezik i njegov pripadni SDK. O samome Dart-u rad će razglabati u idućim poglavljima, no za početak, potrebno je pojasniti njegov udio u nastanku Flutter-a. Dart SDK sadrži sve programske biblioteke potrebne za njegov osnovan rad, te *bin* direktorij sa alatima za primjenu u naredbenom retku (eng. *command line*). Ovaj dio je temelj ostatka Flutter SKD-a, pošto je čitav Flutter, dakle i sve njegove ostale sastavnice, napisan upravo koristeći Dart, uz pomoć C i C++ jezika.[2]

Nadalje, Flutter uključuje vlastiti pogon za iscrtavanje dvodimenzionalnih grafika (eng. *rendering engine*) definiranih u napisanom kodu. S obzirom na razlike u radu platformi za čiju je razvoj Flutter namijenjen, očekivana je visoka razina apstrakcije[3] u njegovoj komunikaciji između korisničkog koda i ciljnih operacijskih sustava. Većina ostalih višeplatformskih razvojnih okolina bira taj pristup pri iscrtavanju grafičkog prikaza aplikacije, jer im on dopušta da se za taj proces u potpunosti oslone na zadane grafike koje već postoje u danom sustavu. Flutter se, međutim, uvelike odrekao takvog sloja apstrakcije, stavljajući u uporabu vlastite biblioteke grafičkih elemenata, koje su povrh svega dovoljno iscrpne da osiguraju što manji broj razlika u izgledu aplikacije između pojedinih platformi. Uz tu slobodu, Flutter omogućava i uporabu grafičkih komponenti (tzv. *widgets*) koje su svjesne platforme na kojoj su iscrtane te stoga poprimaju njen pripadni zadani izgled.

[2] Biessek, A. (2020). Flutter for Beginners. Packt

[3] U ovom kontekstu se pod apstrakciju podrazumijeva korištenje i stvaranje programa i koda koji obuhvaćaju kompliciranije funkcije rada aplikacije koje korisnik ne mora vidjeti niti poznavati kako bi mogao rukovati konačnom aplikacijom putem njenog sučelja.



Slika 1. Razlike *Cupertino* i *Material widget-a* iste aplikacije stvorenih u Flutter-u.
Izvor: https://*medium.com/flutter/do-flutter-apps-dream-of-platform-aware-widgets-7d7ed7b4624d*

Widget-i kao predefinirane strukture, *Cupertino* i *Material* biblioteke, te apstrakcijski sloj sa svrhom iscrtavanja pripadaju skupu alata koji sačinjava Flutter razvojni okvir (eng. *framework*). Njegovi posljednji dio jest kolekcija osnovnih programskih klasa, koja omogućuje jednostavno stvaranje animacija, gesti, te svih ostalih kompleksnih funkcija unutar aplikacije. Flutter razvojni okvir opisuje se je kao reaktivan, što znači da je optimiziran za rad sa asinkronim tokovima podataka, što omogućuje dostavljanje i slanje podataka čim isti postanu dostupni ili su zatraženi. Drugim riječima, na bilo koji zahtjev korisnika aplikacija će odmah osvježiti relevantne podatke.[4]

Još jedan dodatan alat uključen u srž Flutter-a je *Dart DevTools* skup alata za poboljšanje performansa i *debugging* softvera u razvoju. *DevTools* sadrži niz funkcija i alata korisnih pri optimizaciji toka rada, nadzoru nad radom aplikacije, analizi napisanog koda i korištenih resursa, te otklanjanju grešaka ili drugih neželjenih učinaka rada aplikacije u pitanju.

[4] Zaccagnino, C. (2020). Programming Flutter. The Pragmatic Programmers

2.2. Razvojna okolina

Osim dosad navedenih nužnih komponenti Flutter SDK-a, potreban je i IDE[5] u kojeg će sav ovaj softver biti integriran, ili samo jednostavan softver za uređivanje teksta u kombinaciji sa Dart i Flutter *command line* naredbama. Najpopularnije razvojne okoline za programiranje aplikacija u Flutter-u su, u vrijeme pisanja, Google-ov *Android Studio* (za posvećenost Android platformi), Microsoft-ov *Visual Studio Code* (radi svoje jednostavnosti i mogućnosti prilagodbe potrebama i preferencijama korisnika), te GNU Emacs (za uporabi na više različtih računala neovisno o operacijskom sustavu, sa dodanom podrškom za GNU, Linux, FreeBSD, te niz ostalih). Ovaj će se rad za svoje potrebe referirati na korištenje Flutter-a unutar aktualne verzije *Visual Studio Code* razvojne okoline (1.60).

Visual Studio Code se, slično Flutter-u i Dart-u, također uvelike oslanja na rad programera iz *Open Source* zajednice koji ga koriste.[6] Naime, radi se o razvojnoj okolini koja je u svom početnom stanju, kako ju objavljuje Microsoft, vrlo jednostavna i lagana u vidu funkcionalnosti i preduvjeta za korištenje, ali je vrlo fleksibilna po pitanju korisničkog prilagođavanja pojedinih komponenti koje sadrži. Vrlo snažno podržava razvoj korisničkih *plugin*-ova, koji mogu obavljati bilo kakve funkcije, od mijenjanja boja koda ovisno o sintaksi odabranog programskog jezika, do integriranja čitavih razvojnih alata u razvojnu okolinu – i sam Flutter SDK može se u nekoliko klikova instalirati direktno u korisničku postavu *VS Code*-a.

2.3. Testiranje na uređajima i *hot reload*

Za svrhe testiranja i *debugging*-a, Flutter mobilne aplikacije se mogu pokrenuti na jedan od dva načina: na vanjskom uređaju koji sadrži jedan od ciljnih operacijskih sustava, ili na virtualnim instancama istih.

Proces za postavljanje kako fizičkih tako i virtualnih uređaja počinje instalacijom eksternog alata ovisno o ciljnoj platformi – za Android potreban je *Android Studio,* a za iOS *Xcode.* Ovi alati sadrže infrastrukturu potrebnu za izravno slanje izrađenih aplikacija na fizički uređaj povezan sa računalom. S njima je pakiran i softver potreban za stvaranje i pokretanje virtualnih

[5] Eng. *integrated development environment,* integrirana razvojna okolina

[6] Agrawal A., Agrawal A., Arya R., Jain H., Manoorkar J. (2021). Comparison of Flutter with Other Development Platforms. IJRCT. Dohvaćeno sa ijrct.org

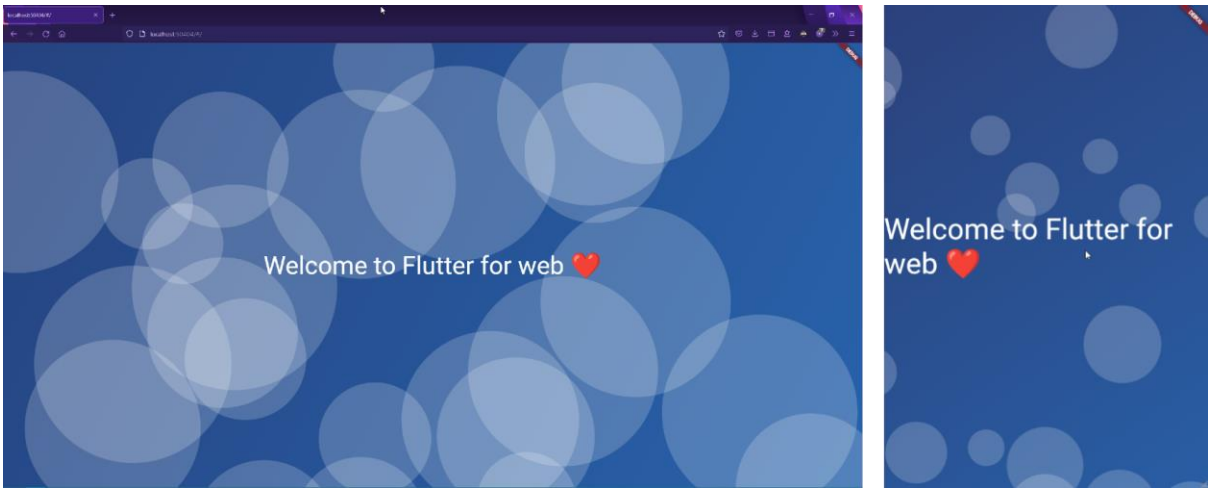
okolina koje oponašaju komponente i sustav ciljnog sustava. Potrebno je napomenuti da je *Android Studio* dostupan i na Windows i na macOS platformama, dok je *Xcode* dostupan isključivo na macOS-u. Srećom, Flutter projekti su u pravilu kompaktni i lako prenosivi između ovih dvaju platformi, ili u virtualni macOS sustav stvoren unutar Windows sustava.

Proces postavljanja ovih dodatnih alata je složen, barem u broju individualnih komponenti koje je potrebno ne samo instalirati već i konfigurirati sukladno sa radnim uvjetima na računalu u pitanju. Stoga je bitno spomenuti *doctor CLI* zapovijed, koja na korisniku jasan i čitljiv način predstavlja rezultate provjere prisutnosti i verzija pojedinih komponenti, njihovu međusobnu kompatibilnost, te savjete za ispravljanja eventualnih grešaka li nedostataka u trenutnoj postavi Flutter-a.

```
PS C:\> flutter doctor

Doctor summary (to see all details, run flutter doctor -v):

[√] Flutter (Channel stable, 2.2.3, on Microsoft Windows [Version
10.0.19043.1165], locale en-GB)

[√] Android toolchain - develop for Android devices (Android SDK
version 30.0.2)

[X] Chrome - develop for the web (Cannot find Chrome executable at
.\Google\Chrome\Application\chrome.exe)

    ! Cannot find Chrome. Try setting CHROME_EXECUTABLE to a Chrome
executable.

[√] Android Studio (version 4.1.0)

[√] VS Code (version 1.60.0)

[√] Connected device (1 available)


! Doctor found issues in 1 category.
```

Slika 2. ispis naredbe *flutter doctor* sa jednom pronađenom pogreškom, koristeći *PowerShell* unutar *Visual Studio Code* naredbenog terminala. Izvor: obrada autora.

U slučaju razvoja za web, Flutter nudi dva različita *renderer*-a, jedan poopćeni osnovni HTML *renderer,* te *CanvasKit*, optimiziran za izjednačavanje izvedbi Flutter web aplikacija između desktop i mobilnih okolina. Obojica su instalirana kao dio Flutter SDK-a i za osnovnu funkciju ne zahtijevaju dodatno postavljanje kao što je bilo potrebno kod sustava za mobilne uređaje. Najznatnija razlika između ova dva *renderer*-a web aplikacija je podatkovna veličina konačne aplikacije; naime, iako je *CanvasKit* svestraniji u svojoj implementaciji, zahvaljujući dodatnim resursima koje uključuje (primjerice mogućnost prikaza *emoji* simbola u formatu svojstvenom pojedinoj platformi), uzrokuje povećanje veličine konačne aplikacije za nekoliko megabajta, što može biti osobito utjecajno pri pokretanju na mobilnim uređajima, ili izuzetno sporim internetskim vezama.



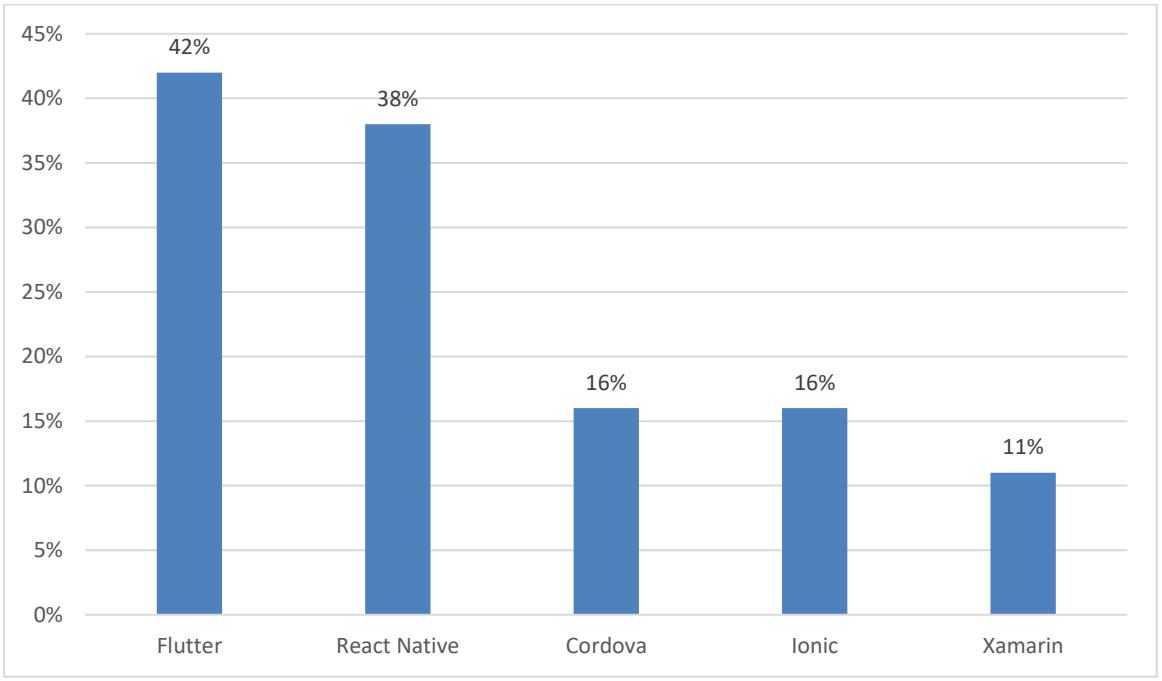
Slika 3. ista Flutter web aplikacija pokrenuta u desktop i u mobilnom načinu pregleda unutar *Mozilla Firefox* preglednika na Windows 10 Pro sustavu, koristeći *CanvasKit*. Izvor: obrada autora.

Bez obzira na izabranu izvedbu *debugging*-a korisničke Flutter aplikacije, programeru je kroz čitav proces testiranja dostupna *hot reload* funkcija. U kratkim crtama, radi se mogućnosti mijenjanja koda aplikacije za vrijeme njenog rada, sa rezultatima promjene vidljivim praktički u stvarnom vremenu. Ovakva funkcija, ili varijacije na istu, danas su dostupne u svim popularnim višeplatformskim razvojnim alatima i čine jedan od najbitnijih dijelova toka rada kod razvoja aplikacija, no Flutter-ova izvedba je upravo ono što ga ističe od, primjerice, Cordov-e ili React Native-a, dva druga najpopularnija razvojna alata na istom području. Aspekt Flutter-ovog *hot reload*-a koji uzrokuje ovoliku prednost je, očekivano, njegova brzina. Iz slike 4 vidljiva je popularnost pojedinih višeplatformskih razvojnih alata, koja je praktično proporcionalna brzini njihovih *hot reload*-ova. Ovo je posebno osjetno pri izradi iOS aplikacija, čuvenim po svom dugom vremenu kompilacije, koja uz Flutter alate može trajat i manje od 30 sekundi, dok se *hot reload* izvršava u milisekundama.[7]

Što u konačnici omogućuje ovoliku prednost u brzini? Radi se upravo u izboru programskog jezika. Dart je, između ostalog, formiran na način da se njegov izvorni kod "kompajlira" *just-in-time,* što znači da se prevodi u strojni kod ne prije početka izvedbe konačnog programa, već u isto vrijeme, slično interpretaciji.[8] JIT prevođenje, u kombinaciji sa prirodno glatkom suradnjom Flutter-ovih alata sa Dart-om, izvorni kod koji se izvodi i sustav na kojem se izvodi (fizički ili virtualni, kako je ranije opisano) mogu iznimno brzo međusobno komunicirati kako

[7] Windmill, E. (2020). Flutter in Action. Manning Publications Co.
[8] Dok ovaj proces prevođenja zvuči identičan interpretaciji, JIT kompilacija funkcionira kao hibrid interpretacije i klasične kompilacije, prevodeći izvorni kod u strojni.

bi se pregled aplikacije mogao osvježiti gotovo trenutno, i bez njenog ponovnog pokretanja.



Slika 4. Udio developera koji aktivno koriste svaku od 5 najpopularnijih platformi za višeplatformski razvoj aplikacija. Izvor: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

Još jedna prednost Flutter-ovog rada na virtualnim uređajima je njihova direktna integracija sa *Dart DevTools* paketom alata, koji pruža detaljan uvid u rad Flutter aplikacije bez obzira na platformu na kojoj se testira, i bez zauzimanja značajne količine dodatnih resursa za svoj rad.

3. KRATKI UVOD U DART JEZIK

Čitav Flutter skup alata zasnovan je na Dart programskom jeziku. Kao i sam Flutter, Dart je u osnovici podržan od strane Google-a, te također prati *Open Source* filozofiju, radi čega je suradnja između njega i Flutter-a posve besprijekorna, kao što je vidljivo na primjeru *hot reload* funkcije opisane u prethodnom poglavlju. Međutim, iako su danas tako blisko povezani, Dart je imao malo drugačiji tok razvoja.

3.1. Razvoj

Prva stabilna inačica Dart-a javno je objavljena 2013. godine, sa ciljem da u budućnosti preuzme mjesto JavaScript-a.[9] Dok se oba jezika mogu koristiti i za web i za mobilni razvoj, Dart-ov daljnji razvoj se pojavom Flutter-a 2017. godine gotovo potpuno okrenuo ka mobilnoj strani. Jedan od razloga radi kojeg je Flutter tim odabrao Dart kao temelj svojeg rada je niz poboljšanja koje Dart uvodi u odnosu na JavaScript, uz zadržavanje vrlo slične sintakse, kako bi prijelaz između ta dva jezika bio što bezbolniji. No, pošto je Dark na prvi pogled toiliko sličan JavaScriptu, zašto postoji kao zaseban programski jezik?

Njihove glavne razlike leže takoreći duboko ispod haube. Učeći na JS-ovom primjeru, Dart je svojim kasnim početkom imao priliku od samih temelja biti fleksibilniji i stabilniji programski jezik, uz jednak ili veći nivo jednostavnosti i sažetosti sintakse. Jedina, no poprilično velika, prednost koju JS ima nad Dart-om je opširnost dokumentacije. Iako Dart-ovi inženjeri rade poprilično dobar posao na tom području, srž ove razlike je jednostavno razlika u dobi ovih jezika – JS se u svrhe web razvoja koristi još od 1996. godine, te je stoga gotovo svaka moguća očekivana i neočekivana pojava u radu s njime vrlo vjerojatno već opisana do posljednjeg detalja.

Pošto je JavaScript tako duboko i tako dugo ukorijenjen u web razvoj, Google razvojni tim se naizgled odlučio povući sa tog područja i za promjenu Dart orijentirati mobilnom razvoju, stvarajući Flutter 2017. godine. Međutim, izdanjem druge inačice Flutter-a, Flutter 2, početkom 2021. godine, među skup alata dodana je i potpuna podrška za web razvoj, što predstavlja naznaku na Dart-ov povratak u utrku za udio u web tržištu, no njegova budućnost na tom području je u vrijeme pisanja još nesigurna.

[9] JavaScript (skraćeno JS) je trenutno jedan od temeljnih jezika u razvoju weba, uz HTML i CSS, no rjeđe se koristi i u razvoju mobilnih aplikacija.

3.2. Osnovni koncepti

Poput svog prethodnika, JavaScripta, Dart je po paradigmi također objektno orijentiran programski jezik, što za neupućene ukratko znači da je svaka komponenta koda objekt, sa određenim svojstvima kojima je moguće manipulirati, i metodama koje se mogu na njemu primjenjivati. Zahvaljujući tome, i svojoj sličnosti JavaScript-u, Dart je brzo postao popularan među programerima koji su prethodno stekli znanja u bilo kojem drugom već rasprostranjenom objektno orijentiranom programskim jeziku, kao što su Java ili C#.

Za početak, poput svih ostalih programskih jezika, Dart kao najosnovnije sastavnice svog koda koristi temeljne koncepte programiranja: varijable kao reference na podatke, sa tipovima varijabli koji služe za određivanje referenciranih podataka, operatore kao osnovu za proces manipulacije i procesiranja podataka, te grananja i petlje za upravljanje tokom izvršavanja koda, što omogućuje praćenje planiranog algoritma izvedbe konačnog programa.

Nadalje, pošto ipak jest objektno orijentiran programski jezik, Dart sadrži funkcije kao jedan od predefiniranih tipova objekata. Funkcije su u srži način za sadržavanje čitavih blokova koda u samo jednom objektu kako bi konačan glavni kod bio prihvatljivo čitljiv i sažet, a pošto su funkcije istovremeno i objekti, mogu se dodjeljivati kao vrijednosti varijablama, te se na njima mogu izvršavati pripadne metode. Ovisno o tipu podatka kojeg funkcija vraća, može imati i ulogu operanda u logičkim ili aritmetičkim operacijama.

Dart je potpuno razvijen objektno orijentiran jezik, te stoga uključuje i niz naprednih koncepata poput nasljeđivanja između takozvanih superklasa i podklasa, odnosa između različitih objekata, ili složenih podatkovnih struktura. Konačno, vrijedi spomenuti i pojam *null safety*. Pošto je sadržaj svake varijable u Dart kodu objekt, taj objekt mora za kao svoju vrijednost vratiti podatke određenog tipa. Međutim, općenito podatak može imati i vrijednost *null*, što znači da je varijabla prazna i s njom se tehnički ne može operirati. Od svoje druge inačice, Dart jezik podržava *null safety*, što znači da osigurava da varijable nikad ne mogu poprimiti *null* vrijednost, osim onda kada je to specifično naznačeno za pojedine varijable.[10]

```
int? aNullableInt = null;
var z = myFunction(x, y)!;
```

Slika 5. Varijabli koja može imati vrijednost null se na tip nadodaje simbol *?*. Varijabli koja bi mogla poprimiti vrijednost null ali ne smije, na kraj deklaracije dodaje se simbol *!*.

[10] Sinha S. (2020). Quick Start Guide to Dart Programming. Apress

3.3. Osnovna struktura Dart programa

```dart
import 'package:hello_dart/hello_dart.dart';
//poziv na lokalnu biblioteku

void printString(String aString) {
  //pocetak definicije nove funkcije
  print('$aString'); //Poziv postojece funkcije unutar nove funkcije
}

void main(List<String> args) {
  var hello = "Hello Dart!";
  int x = 5, y = 6; //deklaracije i inicijalizacija varijabli
  printString(hello); //poziv prethodno definirane funkcije
  if (myMultiply(x, y) % 2 == 0) {
    //poziv funkcije iz biblioteke unutar grananja
    print('Produkt $x i $y je paran broj.\n');
  } else {
    print('Produkt $x i $y je neparan broj.\n');
  }
}
```

Slika 6: primjer koda jednostavnog konzolnog Dart programa, uz popratne komentare.

Pokretanje Dart koda počinje od `main()` funkcije, koja se nalazi u početnoj dart tekstualnoj datoteci, a sadrži početne upute za rad čitavog ostatka programa. Varijable definirane unutar `main()` funkcije dostupne su ostalim objektima koji se nalaze u istoj funkciji. Ostale funkcije koje se pozivaju unutar glavne moraju biti ili definirane u istoj dart datoteci, ili se vanjska datoteka koja ih sadrži mora pozvati na početku koda, koristeći ključnu riječ import.

Kako se u ovom primjeru radi o jednostavnom konzolnom kodu, on se može pokrenuti ili unutar terminala izabranog u konfiguraciji *VS Code*-a, ili putem naredbenog sučelja na slijedeći način:

```
C:\Users\petra\Documents\Flutter projects\hello_dart>dart run

Hello Dart!
Produkt 5 i 6 je paran broj.
```

Slika 7. pokretanje Dart programa putem naredbenog sučelja. Potrebno je lokaciju sučelja postaviti na *root* direktorij programa, te ga pokrenuti naredbom `$dart run.`

3.4. Paketi i biblioteke

Kao i kod mnogih programskih jezika visoke razine, pisanje programa u Dart-u u velikom dijelu oslanja se na njemu dostupne biblioteke. Poanta programskih biblioteka je da sadrže predefinirane nestandardne funkcije ili druge objekte korisne u različitim kontekstima, kako bi ubrzale pisanje koda, poboljšale performanse konačnog programa, ili pojednostavnile čitljivost samog koda.

Jedan klasični primjer biblioteke koji se može pronaći u mnogim programskim jezicima je biblioteka namijenjena naprednim matematičkim operacijama, te sadrži funkcije koje ih obavljaju koristeći osnovne operatore definirane u samoj srži pripadnog programskog jezika. Konkretno na primjeru Dart-a radi se o dart::math biblioteci. Potrebne biblioteke se u pravilu pozivaju na samom početku glavnog koda, ključnom riječju *import*, i pozivaju se samo one koje će se zapravo i koristiti, kako bi se smanjila konačna veličina pohrane potrebna za konačni kompajlirani program. Pojedini korisnici mogu po potrebi pisati i vlastite biblioteke, specifične za tip rada kojeg obavljaju svojim kodom. Takve biblioteke moraju se nalaziti unutar direktorija programa, u specifičnom poddirektoriju, kako je opisano u nastavku.

Dodatna funkcionalnost se Dart-u može dodati putem paketa, koji mogu sadržavati biblioteke i ostale alate napisane od strane drugih korisnika. Ovi paketi se u pravilu distribuiraju putem *pub.dev* internetskog repozitorija, o čijoj funkciji dalje razglabaju neka od slijedećih poglavlja.

Kako bi se osigurala čitljivost i lakoća uporabe korisničkih paketa, postoji određena shema za raspored podataka unutar njihovih direktorija, te se od *Open Source* zajednice koja stvara sadržaj za Dart očekuje da se barem djelomično pridržava tih pravila.[11]

[11] Potpuna uputstva za stvaranje i objavu paketa mogu se pronaći na domenama dart.dev i pub.dev

3.4.1. Nacrt sadržaja Dart paketa

```
enchilada/
  .dart_tool/
  .packages
  pubspec.yaml
  pubspec.lock
  LICENSE
  README.md
  CHANGELOG.md
  benchmark/
    make_lunch.dart
  bin/
    enchilada
  doc/
    api/
    getting_started.md
  example/
    main.dart
  lib/
    enchilada.dart
    tortilla.dart
    guacamole.css
    src/
      beans.dart
      queso.dart
  test/
    enchilada_test.dart
    tortilla_test.dart
  tool/
    generate_docs.dart
  web/
    index.html
    main.dart
    style.css
```

Slika 8. vizualni prikaz ispravne hijerarhije datoteka unutar korisničkog Dart/Flutter paketa. Izvadak iz uputstava *dart.dev* dokumentacije.

Kao što slika 5 prikazuje, pravila za formiranje Dart paketa počinju od uspostavljanja ispravne hijerarhije direktorija koji ga sačinjavaju. Određene datoteke moraju se nalaziti u korijenskom (*root*) direktoriju. Najosnovnija od tih datoteka je *pubspec.yaml,* koja sadrži metapodatke potrebne za definiranje sadržaja paketa, te njegovih ovisnosti o drugim resursima. U osnovici, radi se o tekstualnoj datoteci napisanoj YAML jezikom.[12] Metapodaci koje je nužno unijeti uključuju, između ostalog, ime paketa, numeriranu trenutnu verziju izdanja, te ciljnu verziju Dart SDK-a. Tokom razvoja paketa, za slučaj da na njemu radi više suradnika, nastaje i *pubspec.lock* datoteka, koja služi za provjeru i izjednačavanje verzija resursa koje pojedinci

[12] Šaljivo nazvan *YAML Ain't Markup Language,* YAML je univerzalan ljudima čitljiv jezik za organizaciju podataka unutar jednostavno formatiranih tekstualnih datoteka.

koriste tokom razvoja, kako bi se osigurala jedinstvena lista ovisnosti i preduvjeta potrebnih za korištenje konačnog izdanja paketa.

Ostale datoteke koje se trebaju nalaziti u glavnom direktoriju paketa zapravo su standardna pojava pri objavi bilo kakvog softvera, ne samo za razvojne alate već i za bilo koji drugi tip korisničkog softvera, od fiskalizacijskih aplikacija do videoigara. Od njih je korisniku najbitnija README datoteka, koja sadrži čitak opis rada pripadnog softvera, te po potrebi uputstva za postavljanje, dodatne komentare, i zahvale suradnicima. Nadalje, CHANGELOG datoteka sadrži tekst podijeljen u sekcije za svaku dotad objavljenu verziju, opisujući promjene, novosti, i ispravke prethodnih grešaka u radu. Ove dvije datoteke imaju u nazivu ekstenziju *.md*, što naznačuje da su pisane u Markdown jeziku[13]. Konačno, u istom direktoriju mora se nalaziti i LICENSE datoteka, koja definira uvjete uporabe priloženog softvera. U slučaju Dart paketa, preporuča se korištenje OSI[14] odobrenih licenci kako bi se, u duhu *Open Source* filozofije, pripadni paket mogao slobodno mijenjati i proširivati.

```
# 1.0.1

* Fixed missing exclamation mark in `sayHi()` method.

# 1.0.0

* **Breaking change:** Removed deprecated `sayHello()` method.
* Initial stable release.

## Upgrading from 0.1.x

Change all calls to `sayHello()` to instead be to `sayHi()`.

# 0.1.1

* Deprecated the `sayHello()` method; use `sayHi()` instead.

# 0.1.0

* Initial development release.
```

Slika 9. Primjer jednostavnog CHANGELOG teksta u Markdown jeziku, prikazanog unutar Notepad++ tekstualnog editora. Izvadak iz dart.dev dokumentacije, obrađen od strane autora.

Iduća dva poddirektorija, *lib* i *bin*, sadrže sve datoteke koje moraju biti „javne“, to jest dostupne vanjskom kodu i ostalim uključenim paketima. *lib*, kako naziv direktorija predlaže, sadrži

[13] Radi se o jeziku neovisnom o platformi, koji se koristi za napredno formatiranje jednostavnog teksta za poboljšanje čitljivosti.

[14] *Open Source Initiative* služi kao autoritet za definiranje i provjeru bilo kojeg softvera kojeg vlasnici žele označiti kao *open source*.

biblioteke, koje moraju biti u potpunosti pisane Dart-om. U slučaju da paket sadrži veći broj biblioteka, dopušteno ih je organizirati u dodatne poddirektorije unutar *lib* direktorija. Ovakvi poddirektoriji sa svrhom organiziranje mogu imati bilo kakva imena, osim *src*, što je naziv poddirektorija rezerviranog za biblioteke koje paket koristi u svome kodu i nisu namijenjene za uporabu u bilo koje druge svrhe. *lib* i njegovi poddirektoriji ne smiju sadržavat ništa osim biblioteka.

```dart
class PowerCalculator {
  void calculatePower(int myNumber,
int powerNumber) {
    int base = myNumber;
    for (int i = 1; i < powerNumber;
 i++) {
      myNumber *= base;
    }
    print(myNumber);
  }
}
```

```dart
import 'package:dart_lib/PowerCalculator.dart';

void main() {
  int myNumber;
  int powerNumber;
  print("enter a number.\n");
  myNumber = int.parse(
stdin.readLineSync()!);
  print("enter power.\n");
  powerNumber = int.parse(stdin.readLineSync()!);
  var powerCalc = PowerCalculator();
  powerCalc.calculatePower(myNumber, powerNumber);
}
```

Slika 10. Lijevo: jednostavna funkcija pohranjena u lokalnoj biblioteci *lib/PowerCalculator.dart.* Desno: jednostavni Dart konzolni program koji koristi navedenu biblioteku.

Slično *lib*-u, *bin* sadrži isključivo alate uključene u paketu, i to samo one koji su javni. U slučaju da paket sadrži alate za koje razvojni tim namjerava da ostanu privatni tom paketu, sa svrhom da vanjski softver koji ih koristi ovisi o njegovoj prisutnosti, oni se stavljaju u *tool* direktorij unutar *root*-a paketa. Jedna od svrha *bin* direktorija je i da sadrži alat za aktivaciju *bin* alata svih vanjskih paketa o kojem pripadni paket ovisi za svoje funkcioniranje. Nalazeći se u bilo kojem direktoriju paketa u pitanju, ovaj alat se može jednostavno pokrenuti sljedećom naredbom:

```
$ dart run
```[15]

U slučaju da je sadržaj paketa namijenjen za web, njegov početni (*entrypoint*) Dart kod, uključujući *main.dart* i sve ostale popratne datoteke, poput onih koje sadrže HTML i CSS kod potreban za prikaz aplikacije u web pregledniku, treba se nalaziti u direktoriju pod nazivom *web*. Potrebno je napomenuti da se biblioteke koje web aplikacija koristi još uvijek moraju nalaziti u *lib* direktoriju.

[15] Simbol $ je konvencionalna oznaka za kod ili naredbe koje se unose u terminal, to jest *command line interface*.

test i *benchmark* direktoriji sadrže kod koji omogućava provjeru funkcionalnosti aplikacije. Za svrhe testiranja, Dart razvojni tim stvorio je paket pod nazivom *test* koji svojim funkcijama pomaže pri pisanju koda koji testira kritične funkcije aplikacije. Datoteke u ovom direktoriju moraju na kraju svog imena imati „*_test*“, kako je prikazano na slici 5. Dok ovaj direktorij mora postojati za svaki kompleksni Dart paket, *benchmark* direktorij potpuno je opcionalan, te se preporuča za slučajeve u kojima je potrebno mjerenje određenih performansi pri pokretanju koda, kao što su brzina izvršavanja ili količina radne memorije u uporabi.

Direktorij *doc*, kako mu ime nalaže, sadrži dokumentaciju o paketu. Za njeno pisanje ne postoje nikakva službena pravila ili konvencije, čak niti predodređen programski jezik ili format. Jedino što se očekuje je da je dokumentacija kvalitetno i čitljivo napisana, sa podrobnim objašnjenjima svakog aspekta funkcionalnosti paketa. *doc* može sadržavati i *api* poddirektorij, koji nastaje u slučaju da razvojni programer pokrene alat *dartdoc*, koji automatski kreira osnovnu dokumentaciju na osnovi napisanog koda, u obliku jednostavnog statičnog HTML dokumenta koji sadrži definicije svih programskih elemenata korištenih u navedenom projektu.

Slika 11. izvadak statične web stranice koja nastaje pokretanjem *dartdoc* naredbe. Obrada autora.

Konačno, jedan neobavezan ali vrlo koristan direktorij u paketu je *examples.* Unutar njega moguće je uključiti primjere koda ili čitave aplikacije koje koriste dotičan paket, kako bi se korisniku prikazali optimalni načini uporabe pripadnog koda, ili pružili savjeti ili objašnjenja vezani za određene funkcije.

3.4.2. Objavljivanje Dart paketa

Nakon što je razvojni tim u prihvatljivom kapacitetu zadovoljio prethodno opisane zahtjeve komponiranja Dart paketa, može se odlučiti na dijeljenje svog paketa sa ostatkom zajednice Dart programera. Prije objave, neophodno je obaviti nekoliko konačnih provjera: osigurati da paket nakon *gzip*[16] kompresije zauzima manje od 100 megabajta prostora za pohranu, te osigurati da su sve biblioteke o kojima paket ovisi dostupne ili unutar Flutter ili Dart SDK-a, ili javno dostupne putem *pub.dev* repozitorija zahvaljujući drugim programerima unutar *Open Source* zajednice.

Pretposljednji korak je stvaranje ili korištenje Google korisničkog računa, koji se povezuje na stranice *pub.dev* repozitorija kao pokazatelj identiteta autora paketa. Osnovni korisnički račun nije potvrđen kao povjerljiv od strane zajednice, te kao komunikacijski kanal navodi samo i isključivo email adresu vezanu za Google korisnički račun u pitanju. Alternativa tome je prijava za verifikaciju računa, koja uključuje prijavu i ovjeru internetske domene koja pripada razvojnom timu o čijem se računu radi putem Google Search konzole[17], kako bi se sadržaj repozitorija zaštitio od zlonamjernih automatiziranih korisničkih računa koji bi dijelili štetan kod pod krinkom verificiranog računa.

U konačnici, posljednji korak za objavu paketa u pub repozitorij je pokretanje samo jedne linije koda u bilo kojem naredbenom sučelju postavljenom za rad sa Dart-om:

```
$ dart pub publish
```

Ista komandna linja može se koristiti i za prvotno postavljanje u repozitorij, i za njegovo ažuriranje. Prije postavljanja, moguće je istu naredbu pokrenuti sa dodatnim argumentom:

```
$ dart pub publish –-dry-run
```

Ovaj postupak pokrenut će probno postavljanje u repozitorij, tokom kojeg će *pub* alat izvršiti automatiziranu provjeru ispunjenja prethodno navedenih zahtjeva, posebice vezano za format

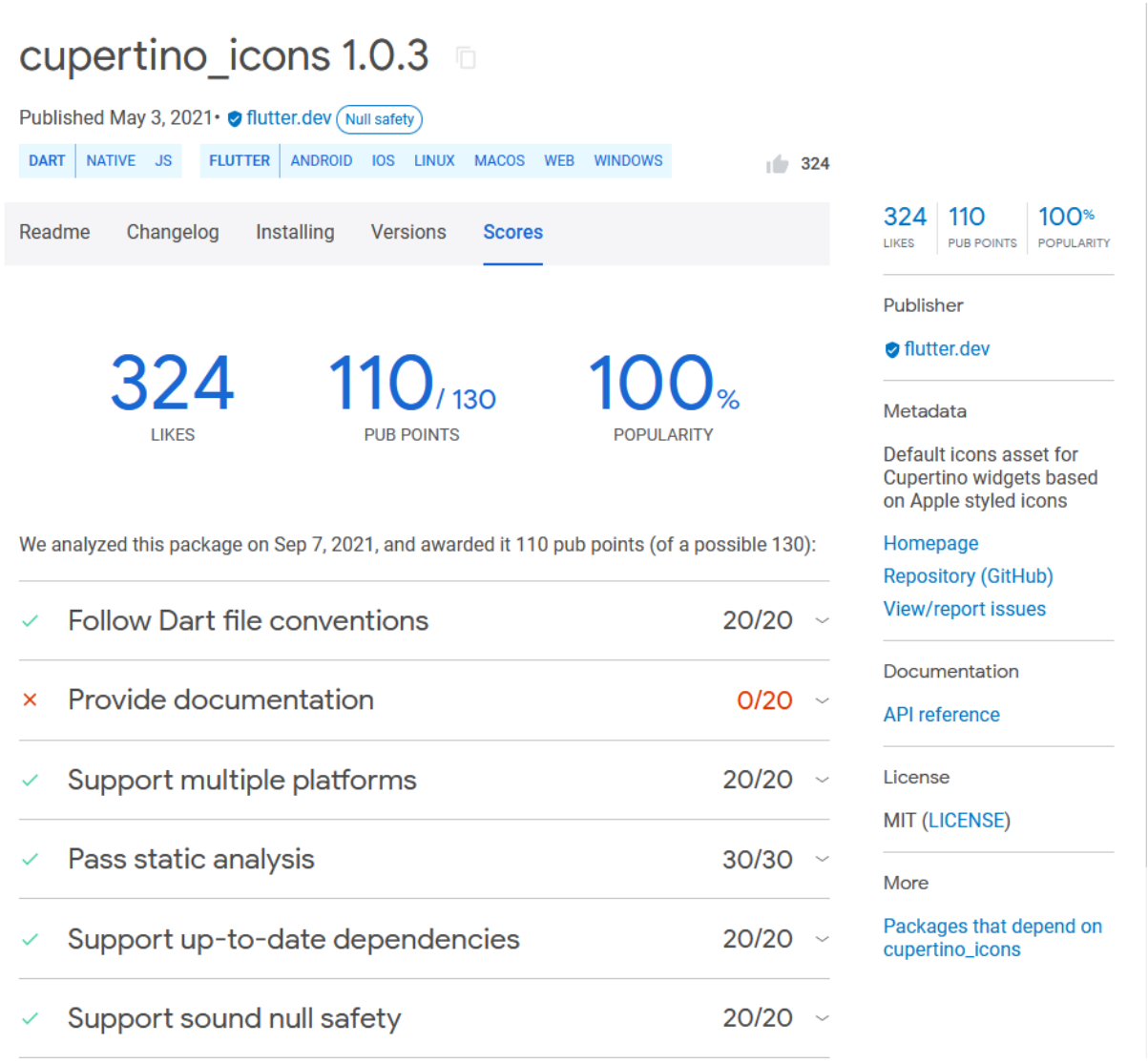
[16] *gzip* je jedan on mnogih formata kompresije digitalnih podataka, usporediv sa poznatijim *rar* i *zip* formatima, a razvijen kao dio GNU projekta slobodnih operacijskih sustava i softvera. Dart pub zajednica preferira *gzip* za svrhe dijeljenja paketa, radi njegove sukladnosti *open source* filozofiji.

[17] Radi se o provjeri DNS arhiva priložene web domene putem Google-ovih alata, koja sadrži povijest IP adresa povezanih s njom.

pubspec datoteke i hijerarhije osnovnih direktorija, no neće izvršiti konačnu objavu paketa dok se ne pokrene ista naredba sa izostavljenim *dry run* argumentom. Kada je paket objavljen, dostupan je ostatku zajednice za korištenje i modificiranje u skladu sa priloženom licencom.

3.4.3. Rangiranje i ocjenjivanje

Uz temu objave paketa vrijedi u konačnosti makar u kratkim crtama spomenuti sustav rangiranja i ocjenjivanja istih putem *pub.dev* web stranica. Pristupom osnovnom pregledu bilo kojeg paketa dostupnog na repozitoriju, uz sve njegove uobičajene informacije definirane pri objavi od strane njegovog autora, posjetitelju je dostupan i uvid u tri različite metrike po kojima se paket rangira kroz kategorije repozitorija sa ciljem isticanja najkvalitetnijih primjeraka rada *Open Source* zajednice.



Slika 12. Rubrika ocjena na jednom od trenutno najpopularnijih javno dostupnih paketa.

Na slici 12 jasno su istaknute te tri metrike. Prva od njih, *Likes,* jednostavno prikazuje broj individualnih korisnika koji su dotični paket označili kao da im se sviđa. To služi kao sirovo mjerilo korisničkog zadovoljstva rada s paketom. *Pub points,* iduće mjerilo popularnost, malo je kompleksnije. Radi se o ocjeni koja ovisi o nizu komponenti, također poimence nabrojenim na slici, od kojih svaka nosi određen dio od ukupnih 130. Prvih 20 bodova odnosi se na prethodno definirana očekivanja za oblikovanje *pubspec.yaml, README.md,* i *CHANGELOG.md* datoteka. Jednak broj bodova nosi kombinacija priložene *dartdoc* dokumentacije i pripadnih primjera uporabe u *examples* direktoriju. Još 20 bodova paket dobiva ako podržava sve tri platforme za koje su Flutter i Dart namijenjeni: Android, iOS, i web. Najveći broj bodova, 30, nosi potpun prolazak automatizirane *pub* analize koda bez grešaka, upozorenja, ili grešaka u prihvaćenim stilističkim konvencijama pisanja koda. Iduća komponenta od 20 bodova provjerava ažurnost paketa o kojima dotični paket ovisi, te kompatibilnost sa trenutnom stabilnom verzijom Flutter i Dart SDK-a. Konačnih 20 bodova dodjeljuje se paketu ako i on i svi paketi o kojima ovisi podržavaju *null safety.* Konačno, popularnost paketa računa se ovisno o tome koliko ga aplikacija objavljenih u zadnjih 60 dana koristi (100% se dodjeljuje najviše, a 0% najmanje korištenom).[18]

Ovisno o rezultatima ova tri mjerila, paketi se na stranicama *pub.dev* repozitorija ističu i rangiraju po kategorijama kvalitete, kako bi najizvrsniji uradci zajednice bili lako dostupni svim korisnicima, i primili priznanja koja zaslužuju.

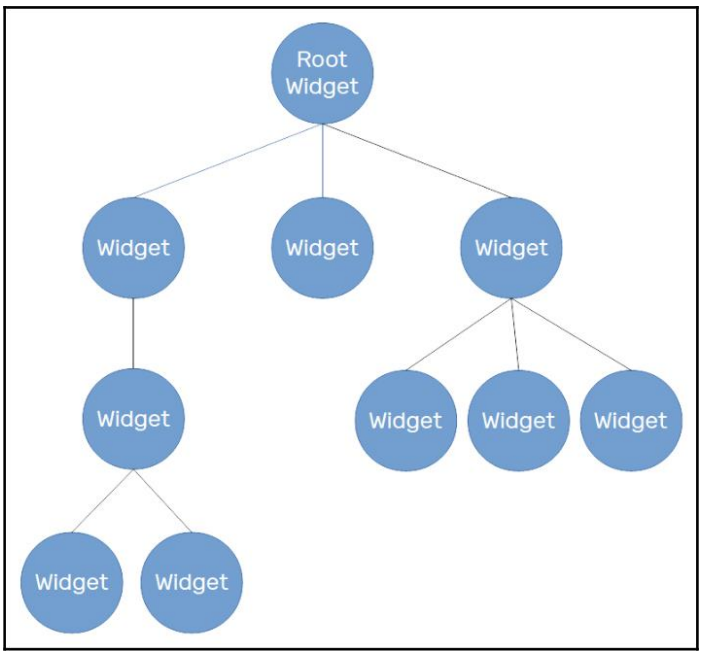
[18] Pošto se u priloženom primjeru radi o jednom od osnovnih paketa kojeg koriste praktički sve Flutter aplikacije, njegova popularnost je na 100%.

4. UVOD U WIDGET-E

Flutter alati uvelike utječu na rad u Dart-u, posebice na sam tok pisanja koda. Osnova građenja korisničkog sučelja svake Flutter aplikacije su takozvani *widget-i*. *Widget* se, u kontekstu, Flutter-a, definira kao nepromjenjiva Dart klasa objekata, koja sadrži varijable i metode potrebne za sastavljanje sučelja aplikacije, koje nastaje nizanjem i ugnježđivanjem *widget*-a. Jedna rečenica koja se često provlači kroz svu dostupnu Flutter dokumentaciju jest: *in Flutter, everything is a widget.* Svaka zasebna komponenta aplikacije je widget. Svaki gumb, tekst, margina između vizualnih elemenata, čak i sama aplikacija, sadržani su u *widget*-ima.

Kako bi svi ovi *widget*-i u konačnici stvorili funkcionalnu aplikaciju, potrebno je uspostaviti logiku njihovih međusobnih veza, takozvano stablo widgeta (*widget tree*). Ono u vizualnom obliku izlaže odnose između svih *widget*-a koji sačinjavaju aplikaciju, to jest njihovu hijerarhiju – odnos između *widget*-a roditelja (*parent*) i *widget*-a djece (*children*) koji se nalaze unutar njih. Ono uključuje i vidljive *widget*-e, i one koji nisu vidljivi krajnjem korisniku, već postoje u kodu sa svrhom manipulacije izgledom i rasporedom svojih *children* widgeta. Korijen stabla je onaj *widget* koji prvi slijedi funkciju runApp() u glavnoj funkciji Dart koda.

Pošto su *widget*-i ne samo funkcionalni već i vizualni elementi aplikacije, po načinu iscrtavanja dijele se na dvije vrste. *Stateless* widgeti (klasa StatelessWidget) nanovo se iscrtavaju samo pri pokretanju aplikacije, ili kada to zahtijeva ponovno iscrtavanje njihovih *widget*-a roditelja. S druge strane, prikaz *stateful widget*-a (klasa StatefulWidget) može se ponovno iscrtati bilo kad, pozivanjem odgovarajuće funkcije. Ovo ih čini složenijim za korištenje, pošto su neki od njihovih podataka stalno podložni promjenama, no to svojstvo otvara mogućnost za osvježavanje prikaza u bilo kojem trenutku po dolasku novih podataka.



Slika 13 shematski prikaz stabla *widget*-a. Svaki roditelj grana sa na jedno ili više djece.

Kao i bilo kakvi ostali objekti, i *widget*-i imaju niz svojstava na koja se kod aplikacije može pozvati u radu s njima. Većina tih svojstava odnosi se na njihov izgled. Na primjeru *Text widget*-a, neka od njegovih svojstava su *textDirection* (smjer teksta), *style* (stil izgleda samog teksta), i slično. Ovdje je prigodno i spomenuti kako se imena svojstva *widget*-a uvijek pišu u formatu *camelCase*.[19]

[19] *camelCase* je praksa formatiranja tekstualnih fraza na način da se pojedine riječi u frazi ne razdvajaju razmacima ili drugim znakovima, već je početak svake riječi naznačen velikim početnim tiskanim slovom, a ostatak slova je mali.

5. OSNOVNI WORKFLOW

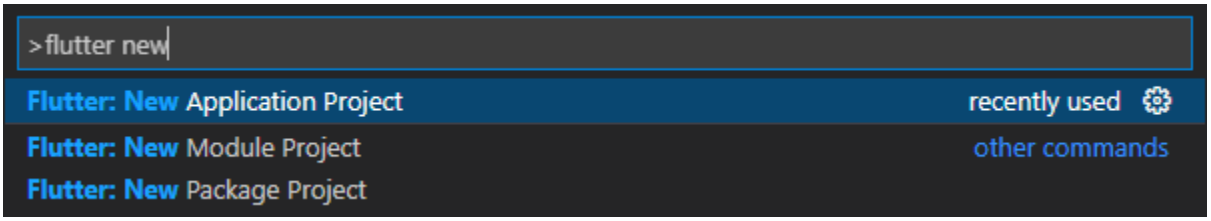
Sa završenim upoznavanjem sa Dart jezikom i konceptom *widget*-a kojeg Flutter u njega unosi, vrijeme je za započinjanje pisanja prve aplikacije. Za svrhe ovog rada, bit će prikazan proces pisanja jednostavne aplikacije kalkulatora.

5.1. Stvaranje projekta u IDE-u

Flutter projekt može se u najsirovijem obliku stvoriti putem *command line interface*-a u bilo kojem direktoriju koristeći slijedeću naredbu:

```
$flutter create my_calculator
```

U ovom primjeru argument naredbe „my_calculator“ je ime projekta čije stvaranje pokrećemo. Ista zapovijed može se iskoristiti i u *VS Code* razvojnoj okolini putem dostupnog terminala naredbene linije, ili što je još uobičajeniji način, putem *Code*-ove pametne naredbene palete upisom bilo kojeg teksta koji prirodno navodi na stvaranje novog Flutter projekta.



Slika 14. Primjer pokretanja stvaranje nove Flutter aplikacije putem *VS Code* naredbene palete. Obrada autora.

Aplikacija će biti stvorena kao direktorij koji uključuje sve ostale poddirektorije i datoteke koji su preduvjeti za korištenje Flutter-a. Također će stvoriti i jednostavnu aplikaciju, koja se u pravilu koristi kao kostur za pisanje nove aplikacije.

Za početak potrebno je pobrisati sve komentare u kodu[20] i isprazniti iz klase *_MyHomePage* sav tekst koji se nalazi unutar identifikatora *body*. Prema ovakvom kosturu koda, u toj klasi, koja je u kontekstu Flutter-a *widget*, nalazit će se većina korisnički napisanog koda. Odavde, potrebno je razmisliti o *widget*-ima od kojih će se aplikacija sastojati.[21]

[20] Linija komentara u Dart-u označava se sa simbolima „//“, a blok komentara sa „/*sadržaj komentara*/“.
[21] Freitas, E. (2019.) Flutter Succinctly. *Syncfusion, Inc.*

5.2. Slaganje widget-a

Pošto se radi o kalkulatoru, za njega je prvenstveno potrebno napraviti nekakav glavni prikaz, te gumbe za unos. Prikaz neka za početak bude jednostavan *Text* widget na samom vrhu okvira aplikacije, ispod njenog naslova. Kako bi ovaj prikaz bio postavljen na željeno mjesto, s obzirom na to da će se ispod njega nalaziti čitav niz gumbova, potrebno je stvoriti *child widget* tipa *Column,* te kao njegov prvi *child widget* dodati *Container* koji će služiti za pozicioniranje *Text widget*-a koji će biti njegov *child*. Ovome tekstu dalje treba dodijeliti neku vrijednost, koja bi u početnom stanju trebala biti 0. S tom svrhom se na samom početku *_MyHomePage widget*-a inicijalizira *String*[22] varijabla vrijednosti „0“. Vrijednost se stavlja u varijablu, a ne dodjeljuje se direktno *widget*-u, zato što želimo da nam njegova vrijednost uvijek bude dostupna i promjenjiva. Daljnja funkcionalnost i interaktivnost ovog teksta sa gumbima bit će kasnije proširena.

Pošto će biti potrebno stvoriti više istih gumba identičnih svojstava, kako bi se izbjeglo prepisivanje uvijek istog koda za svaki gumb, prigodno je napisati funkciju koja će vraćati novi *widget* gumba, sa određenim svojstvima, koja će ponovno biti sadržana u njegovim *child widget*-ima. Primarno svojstvo po kojem će se gumbi razlikovati su njihovi pripadne znamenke i simboli. Kako bi se osiguralo jednostavno stvaranje tih razlika unutar ranije spomenute funkcije za stvaranje gumba, određuje se jedan podatak tipa *String* kao argument pri pozivanju funkcije. Taj *string* bit će kasnije proslijeđen *child widget*-u *Text*. Njemu se također može pridružiti svojstvo *style,* koje se koristi za definiranje oblika teksta, poput njegove veličine, boje, fonta, poravnanja, podebljanja, i slično. Veličina fonta i ostala svojstva vezana za veličinu mogu se određivati u broju piksela neovisnim o uređaju, ili kao produkt zadane veličine teksta bilo kojeg broja. Neke od drugih opcija stila teksta predefinirane su u programskom jeziku unutar postojećih objekata koji se mogu koristiti ne samo za tekst, nego i na bilo kojem drugom mjestu u kodu koje prihvaća podatke u tom formatu. Na primjeru boje, moguće je ručno dodijeliti tekstu bilo koju boju u *hex* ili RGB formatu, no jednostavnije je i sažetije upotrijebiti jednu od boja predefiniranih u objektu *Color*. Konačno, potrebno je stvorene gumbe staviti unutar *Expanded widget*-a, koji određuje da će njegova djeca sa

[22] *String* varijable su namijenjene sadržavanju i lakom prikazu teksta. Zato što je namijenjena za tekst, ovdje se broj 0 nalazi u navodnim znakovima, kako bi ga se i tretiralo kao tekst.

jednakim udjelima razdijeliti vertikalan i horizontalan prostor koji im je dostupan. To u ovom slučaju automatski postavlja veličinu gumba kalkulatora na najveću moguću u dostupnom prikazu.

```
class _MyHomePageState extends State<MyHomePage> {
  String calcResult = "0";

  Widget makeAButton(String textValue) {
    return Expanded(
        child: MaterialButton(
            onPressed: () => buttonPressed(buttonTextValue),

            child: Text(textValue,
                style: const TextStyle(
                    fontSize: 22.0, fontWeight: FontWeight.bold)),
            textColor: Colors.white,
            color: Colors.deepPurpleAccent,
            padding: const EdgeInsets.all(20)));
  }
```

Slika 15. kod opisane metoda za stvaranje gumba. Boja je izabrana iz predefiniranih boja objekta *Color*.

Prije nego se funkcija za stvaranje gumba stav u uporabu, potrebno je razmisliti o njihovom položaju. Kako bi se gumbi poredali u više redaka, jednostavno se u postojeći *widget Column* dodaje onoliko *widget*-a djece tipa *Row* koliko je redaka potrebno, u ovom slučaju pet. Nakon toga, potrebno je samo kao djecu pojedinog retka dodati rezultate prethodno napisane funkcije njenim pozivanjem, ovdje imenovane *makeAButton*, onoliko puta koliko je gumba potrebno u pojedinom redu. Zahvaljujući roditelju *widget*-u *Expanded* definiranom u funkciji pri stvaranju pojedinog gumba, svaki od njih će zauzimati jednaku širinu u pripadnom retku, bez obzira na njihov broj. Funkcija se poziva onoliko puta koliko djece je potrebno u retku, a za argument joj se pri svakom pozivanju upisuje željeni tekst pojedinog gumba.

```
Row(
                children: [
                  makeAButton("7"),
                  makeAButton("8"),
                  makeAButton("9"),
                  makeAButton("-")
                ],
              ),
```

Slika 16. Primjer sastavljanja prvog retka kalkulatora pozivanjem funkcije za stvaranje gumba.

Nadalje, *widget* tipa gumba izabran u ovom primjeru, pod nazivom *MaterialButton*[23] za svoje funkcioniranje zahtijeva postojanje svoje pripadne metode svojstva *onPressed*, koje, kao što mu ime nalaže, definira kod koji će se pokrenuti pri pritisku na gumb. Ovu funkciju je moguće definirati bilo gdje u kodu, no može joj se dodijelit i prazan kod ako je potreban gumb koji ne radi ništa. U primjeru rada, sa svrhom bolje organiziranosti koda, *onPressed* funkcija je definirana pri vrhu *_MyHomePage* klase, ispod definiranih varijabli. Za argument, primat će *string* koji se nalazi u prethodno definiranom tekstu pojedinog gumba. Pošto funkcija koju gumb obavlja ovisi o broju ili simbolu kojeg nosi unutar sebe, funkcija će tu vrijednost primati kao argument (dijete *Text widget*-a gumba). Za odlučivanje što će gumb raditi ovisno o sadržaju pritisnutog gumba, na primjeru jednostavnog kalkulatora koji će izvršavati izračune sa dva člana i jednim operatorom, dovoljno je sastaviti *if* grananje.

Prije početka pisanja samih uvjeta grananja, definiraju se varijable koje e biti potrebne za funkciju kalkulatora. Ovdje su to dvije varijable tipa *double*[24] za pohranu brojeva na kojima se operira, varijabla tipa *String* za pohranu operatora, te još jedan *String* koji će pohraniti rezultat i u konačnici ga proslijediti na prikaz.

```
String _calcResult = "0";
double n1 = 0.0;
double n2 = 0.0;
String operator = "";
```

Slika 17. deklaracije potrebnih varijabli.

[23] Iz Flutter zadane biblioteke *material.dart*, koja sadrži elemente vizualnog dizajna svojstvene android platformi.

[24] *Double* tip varijable koriste se za pohranjivanje brojeva sa velikim brojem decimala. Ovdje je prikladan jer kalkulator sadrži i funkciju dijeljenja.

Kad su radu kalkulatora potrebne varijable dostupne i deklarirane koristeći prikladne tipove podataka, moguće je početi definirati funkcije pojedinih gumbova. Prvu od funkcija je najjednostavnije definirati – *CE*, funkciju za brisanje trenutnih podataka u kalkulatoru, prikazanih i pohranjenih. Kad je njen gumb pritisnut, i trenutni prikaz i sve varijable će se vratiti u početno stanje, nulu.

```
if (buttonTextValue == "CE") {
      _calcResult = "0";
      n1 = 0.0;
      n2 = 0.0;
      operator = "";
}
```

Slika 18. funkcija gumba *CE*.

Nadalje, ako je izabran jedan od dostupnih operatora, potrebno je pohraniti prethodno izabran broj i operator u pripadne varijable, u iščekivanju korisničkog unosa idućeg broja i pokretanja funkcije računanja pritiskom na gumb jednakosti. Uz to potrebno je i vratiti prikaz kalkulatora na zadano stanje, kako bi korisnik znao da može unijeti idući broj.

```
else if (buttonTextValue == "+" ||
        buttonTextValue == "-" ||
        buttonTextValue == "*" ||
        buttonTextValue == "/") {
      n1 = double.parse(calcResult);
      operator = buttonTextValue;
      _calcResult = "0";
    }
```

Slika 19. Kod gumba pri izabiru operatora.

Vrijednost varijable prvog broja, *n1*, kako se vidi iz priloženog mora se prvo iz tipa *string*, koji je tip unosa i teksta gumba, pretvorit u tip *double* kako bi se na njoj mogle vršiti matematičke operacije. Ovo se izvršava svojstvom varijable *parse*.

Ako je izabran gumb decimalne točke, najprije se provjerava sadrži li broj na prikazu već decimalnu točku. Ako ju već sadrži, izlazi se iz petlje i čeka idući unos, a u suprotnom točka se nadodaje na pohranjen broj.

```
else if (buttonTextValue == ".") {
      if (_calcResult.contains(".")) {
        _showSnackAlert(context);
        return;
      } else {

        _calcResult += buttonTextValue;
      }
```

Slika 20. Kod za dodavanje decimalne točke.

Kao dodatna povratna informacija korisniku, ovdje je u petlju dodana i funkcija nazvana *_showSnackAlert*. Ona će, u slučaju da je decimalna točka već u prikazu, na dnu ekrana kratko prikazati poruku koja upućuje korisnika na to da je decimalnu točko pokušao unijeti dvaput.[25]

```
void _showSnackAlert(BuildContext context) {
    final scaffold = ScaffoldMessenger.of(context);
    scaffold.showSnackBar(
      const SnackBar(
        content: Text("There is already a decimal point."),
      ),
    );
  }
```

Slika 21. Kod za pozivanje funkcije ispisa obavijesti korisniku.

Konačno, preostalo je samo definirati funkciju gumba jednakosti. Od njega se očekuje da u prikaz kalkulatora ispiše rezultat matematičke operacije između dva broja. U prethodnim funkcijama gumba već je definirana pohrana jednog broja i operatora. Pritiskom na jednakost bi se, stoga, trebao pohraniti broj koji je u tom trenutku na prikazu, te prikaz zamijeniti rezultatom tražene matematičke operacije.

[25] U Flutter-u ovaj oblik prikazivanja obavijest i poruka korisniku poznat je kao *Snack Bar*, radi čega se i primjerna funkcija slično zove.

```
else if (buttonTextValue == "=") {
      n2 = double.parse(calcResult);

      if (operator == "+") {
        _calcResult = (n1 + n2).toString();
      } else if (operator == "-") {
        _calcResult = (n1 - n2).toString();
      } else if (operator == "*") {
        _calcResult = (n1 * n2).toString();
      } else if (operator == "/") {
        _calcResult = (n1 / n2).toString();
      }
```

Slika 22. Kod za slučaj pritiska gumba sa simbolom jednakosti.

Ovdje je ponovno potrebno obratiti pozornost na prisutne tipove podataka. Brojevi u već pretvoreni u *double* tip podatka, stoga ih nije potrebno mijenjati prije obavljanja prikladne matematičke operacije. Međutim, pošto je rezultat operacija između dva *double* podatka ponovno *double*, a prikaz kalkulatora korist podatke tipa *string*, potrebno je upotrijebiti svojstvo *toString* koje će izvršiti potrebnu pretvorbu.

Nakon što *if* petlja nakon svakog pritiska gumba završi i po potrebi promijeni vrijednosti varijabli u kodu, potrebno je osvježiti stanje *widget*-a kako bi nastale promjene bile vidljive u prikazu. Ovaj proces je na primjeru kalkulatora vrlo jednostavan, jer sadrži samo promjenu teksta prikaza na vrhu aplikacije.

```
setState(() {
      calcResult = double.parse(_calcResult).toStringAsFixed(2);
    }
```

Slika 23. Kod osvježavanja trenutnog stanja *widget*-a.

Ovim posljednjim korakom završen je kod za aplikaciju jednostavnog kalkulatora koristći Flutter, no sad ga je potrebno i testirati jednom od prethodno opisanih metoda slanja Flutter aplikacija na neki od ciljnih sustava

5.3. Testiranje aplikacije

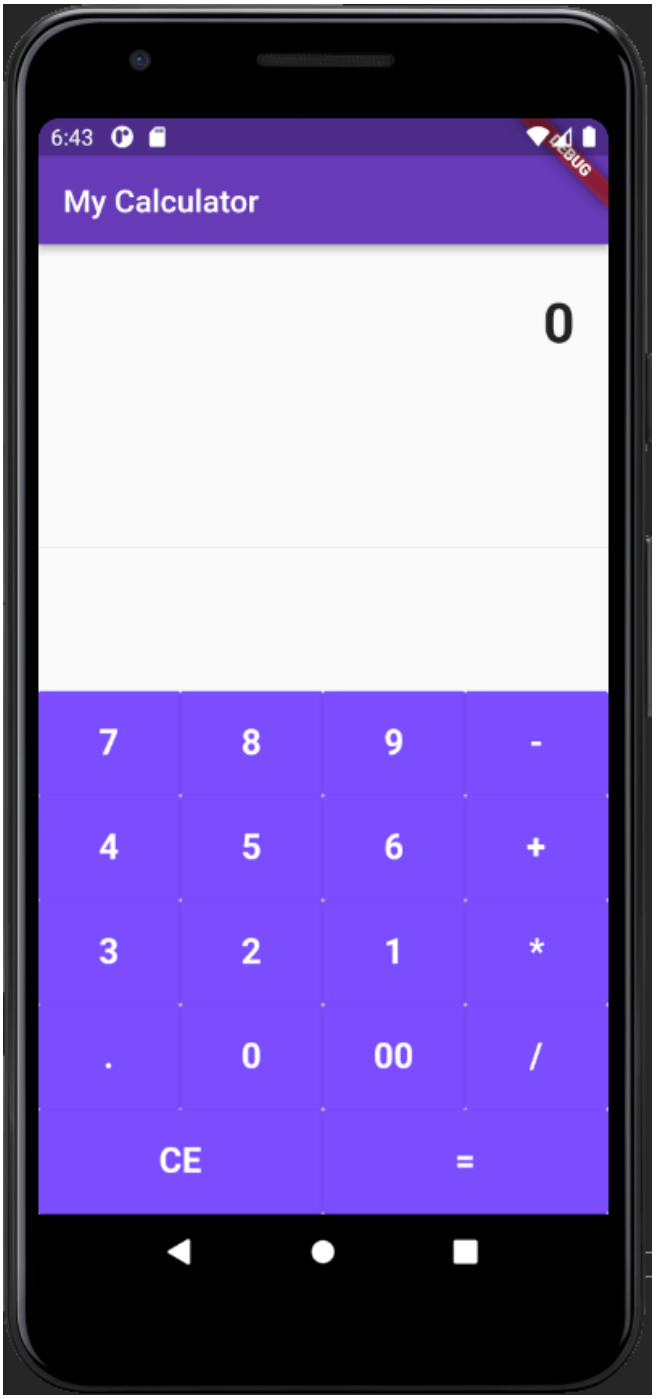
Kao što je prethodno opisano, Flutter aplikacije se u svrhe testiranja i ispravljanja mogu pokretati kako na fizičkom tako i na virtualnom uređaju, sve koristeći Android ili iOS SDK. U svrhe rada korišten je Android SD, koji je u oba slučaja vrlo jednostavan za uporabu. U slučaju slanja aplikacije na fizički uređaj, SDK ga automatski prepoznaje čim je u *debug* načinu povezan na računalo putem USB sučelja. Za pokretanje aplikacije u virtualnom okruženju, što je u većini slučajeva mnogo zgodnije za testiranje aplikacija čiji se kod često mora osvježavati, potrebno je prvo instalirati jednu od virtualnih inačica Android uređaja dostupnih putem Android SDK sučelja za postavljanje. Potom je potrebno pokrenuti virtualni uređaj te ga izabrati kao ciljnu platformu za testiranje, što je opcija koja se u *VS Code*-u u bilo kojem trenutku može naći u donjem desnom kutu. Postoji i niz drugih parametara pokretanja Flutter aplikacije svojstvenih *VS Code*-u, a oni se mogu manualno definirati dodavanjem *launch.json*[26] datoteke u korijenski direktorij projekta aplikacije.

```
"version": "0.2.0",
    "configurations": [
        {
            "name": "my_calculator",
            "request": "launch",
            "type": "dart"
        },
        {
            "name": "my_calculator (profile mode)",
            "request": "launch",
            "type": "dart",
            "flutterMode": "profile"
        }
    ]
}
```

Slika 24. launch.json kod generiran za dotičnu aplikaciju.

[26] JSON je slično YAML-u nezavisan jezik za formatiranje podataka unutar jednostavnih tekstualnih dokumenata.

Pokrenuta u virtualnoj okolini, konačna aplikacija kalkulatora sagrađena za svrhe ovog rada izgleda ovako[27]:



Slika 25. Aplikacija čiji je nastanak prethodno opisan u radu, pokrenuta u testnom načinu na virtualnom uređaju. Obrada autora.

[27] Aplikacija je pokrenuta u virtualnoj verziji Google Pixel 3a uređaja putem android-x86 emulacijskog sustava.

5. ZAKLJUČAK

S obzirom na to da je prikazana aplikacija od strane početnika napisana gotovo samostalno u roku od otprilike sat vremena, jasno je da je Flutter moćan alat, a Dart veoma fleksibilan jezik. Iako je Dart vrlo lagan za naučiti za one koji imaju bilo kakvo osnovno poznavanje nekog od popularnih objektno orijentiranih programskih jezika, Flutter u njega unosi *widget*-e kao koncept, te se oni mogu pokazati kao blaga poteškoća pri navikavanju na ovaj novi tok rada. U konačnici, najveća pomoć pri učenju i eksperimentiranju u ovom slučaju bio je *IntelliSense* alat *VS Code*-a, koji pruža niz naprednih funkcija omogućenih zahvaljujući strojnom učenju i umjetnoj inteligenciji koje Microsoft koristi u razvoju tog alata za *Visual Studio* inačice.
IntelliSense je čak i za ovako nov programski jezik i skup alata bio sposoban ponuditi opise svih tipova *widget*-a, što je posebno korisno za one koji se koriste samo za mijenjanje izgleda i organizaciju ostalih *widget*-a.
Vrijedno je i napomenuti kako je konačan kod bio iznenađujuće čitak i organiziran, bez obzira na količinu nivoa ugniježđenja između svih brojnih *widget*-a. Naravno da je za to djelomično zaslužan i sam IDE i njegove funkcije, no Dart se pokazao kao iznimno sažet i razumljiv programski jezik, upravo dovoljno blizak prirodnom jeziku da je na prvi pogled čitljiv, ali i dovoljno udaljen od njega da nije preopširan.
Podrška *open source* zajednice za Flutter također nije na odmet. Za mnogo funkcionalnosti koje bi početnik proveo sate pokušavajući samostalno sastaviti ili razumjeti, postojali su lako brzo dostupni paketi spremni za umetanje u bilo koju aplikaciju kroz samo nekoliko klikova i jednom linijom koda. Bilo koje greške koje IDE nije mogao adekvatno razumljivo opisati, zajednica je na forumima poput *Stack Exchange*-a[28] podrobno opisala i dokumentirala. Čak je i trenutna javna grana Flutter-a vrlo dostupna općim korisnicima putem *GitHub*-a[29], gdje je lako dokumentirati i objaviti bilo kakve pronađene greške kako bi ih programeri izvornog koda ispravili.
Sad već često spomenuta dokumentacija se također pokazala iznimno bitnom u zajednici Flutter programera. Apsolutno svaki primjer ili komad literature bio je dubinski komentiran i nadopunjen dodatnom dokumentacijom, i to ne samo onom koju *dartdoc* automatski generira, iako se i ona pokazala impresivnom.

[28] *Online* forum posvećen programiranju, sa naglaskom na suradnju članova.
[29] Javni internetski repozitorij koda i aplikacija sa aspektima društvene mreže, jedno od središta *open source* zajednica za međusobno dijeljenje rada.

Zaključno, Flutter je izvrstan primjer činjenice da tehnološki napredak leži u rukama *open source* zajednice. Iako je započet od strane Google-a, zajednica je vrlo brzo prihvatila i praktički preuzela razvoj Flutter-a. Zahvaljujući tome on je danas jedan od daleko korisnicima najprilagođenijih programskih alata. Njegova pristupačnost i kvaliteta očiti su iz njegovog svakodnevno rastućeg udjela u tržištu ne samo mobilnih već i web aplikacija, a još uvijek je nezamislivo što se sve još može postići ovako moćnim i stabilnim alatima.

LITERATURA

Zaccagnino, C. (2020). Programming Flutter. *The Pragmatic Programmers*

Agrawal A., Agrawal A., Arya R., Jain H., Manoorkar J. (2021). Comparison of Flutter with Other Development Platforms. *IJRCT*. Dohvaćeno sa *ijrct.org*

Windmill, E. (2020). Flutter in Action. *Manning Publications Co.*

Sinha S. (2020). Quick Start Guide to Dart Programming. *Apress*

Biessek, A. (2020). Flutter for Beginners. *Packt*

Freitas, E. (2019.) Flutter Succinctly. *Syncfusion, Inc.*

PRILOZI