Prednosti i nedostaci razvoja višeplatformskih aplikacija

Krušlin, Antonio

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Prednosti i nedostaci razvoja višeplatformskih aplikacija

Diplomski rad

Antonio Krušlin

Osijek, 2022.

SADRŽAJ

1. UVOD

Razvijanje mobilne, web ili desktop aplikacije nikad nije bio jednostavan posao, nekada je teže odabrati tehnologiju koja odgovara proizvodu, nego li programirati u njoj. Na samome početku razvoja mobilnih aplikacija programeri su imali na izbor samo korištenje jezika ili programskog okvira ovisno o platformi, Android - Java, iOS - Objective-C. Izrada i održavanje dvije baze koda zagorčava život programeru, ali to se promijenilo dolaskom tehnologija za višeplatformski razvoj. Svakih nekoliko godina na tržište se pojavi novi način višeplatformskog razvoja, te danas već postoji dosta takvih tehnologija, a samo neke od njih su Flutter, React Native, Xamarin, Ionic i Cordova. Dolaskom novih načina razvoja mobilnih aplikacija bitno je temeljno provjeriti i evaluirati novonastalu tehnologiju, kako bi se znalo je li ona spremna za korištenje u produkciji. U ovome diplomskom radu usporedit će se nativni način izrade aplikacije s višeplatformskim razvojem. Biti će prikazane prednosti i mane višeplatformskog programiranja, kada i gdje ih koristiti, a kada ih je bolje zaobići i držati se nativnog razvoja aplikacije. Kao primjer bit će napravljena aplikacija za praćenje režijskih troškova u programskom okviru zvanom Flutter, gdje će biti opisana njezina implementacija. Pošto je tehnologija relativno mlada po pitanju produkcije bitno je naglasiti da će se koristiti trenutne najbolje prakse razvoja.

1.1. Zadatak diplomskog rada

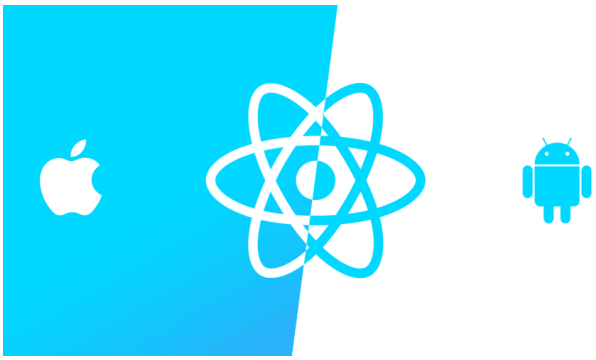
Istražiti alate za razvoj višeplatformskih aplikacija te u jednom od njih (Flutter) izraditi primjer višeplatformske aplikacije. Opisati proces razvoja višeplatformske aplikacije te ga usporediti s nativnim načinom. Na razvijenom primjeru pokazati prednosti i nedostatke razvoja višeplatformskih u odnosu na nativne aplikacije za specifičnu platformu.

2. PREGLED PODRUČJA TEME

U ovom će se poglavlju prikazati trenutno stanje na tržištu po pitanju višeplatformskih tehnologija, gdje će se svaka tehnologija ukratko opisati i navesti u kojim se sektorima najviše koriste i usporedba višeplatformskog tržišta.

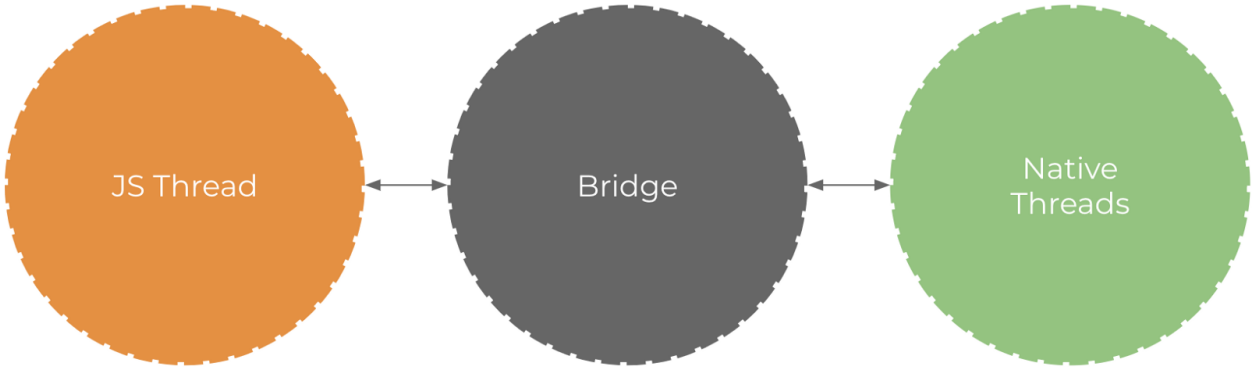
2.1. React Native

React Native razvojni je okvir u programskom jeziku zvanom JavaScript, otvorenog koda gdje svatko može dati svoj doprinos zajednici rješavajući probleme. Koristi za razvijanje mobilnih aplikacija za Android i iOS platforme od 2015. godine, a baziran je na programskom okviru zvanom React razvijen od strane Facebook-a koji se koristi za izradu web stranica [1].



Sl. 2.1. React Native logo [2]

React Native koristi takozvani most koji služi kao spona između procesnih niti JavaScripta i nativnog dijela. Procesne niti su niti u različitim tehnologijama, ali nudi dvosmjernu i asinkronu komunikaciju između njih [3].



Sl. 2.2. React Native bridge – komunikacija [4]

Veliki broj poznatih aplikacija je napravljen u React Native-u, poput Facebook-a, Instagram, Tesla, Skype, Walmart, Airbnb, SoundCloud Pulse, Uber Eats [5].

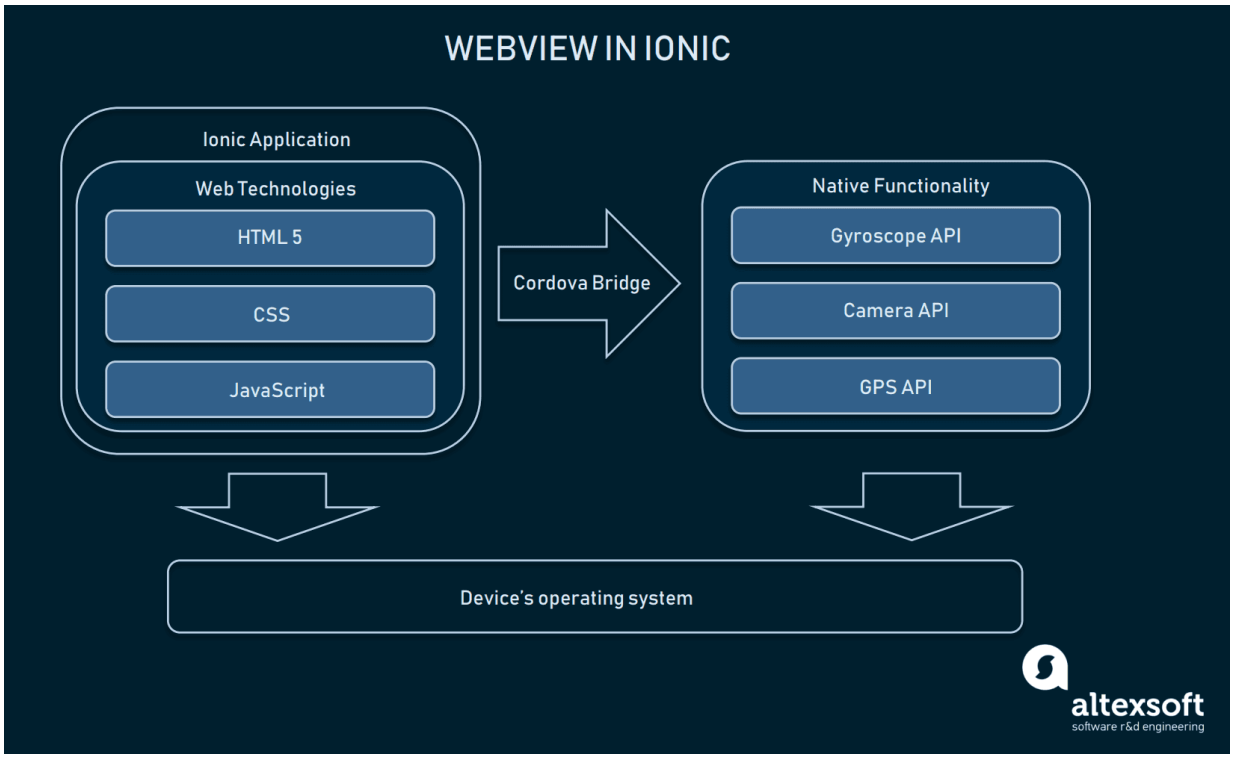
2.2. Xamarin

Programska platforma nastala 2013. godine, otvorenog je koda i u vlasništvu Microsofta od 2016. godine. Programski jezik koji se koristi za razvoj aplikacija je C#, gdje se omogućava iskorištavanje već postojećeg koda. Kompajliranje se događa nativno što znači da je bio glavna opcija za razvoj aplikacije s visokim performansama, prema tome aplikacije su imale nativne performanse, ali i izgled [6]. Iako je Microsoft akvizirao Xamarin, veliki preokret se dogodio 2019. godine kada se Microsoft odlučuje prebaciti veliku većinu aplikacija u React Native [7]. Koriste ga aplikacije poput The Postage, UPS, Alaska Airlanes, Aggreko, HCL, BBC Good Food, Just Giving [8]. Xamarin.Forms će nestati odnosno bit će dobiti spojen s .NET i nastat će nova platforma zvana MAUI.

Sl. 2.3. Xamarin logo [9]

2.3. Ionic

Razvojno okruženje otvorenog koda razvijeno 2013. godine za hibridne mobilne aplikacije. Ionic koristi tehnologije poput React, JavaScript, Angluar i Vue. Koristeći navedene web tehnologije Ionic omogućava izgradnju višeplatformske mobilne aplikacije. Cordova most spaja Ionic aplikaciju s nativnim funkcionalnostima, a u njemu se nalaze dodatci za korištenje kamere mobitela, žiroskop i ostalih senzora prikazano na slici 2.4. [10].



Slika 2.4. Prikaz komunikacije Ionic aplikacije [10]

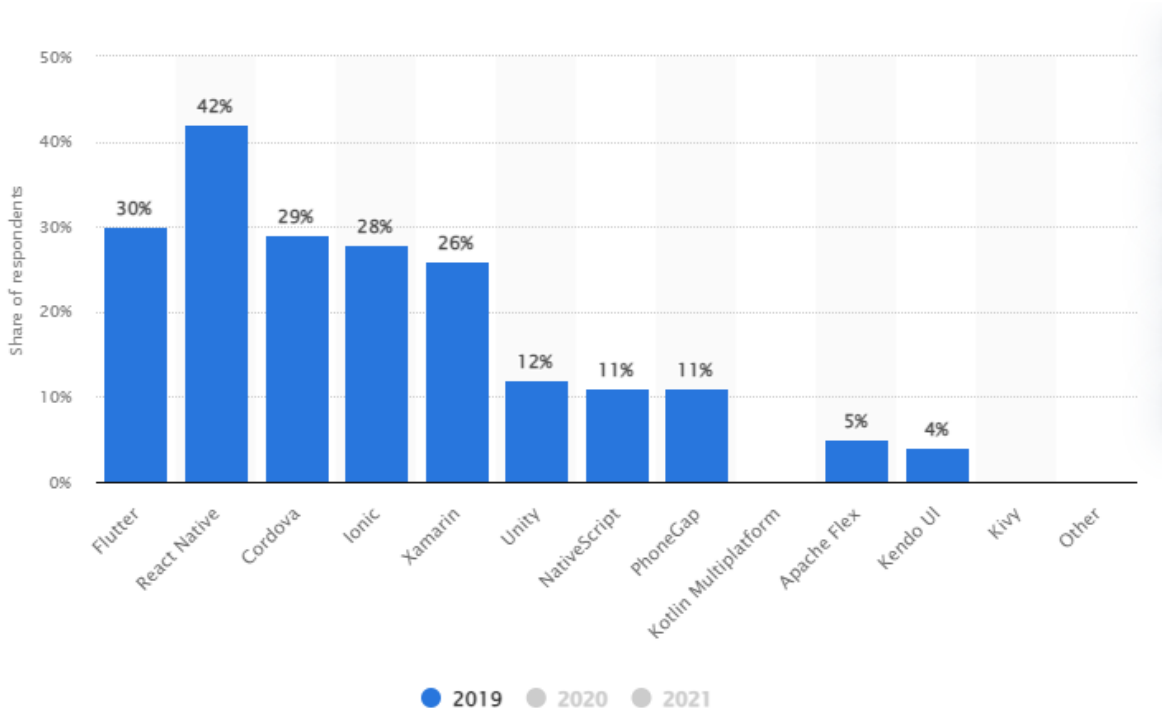
2.4. Apache Cordova

Razvojni okvir za izradu mobilnih aplikacija nastao je 2009. godine od tvrtke Nitobi. Omogućio je razvojnim programerima da naprave hibridnu mobilnu aplikaciju koristeći HTML, CSS i JavaScript. Hibridna aplikacija jer nije prava nativna aplikacija zbog toga što ne koristi nativne komponente, nego web preglednik, a niti je web orijentirana jer je zapakirana tako da ima pristup nativnom API-u mobilnog uređaja. Zbog nuspojava takvog načina izgradnje, aplikacije su imale lošije performanse od nativnih te su znale biti odbijene od strane trgovina aplikacija [11].

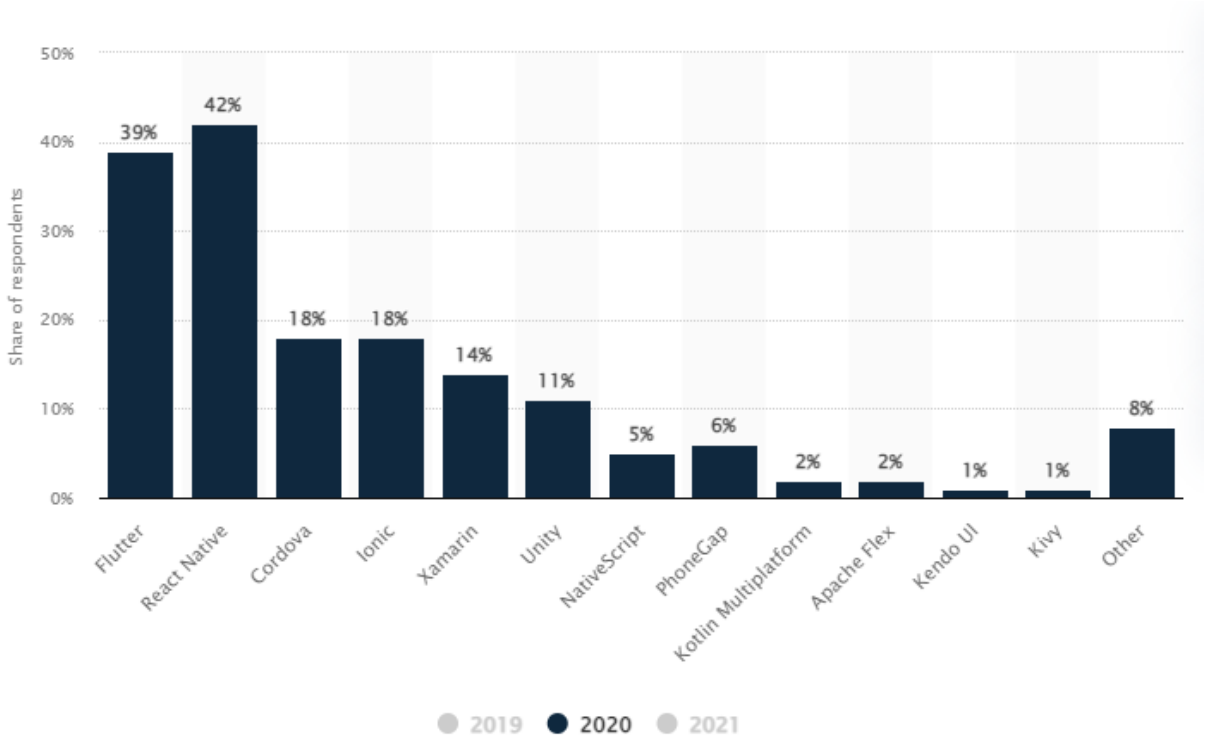
Sl. 2.5. Apache Cordova logo [12]

2.5. Trenutno stanje na tržištu

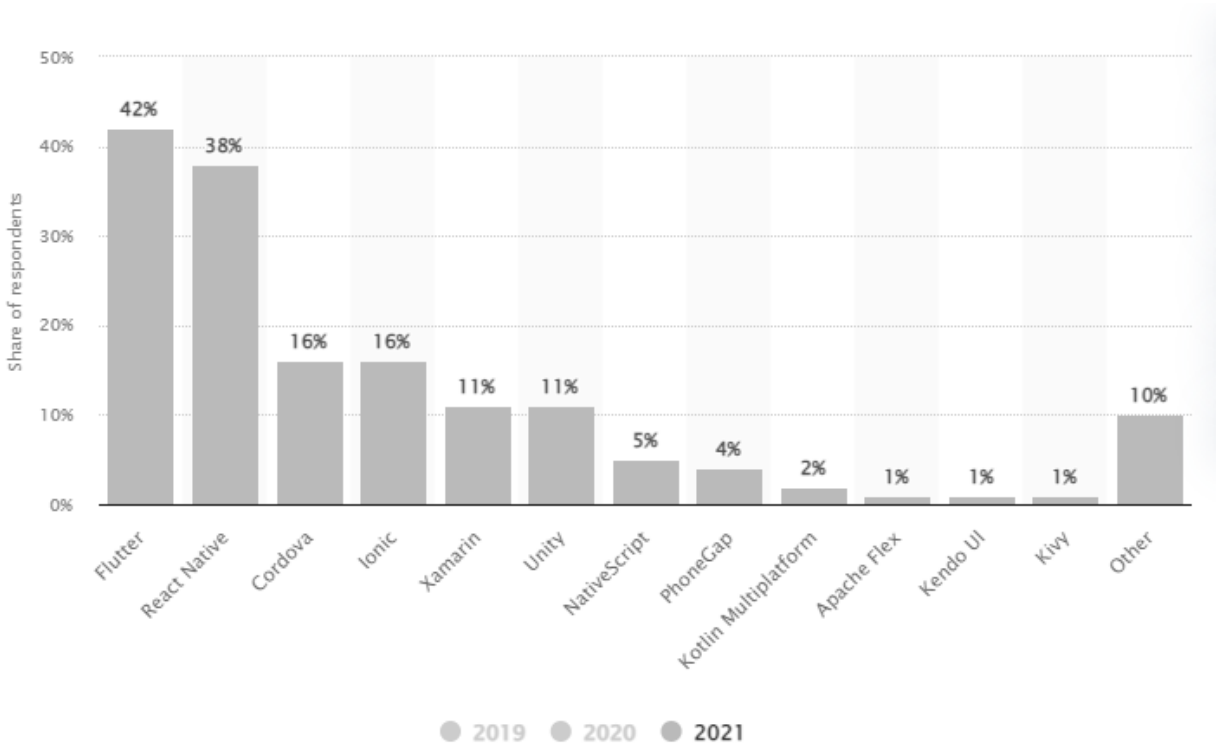
Dolaskom novih hibridnih tehnologija poput Flutter-a dosta starijih tehnologija poput Xamarina, Cordova i Ionica izgubili su određeni postotak programera koju su radili u tom razvojnom okviru.



Sl. 2.6. Višeplatformski razvojni okviri korišteni od strane programera za 2019. godinu [13]



Sl. 2.7. Višeplatformski razvojni okviri korišteni od strane programera za 2020. godinu [13]



Sl. 2.8. Višeplatformski razvojni okviri korišteni od strane programera za 2021. godinu [13]

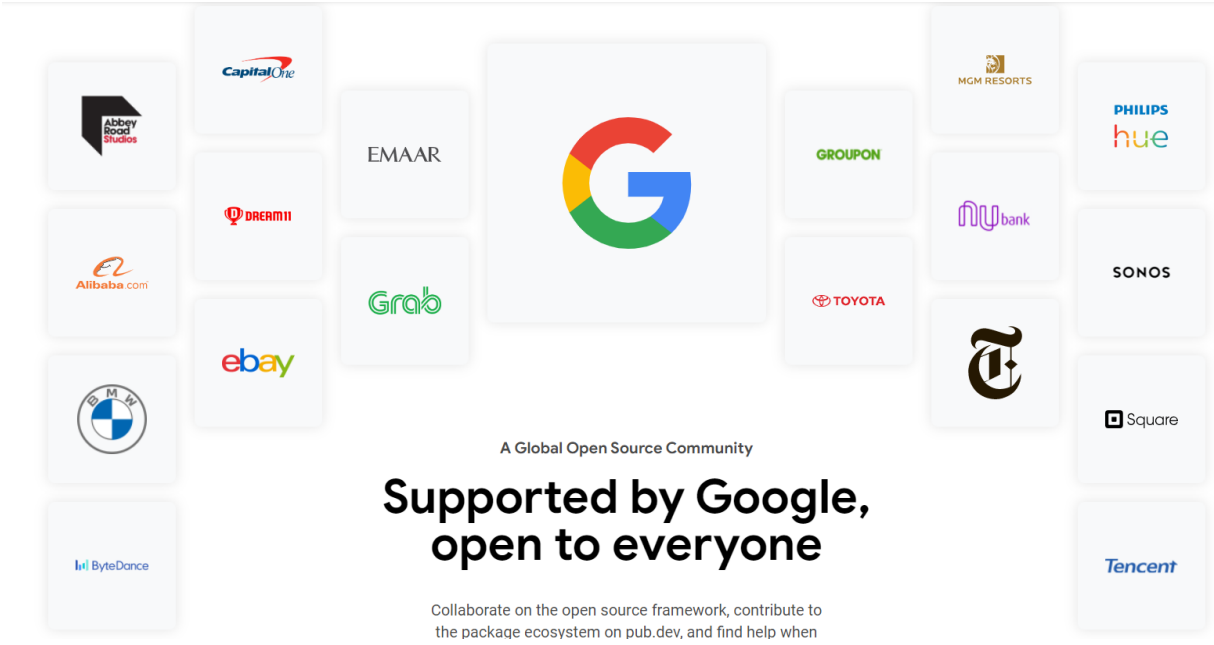
Iz gore navedena tri grafa može se primijetiti da je Flutter u zadnje dvije godine preuzeo veliki broj programera ostalih višeplatformskih razvojnih okvira. Podaci i grafovi su javno dostupni na stranici Statista [13].

3. FLUTTER

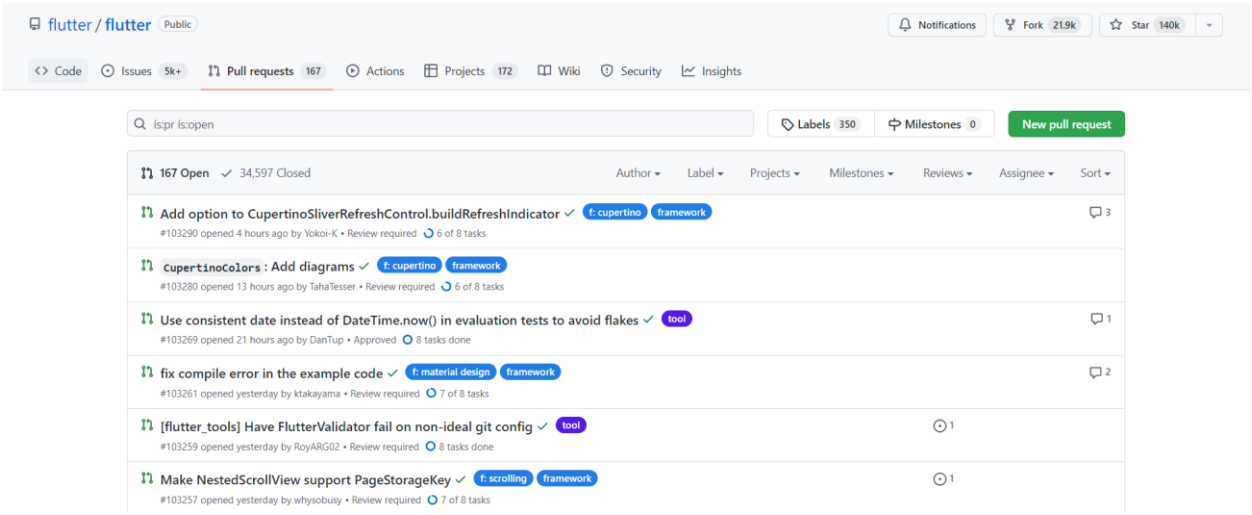
Flutter je razvojni okvir otvorenog koda razvijen od strane Google-a. Jezik u kojem se aplikacije razvijaju je Dart, a prva verzija odnosno alfa verzija izašla je sredinom 2017. godine. Do danas postao je jedan od dominantnijih razvojnih okvira u kategoriji višeplatformskog razvoja, zbog velikog broja pokrivenih platformi kao što su iOS, Android, Web, Mac, Windows, Linux. Veliki broj programera daje svoj doprinos platformi tako što izrađuju vlastite pakete ili pak rješavanjem problema vezanih za razvojni okvir. Na stranici za pakete mogu se pronaći razni dodaci koji uvelike olakšavaju samu izradu aplikacije, većina je besplatna, ali postoje i paketi za koje je potrebno imati plaćenu licencu. Paketi koji zahtijevaju plaćenu licencu su razvijeni od strane većih kompanija dok su besplatni paketi većinom izrađeni i održavani od grupe nezavisnih programera.

3.1. Osnovne informacije

Prva stabilna verzija izašla je krajem 2018. godine, a kao što se može vidjeti na slici 2.6. tijekom prve dvije godine od izlaska stabilne verzije, Flutter je zahvatio dobar dio hibridnog mobilnog tržišta najviše zbog dobrih performansi, stalnog nadograđivanja i velike podrške zajednice. Sve više velikih kompanija (Slika 3.1.) prelazi na korištenje Flutter-a od kojih su Times, BMW, Toyota, Alibaba.

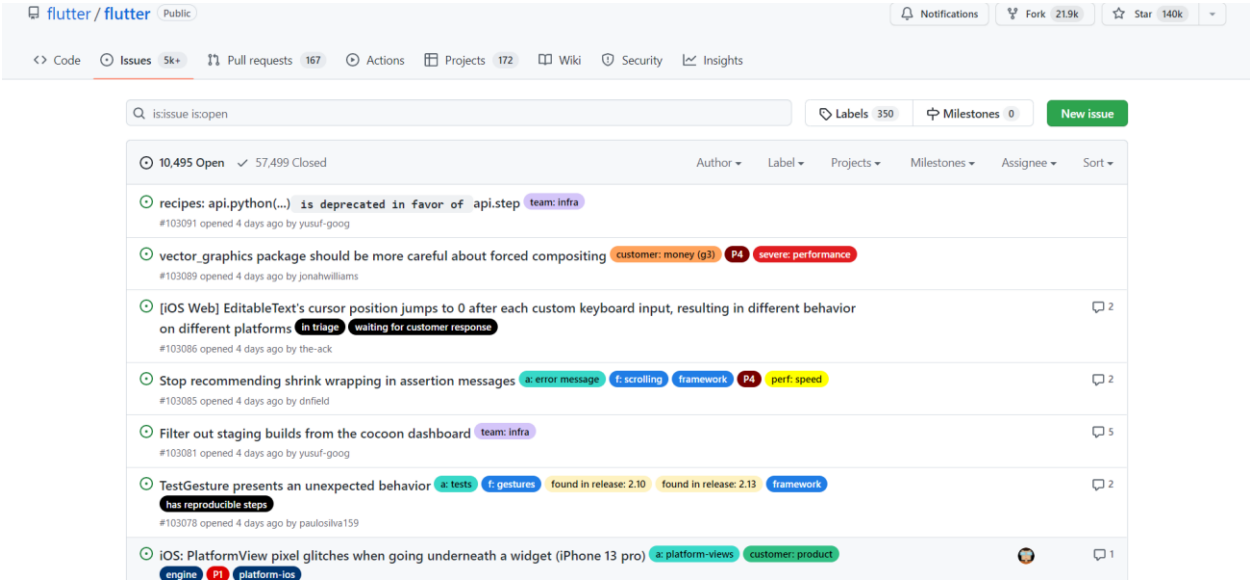


Sl. 3.1. Prikaz kompanija koje koriste Flutter [14]



Sl. 3.2. Flutter GitHub stranica sa zahtjevima za poboljšanje

Prikaz zdrave zajednice je aktivnost njezinih članova, a na slici 3.2. prikazani su samo neki zahtjevi za poboljšanje gdje su korisnici ispravili greške ili nedostatke razvojnog okvira kako bi ga učinili boljim za korištenje. Doprinos svake osobe je bitan, od pronalaska greške pa do njezinog rješavanja. Takav način najbolje prikazuje koliko se korisnici brinu o budućnosti razvojnog okvira.



Sl. 3.3. Flutter GitHub stranica za probleme

Zbog velikog broja platformi ima dosta prijavljenih problema, gdje svaki od njih je prioritiziran i podijeljen po platformi.

3.2. Dart

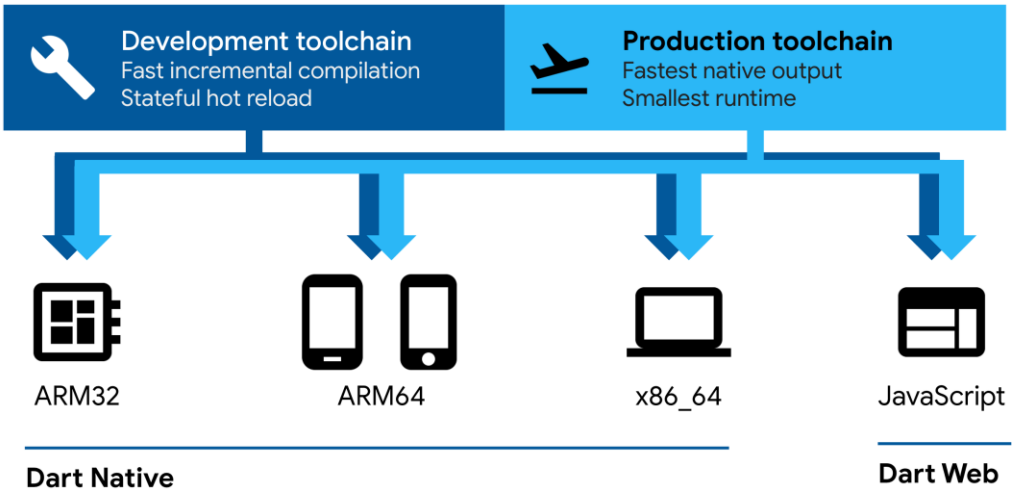
Programski jezik razvijen od strane kompanije Google, koristi se za izradu mobilnih, web i desktop aplikacija pa čak i za izgradnju servera. Nastao je kao projekt u 10. mjesecu 2011. godine, a korisnicima je postao dostupan za korištenje u 11. mjesecu 2013. godine. Cilj im je bio napraviti optimizirani jezik za razvoj brzih aplikacija na različitim platformama [15].

3.2.1. Jezik

Dizajniran je za klijentski način razvoja, prioritiziranjem stabilnog razvoja i visoke kvalitete produkcijskog iskustva na raznim platformama. Programski jezik Dart temelj je razvojnog okvira Flutter, a jezik podržava razne osnovne zadatke razvojnog programera kao što su formatiranje, analiza i testiranje koda. Dart ima veliki spektar biblioteka koje olakšavaju razvoj svakodnevnih zadataka, a to su razne biblioteke poput biblioteke za osnovne funkcionalnosti, kodiranje i dekodiranje sadržaja, matematičke konstante, komunikaciju s drugim uređajima, podrška za asinkrono programiranje, istovremeno izvršavanje korištenjem izolata. Postoje i paketi napravljeni od strane trećih izdavača i velika zajednica koja nudi podršku za razne značajke [16]. Izgledom podsjeća na kombinaciju C# jezika i skriptnog jezika poput Pythona ili JavaScripta, a zbog jednostavne sintakse vrlo brzo se nauči [17, str. 3]. Dart nudi sigurnost i prikazivanje pogrešaka prilikom kompajliranja, strukturu, preciznost, brzinu poput C#, a dok s druge strane nudi mogućnost da jedan jezik bude dovoljan za izradu pozadinskog servisa, mobilne, web ili pak desktop aplikacije [18].

3.2.2. Izvođenje po platformama

Dart dopušta da se kod izvršava na različite načine, te jedna od njih je izvršavanje za nativne platforme kao što su aplikacije za mobilne i desktop uređaje. Dart također sadrži Dart virtualni uređaj za kompajliranje u trenutku i prijevremeno kompajliranje za izradu strojnog koda [19, str. 22]. Kod web platforme gdje su ciljane aplikacije za web, Dart sadrži kompajler u vremenu razvoja (dartdevc) i u vremenu produkcije (dart2js) gdje oba kompajlera prevode Dart u JavaScript kao što je prikazano na slici 3.4. [16].



Slika 3.4. Kompajliranje Dart koda [16]

3.3. Proces razvoja

Zbog velike pokrivenosti platformi, kod se može lako iskoristiti čak i da se radi na drugačijoj platformi. Iskorištavanje već postojećeg koda iz mobilne aplikacije lako se iskoristi za izradu internetske stranice. Jedino treba paziti na pakete koji su korišteni u projektu jesu li kompatibilni za obje platforme. Ako nisu kompatibilne onda treba pronaći odgovarajuću zamjenu. Također bitno je imati na umu koje sve značajke aplikacija sadrži kako bi se mogli pronaći mogući rizici prilikom razvoja, ali i kasnije nadogradnje aplikacije.

3.4. Razvojni alati

Odabir razvojnog okruženja najviše ovisi o pozadini programera koji se prvi put upoznaje s Flutter razvojnim okvirom. Najpopularniji uređivači koda su Android Studio i Visual Studio Code.

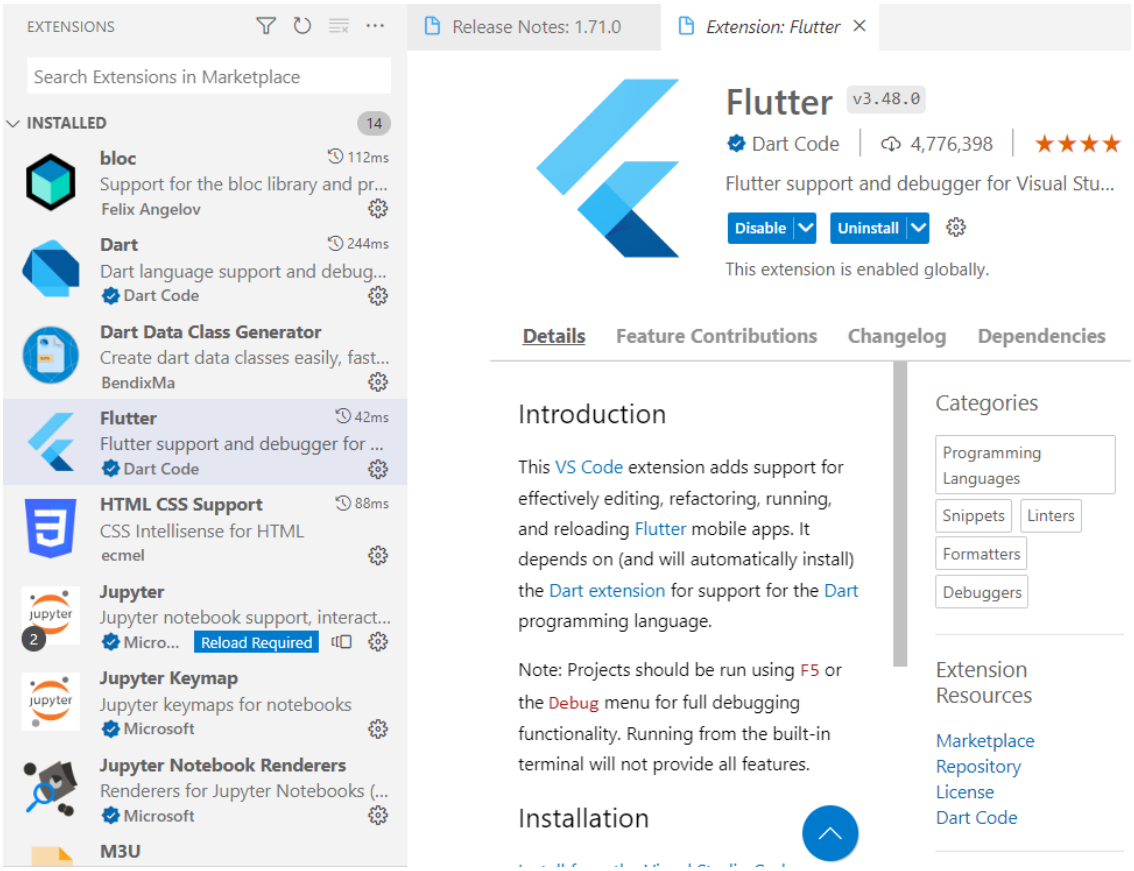
3.4.1. Android studio

Programeri koji prelaze s Android platforme na Flutter mogu nastaviti koristiti Android Studio bez ikakvih problema, što im olakšava prelazak jer ne moraju instalirati nikakve dodatne programe niti se privikavati na nepoznatu radnu okolinu. Android studio nudi veliki broj funkcionalnosti na koje su Android programeri navikli što ga čini glomaznijim, ali i kompleksnijim za nove korisnike.

3.4.2. Visual studio Code

Velika većina novih Flutter programera koristi Visual Studio Code baš zbog njegove jednostavnosti i visokih performansi. Jako je lako prilagoditi uređivač koda prema korisnikovim

željama jer nudi veliki izbor tema za sučelje i paketa kao na slici 3.5. Ovaj uređivač koda koristi se za razvoj u raznim drugim jezicima kao što su JavaScript, Python, TypeScript, HTML/CSS, PHP.

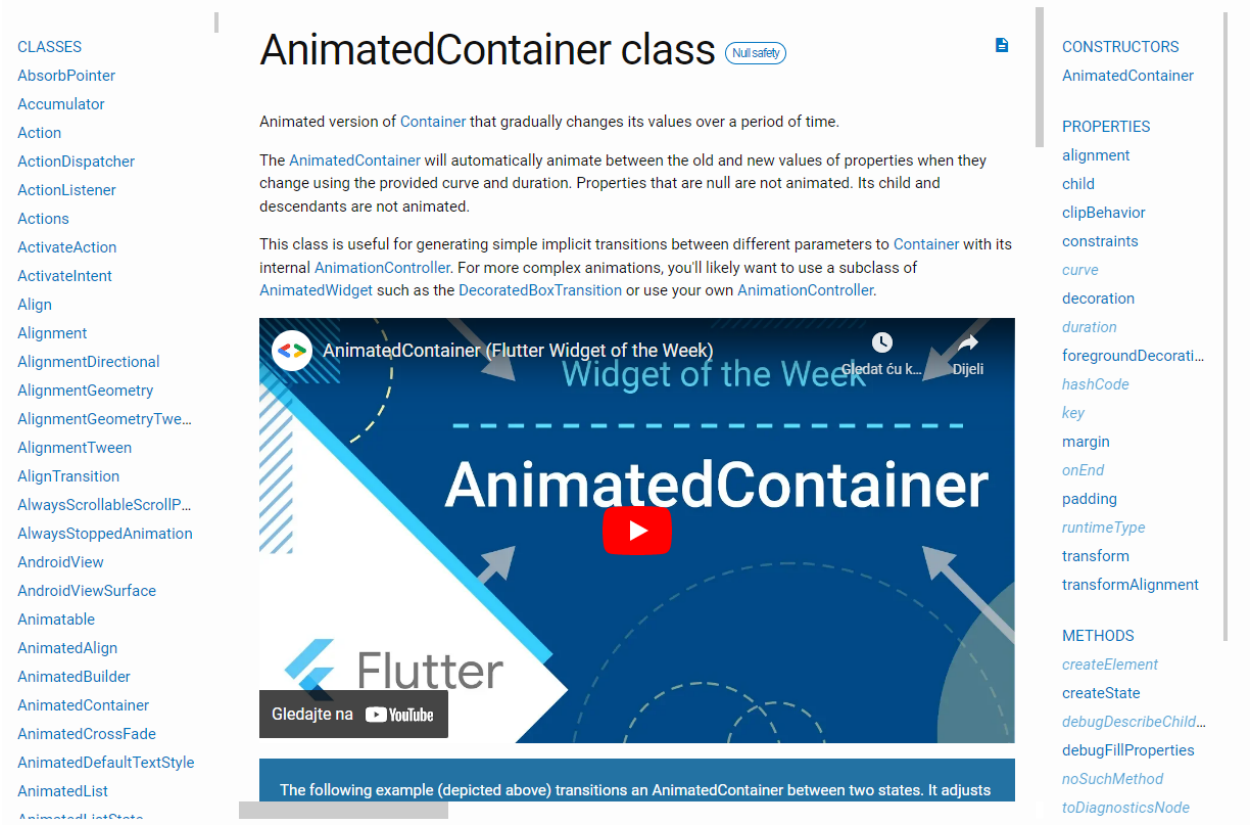


Sl. 3.5. Izbornik paketa u Visual Studio Code uređivaču

3.5. Osnovni elementi

Jedna od ključnih značajki Flutter razvojnog okvira su njegovi već napravljeni elementi. Flutter okvir dolazi s velikim brojem već napravljenih elemenata spremnih za korištenje i jednostavno uređivanje. Opća podjela elemenata je na elemente sa stanjima i elemente bez stanja, razlika je u tome što elementi sa stanjem imaju mogućnost ponovnog prikazivanja prilikom promjene stanja. Takvi elementi utječu na performanse aplikacije i stoga je bitno znati odrediti koji elementi trebaju imati stanje, a koji ne, kako bi se izbjegla prekomjerna ponovna izgradnja elementa na zaslonu. U tom slučaju aplikacija će tražiti veću količinu radne memorije kako bi zadovoljila trenutne potrebe, a u slučaju kada je aplikacija ograničena radnom memorijom (npr. mobitel) može doći do trzanja, zaleđivanja ili u najgorem slučaju dovesti do rušenja aplikacije. Vrlo je bitno upoznati razne elemente koje Flutter okvir nudi, kako bi se spriječio scenarij izrade elementa koji već postoji u biblioteci osnovnih Flutter elemenata. Poznavanjem svih dostupnih alata Flutter razvojnog okvira

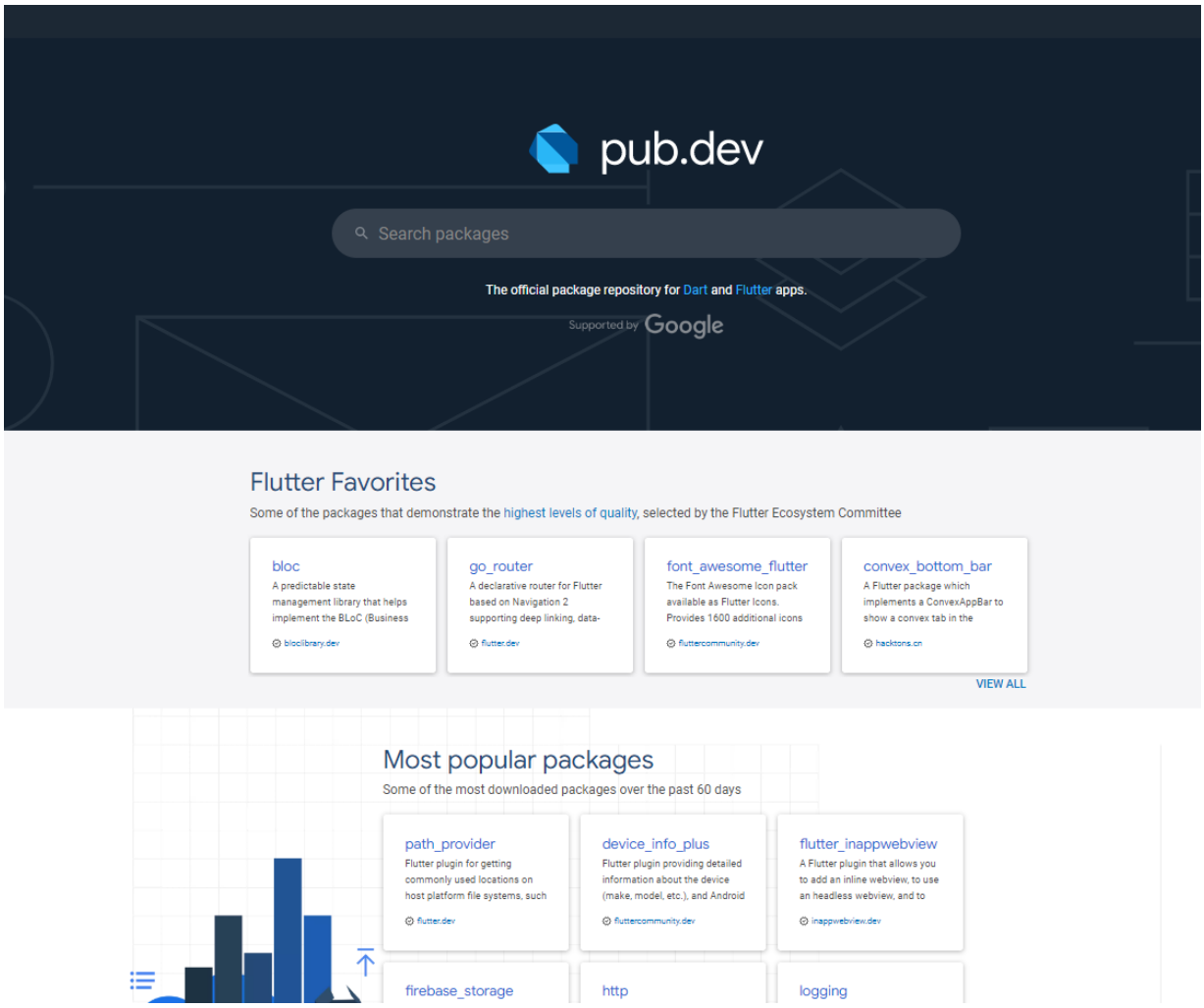
lakše je procijeniti koliko je teško izraditi neku funkcionalnost u aplikaciji. Na službenoj stranici Flutter okvira moguće je pronaći katalog koji sadrži velik broj elemenata gdje su opisane njihove funkcionalnosti i način uporabe. Također, tu su ponuđeni i video zapisi koji ukratko opisuju način korištenja elementa, njegove prednosti i mane kao što se može vidjeti na slici 3.6. [20].



Slika 3.6. Detalji korištenja animiranog kontejnera [21]

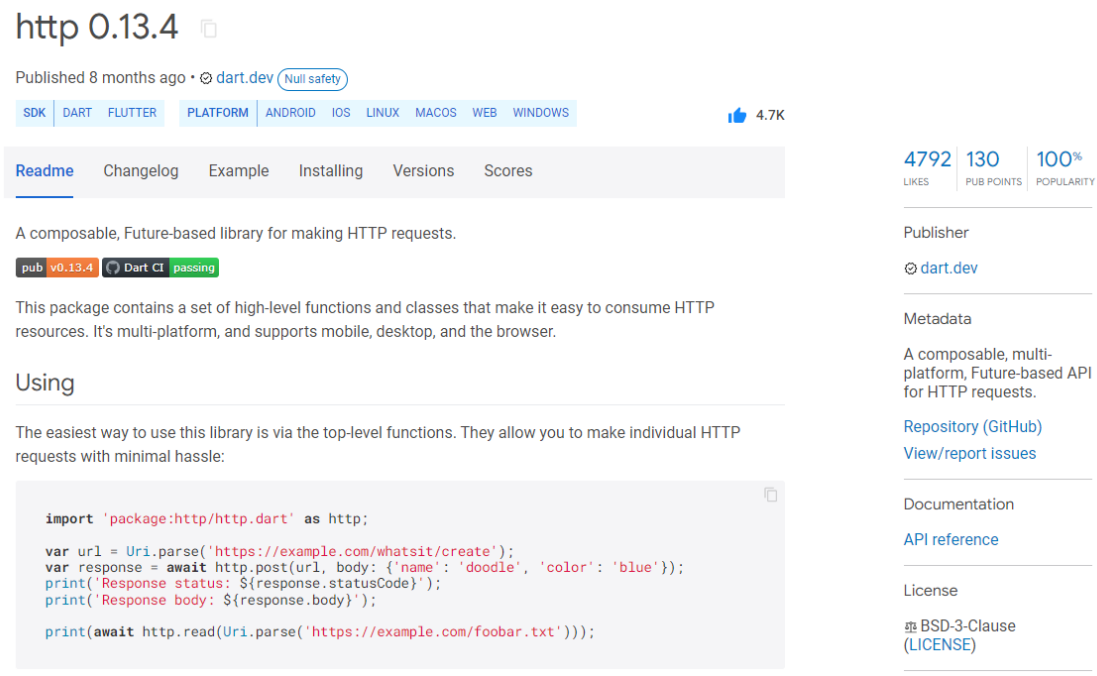
3.6. Paketi

Jedan od glavnih prednosti Flutter-a su besplatni paketi koji pomažu prilikom izrade aplikacije. Paketi su dostupni svima za korištenje, većina ih je besplatna dok neki paketi imaju određene uvjete kada je potrebno kupiti licencu za njihovo korištenje. Takvi paketi su uglavnom kvalitetno napravljeni i pouzdani te imaju veliku mogućnost uređivanja.



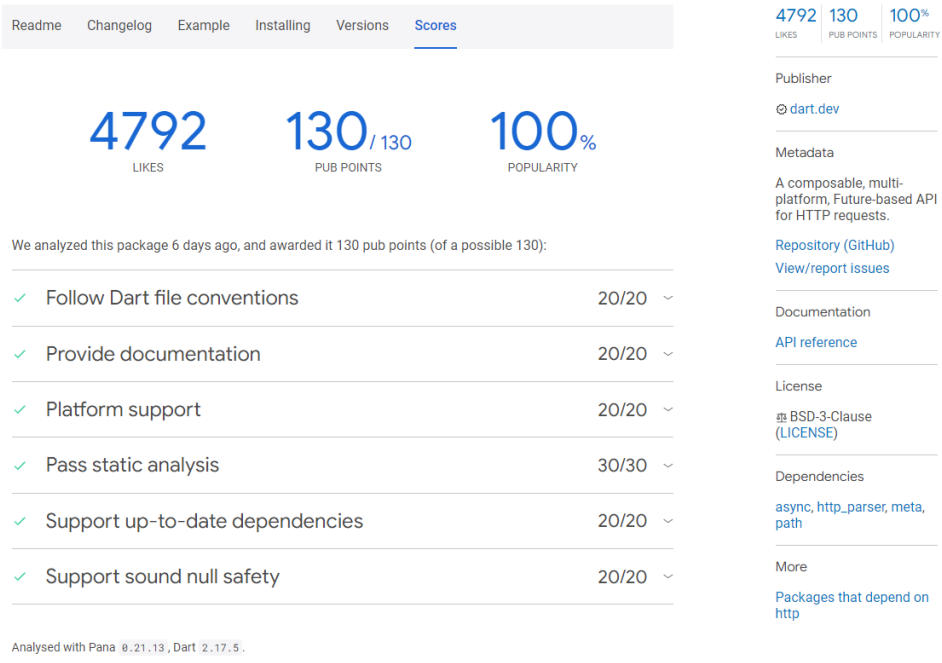
Sl. 3.7. Stranica za pretragu Flutter paketa

Kako je napredovao Flutter tako su napredovali i dostupni paketi, te programer ima mogućnost označiti svaki paket koji im je pomogao i pri tome pokazati ostalim članovima Flutter ekosustava. Na taj način dolazi do filtriranja paketa gdje najbolji paketi završe na vrhu stranice za pretragu paketa kao što je prikazano na slici 3.7. Također postoji mogućnost da se paketi sortiraju s obzirom na platformu jer nisu svi paketi dostupni za sve platforme. Zbog toga bitno je imati u planu koje pakete koristiti ovisno o platformi na koju se aplikacija objavljuje, a ako se gleda na proširivost bitno je izabrati pakete koji su već podržani na velikom broju platformi.



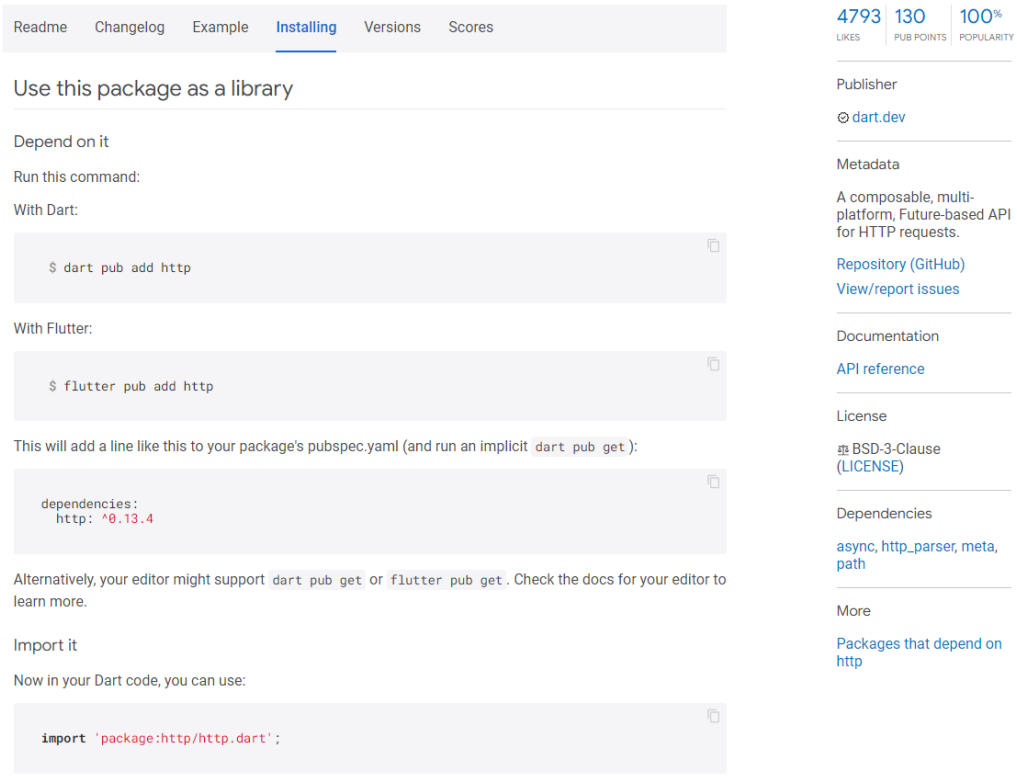
Sl. 3.8. Http paket

Slika 3.8. prikazuje paket koji pomaže u komunikaciji s Http zahtjevima, ovo je klasični prikaz jednog paketa, gdje je na naslovnoj stranici prikazan način upotrebe i najbolje prakse. Slika 3.9 prikazuje način bodovanja gdje je cilj osigurati kvalitetu i sigurnost paketa, popularnost govori koliko se učestalo koristi određeni paket. Vrlo je važno obratiti pažnju na ocjenu prilikom odabira paketa.



Slika 3.9. Ocjenjivanje paketa

Paketi se dodaju u YAML datoteku ručno ili pomoću komandne linije. Upute za instalaciju paketa mogu se pronaći na sekciji za instaliranje paketa kao prema slici 3.10., a važno je pročitati upute zbog toga što neki paketi ovise o drugim paketima tako da je i njih potrebno instalirati kako bi željeni paket dobro radio.



Sl. 3.10. Informacije za instalaciju paketa

Treba naglasiti iako paketi mogu pomoći, ponekad mogu stvoriti probleme pa treba pripaziti pri njihovom odabiru. Paket ne bi trebao biti ovisan o drugim paketima, jer ako jest, onda može doći do problema. Primjerice, ako želimo podignuti verziju paketa A, ali paket B koji je ovisan o njemu trenutno još ne podržava noviju verziju paketa A onda dolazi do sukoba paketa. Ovisno o tome održavanje starijih aplikacija može biti dosta otežano ili pak olakšano, tako da ne treba uvijek uzimati sve pakete, nego pokušati smisliti vlastito rješenje da ne bude ovisno o drugim paketima.

3.7. Flutter verzija 3.0

Dolaskom verzije 3.0 Flutter podržava razvoj aplikacija na svim platformama iOS, Android, web, Windows, macOS, Linux. Ovdje su sve navedene platforme postale stabilni dio Flutter razvojnog okvira. To ne znači da su sve platforme na istoj razini spremnosti za izradu produkcijske aplikacije, ali govori o tome da su sve trenutne platforme ugrađene u sustav i ondje planiraju i ostati. Dolaskom nove verzije došla je podrška za Apple Silicon čip gdje se sada iskorištava puni

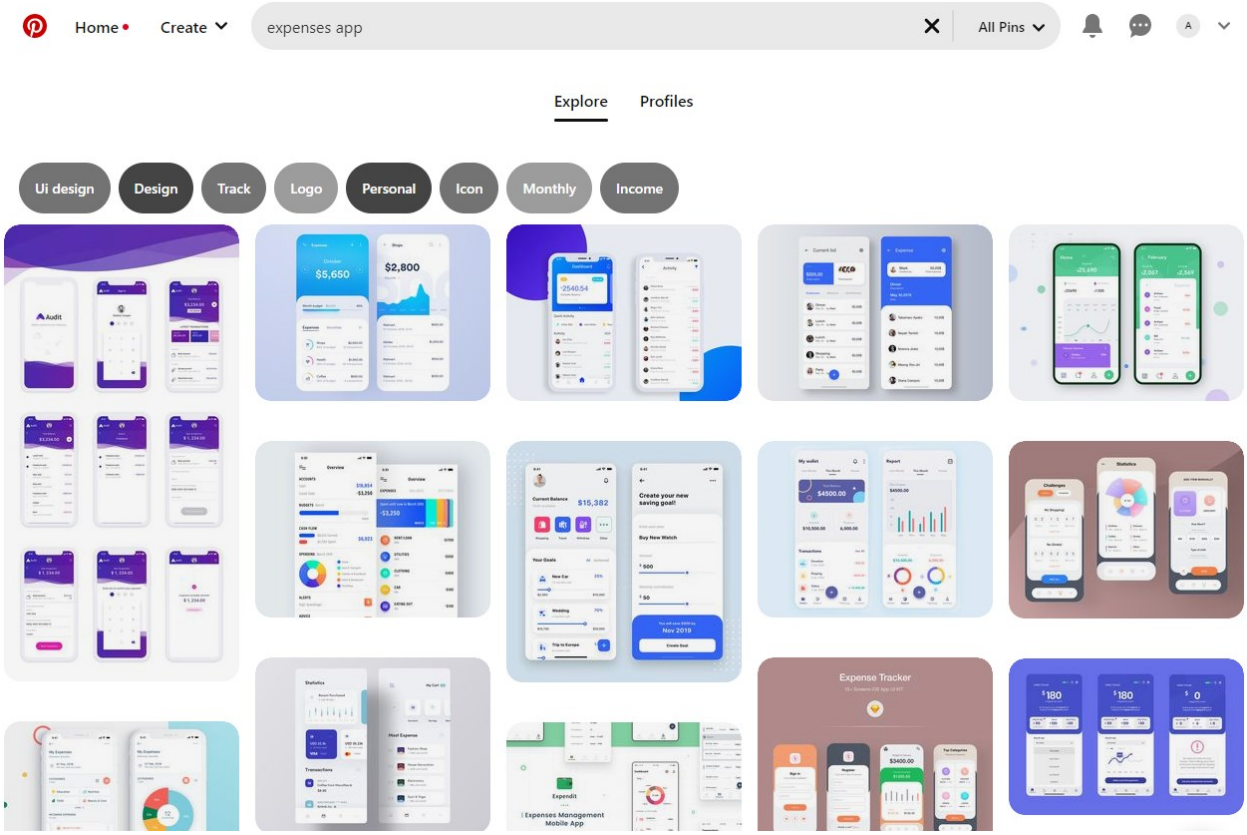
potencijal M1 čipa, a što znači brže kompajliranje aplikacija. Također su došle i razne nadogradnje za već poznate servise kao što su Firebase, Sentry, AppWrite i AWS Amplify. Za većinu programera do sada Flutter je bio većinom razvojni okvir za izradu aplikacija. Nova verzija nudi mogućnost razvoja igara gdje se koriste sve prednosti hardverski ubrzane grafičke podrške [22].

4. PRIMJER IMPLEMENTACIJE APLIKACIJE

Prilikom izrade aplikacije bitno je dobro razraditi sve funkcionalnosti aplikacije kako bi se lako ustanovilo što je sve potrebno, gdje se nalaze mogući blokeri tj. stavke koje zaustavljaju razvoj aplikacije. To može biti od razvojnog okruženja, ograničenja platforme pa sve do poslužitelja usluga treće strane. Nakon što su funkcionalnosti raspisane i provjerene tako da nema nikakvih problema s njihovom implementacijom. Kreće faza postavljanja razvojnog okruženja i razvoj aplikacije.

4.1. Dizajn

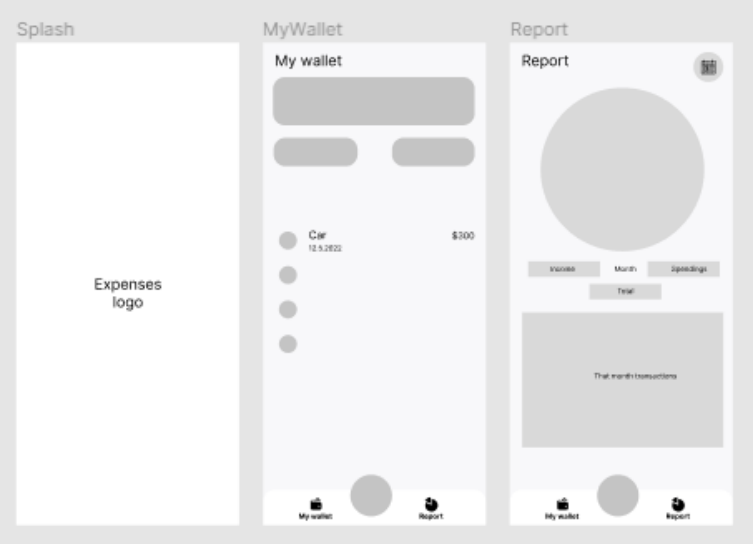
Nakon što su funkcionalnosti aplikacije potvrđene započinje faza dizajniranja aplikacije, koristit će se besplatni program za izradu dizajna koji se zove Figma. Prva faza izrade dizajna jest istraživanje već postojećih rješenja, vrlo je lako pronaći već gotove ideje na stranicama poput Pinterest prikazano na slici 4.1.



Sl. 4.1. Prikaz različitih ideja za dizajn aplikacije [23]

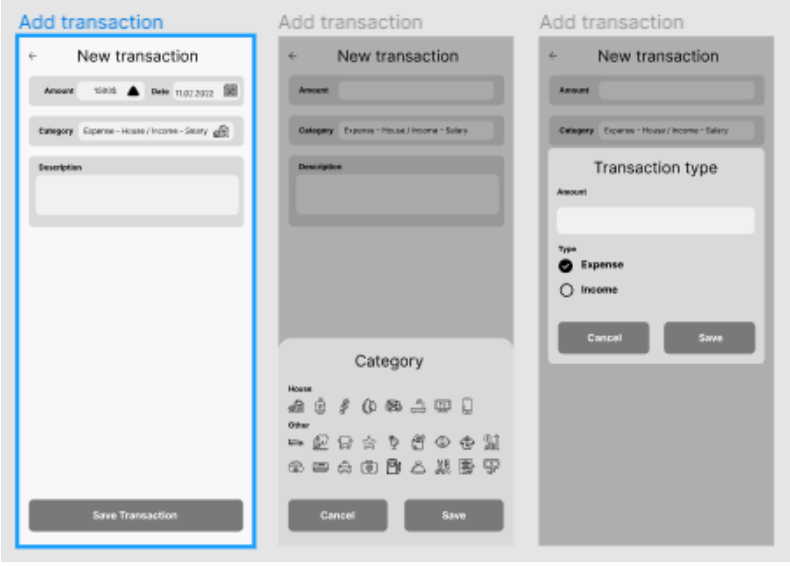
Nakon što se odabere nekoliko ideja koje najbolje odgovaraju zadanim funkcionalnostima započinje faza kreiranja kostura dizajna kao na slici 4.2, gdje se smještaju elementi kako bi

najbolje prikazali funkcionalnosti i učinili ih korisniku aplikacije lako razumljivima. U toj fazi nije bit pronaći boje, nego namjestiti elemente kojima će se tek kasnije detaljno definirati boje, fontovi i ikone.



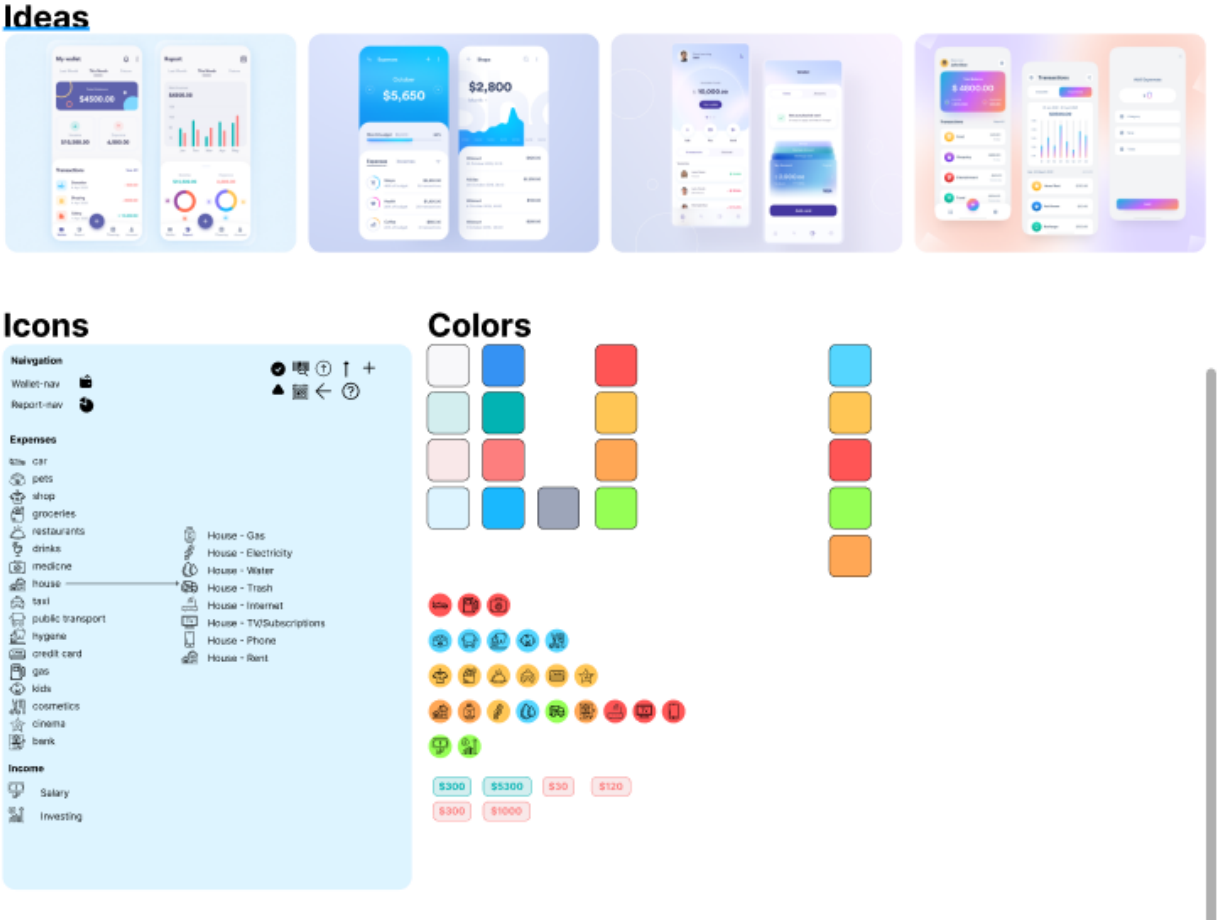
Sl. 4.2. Dizajn kostura aplikacije

Kada je završen kostur svih potrebnih ekrana može početi potraga za elementima poput ikona, boja i fontova. Slika 4.3 prikazuje iduću fazu gdje se nalazi detaljniji dizajn koji pokriva nekoliko stanja jednog ekrana gdje su već ikone definirane.



Sl. 4.3. Dizajn ekrana za dodavanje troška

U ovoj fazi bitno je pronaći odgovarajući stil aplikacije i držati ga se kako bi izgled cijele aplikacije bio konzistentan.



Sl. 4.4. Prikaz Figma ploče s detaljima

Slika 4.4. prikazuje kako izgleda ploča na kojoj se nalaze svi tipovi elemenata od ikona, boja pa do gotovih primjera elemenata.

4.2. Postavljanje razvojnog okruženja

Prva stvar koja je potrebna za početak programiranja jest instalirati razvojni okvir Flutter. To je moguće pomoću komandne linije.

Linija ***Kod***

```
1:      git clone https://github.com/flutter/flutter.git -b stable
```

Sl.4.5. Preuzimanje razvojnog okvira preko komandne linije

Nakon toga treba postaviti razvojno okruženje, a kao što je već ranije spomenuto, trenutno je najpopularniji za uređivanje i izradu koda VS Code, iako se može koristiti i Android Studio. Nakon

toga treba dodati ekstenzije u razvojno okruženje za Dart i Flutter te provjeriti okolinu je li sve uspješno povezano.

Linija ***Kod***

```
1:      flutter doctor
```

Sl.4.6. Provjera Flutter okoline

Pozivanjem gore navedene naredbe korisnik dobiva potpunu sliku svih komponenti.

```
C:\Users\krusl.ANTONIO>flutter doctor

  A new version of Flutter is available!

  To update to the latest version, run "flutter upgrade".

Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.0.1, on Microsoft Windows [Version 10.0.19043.1766], locale en-US)
[!] Android toolchain - develop for Android devices (Android SDK version 33.0.0)
    X cmdline-tools component is missing
      Run `path/to/sdkmanager --install "cmdline-tools;latest"`
      See https://developer.android.com/studio/command-line for more details.
    X Android license status unknown.
      Run `flutter doctor --android-licenses` to accept the SDK licenses.
      See https://flutter.dev/docs/get-started/install/windows#android-setup for more details.
[√] Chrome - develop for the web
[√] Visual Studio - develop for Windows (Visual Studio Community 2019 16.8.3)
[√] Android Studio (version 2021.2)
[√] VS Code (version 1.68.1)
[√] Connected device (3 available)
[√] HTTP Host Availability

! Doctor found issues in 1 category.
```

Sl. 4.7. Ispis Flutter doktora

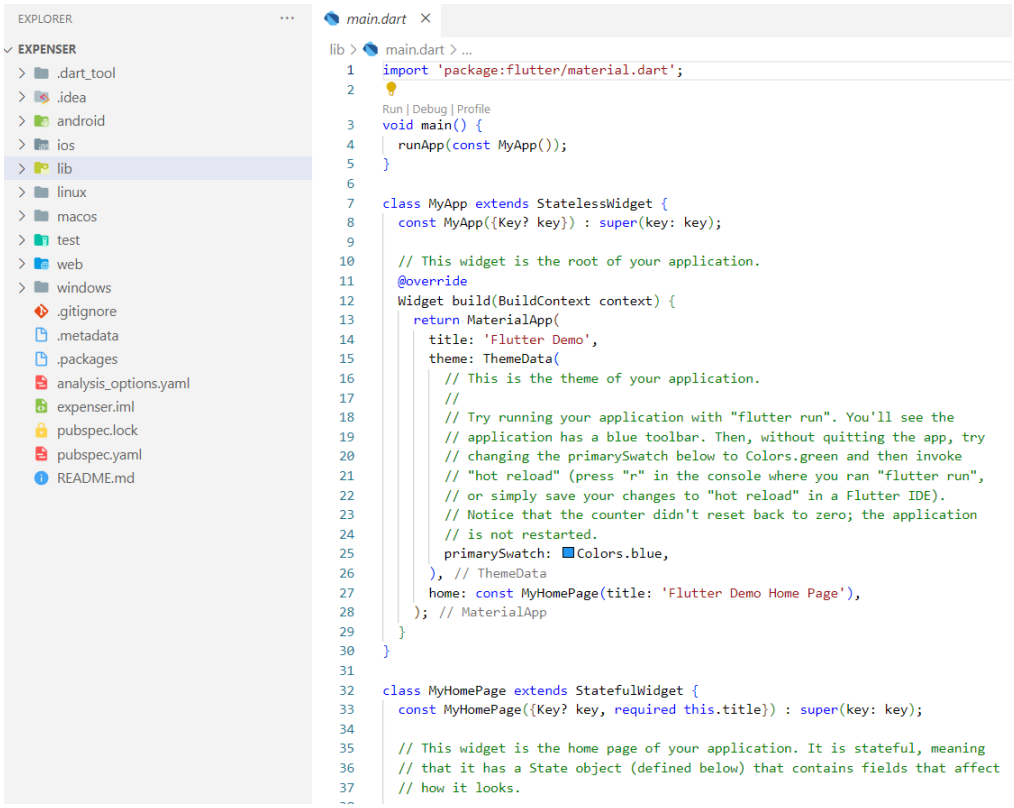
Pomoću toga lagano je provjeriti i otkloniti problem u Flutter okolini. Kao što se može vidjeti na slici 4.7. ako ima problema unutar Flutter okoline ponuđena su i moguća rješenja kako da se otkloni problem. Kada je Flutter okvir postavljen, može se kreirati početna aplikacija pozivom naredbe kao na slici 4.8.

```
krusl@Antonio MINGW64 /d/flutter Playground
$ flutter create expenser
Creating project expenser...
Running "flutter pub get" in expenser...
Wrote 127 files.

All done!
In order to run your application, type:

  $ cd expenser
  $ flutter run

Your application code is in expenser\lib\main.dart.
```

Sl. 4.8. Kreiranje početnog direktorija Flutter aplikacije

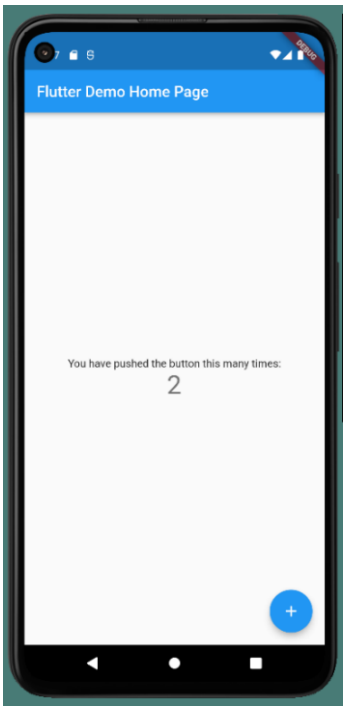
4.3. Pokretanje aplikacije

Nakon što je Flutter kreirao početni direktorij moguće ga je pokrenuti na simulatoru ili pravom uređaju, a to je jedino moguće ako je instaliran Android Studio za pokretanje aplikacije na Android ili Xcode za pokretanje aplikacije na iOS operacijskom sustavu. Gledajući na mobilne platforme na Windows operativnom sustavu moguće je pokrenuti jedino aplikaciju za Android platformu dok kod MacOS moguće je pokrenuti obje platforme Android i iOS.



Slika 4.9. Pregled početnog direktorija

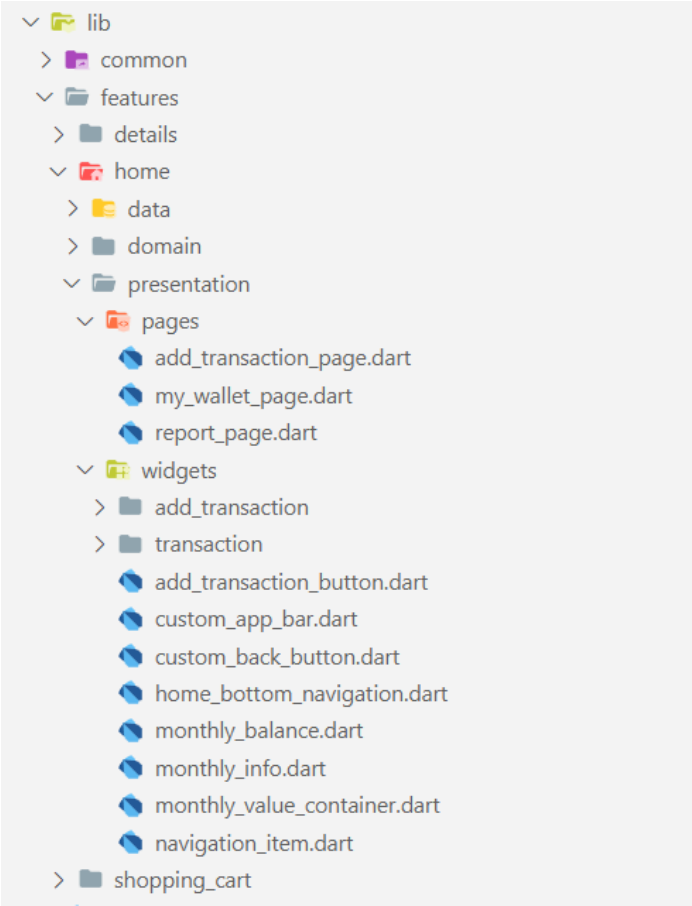
Završetkom instalacije moguće je pokrenuti aplikaciju koja će izgledati kao na slici 4.9. Pokrenuta početna aplikacija (slika 4.10.) prikazuje jednostavni brojač koji nudi osnovni pogled na temeljne komponente Fluttera, ali i strukturu datoteka po platformama gdje svaka platforma ima svoju datoteku.



Slika 4.10. Pokretanje početne aplikacije

4.4. Postavljanje projekta

Što je Flutter duže na svjetskoj pozornici kao jedan od predvodnika višeplatformskog programiranja to je lakše za pronaći razne tipove postavljanja projekta. Od organizacije datoteka pa sve do najboljih projektnih arhitektura. Projekt se kreira preko komandne konzole gdje se upisuje naredba za kreiranje Flutter projekta. Nakon toga bitno je odabrati način na koji će se datoteke raspoređivati i toga se držati do kraja razvoja. Postoje dva načina rasporeda, raspored po funkcionalnostima i po slojevima. Raspored po funkcionalnostima nudi programeru bolju preglednost svih potrebnih datoteka za tu funkcionalnost zbog toga što se sve nalazi na jednom mjestu kao što je prikazano na slici 4.11. Kod podjele po slojevima dolazi do gomilanja datoteka u istu mapu, gdje se kasnije teško snaći, a još teže kada dođe programer koji nije bio od samoga početka izrade projekta [24].



Sl. 4.11. Raspored datoteka po funkcionalnostima

4.5. Pregled aplikacije

U ovom odlomku prikazat će se neki dijelovi aplikacije i objasniti način na koji je napravljeno. Važnost početnog ekrana je da osigura da su svi krucijalni dijelovi aplikacije spremni, kako bi korisnik sa sigurnošću mogao koristiti aplikaciju.

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'package:hive_flutter/hive_flutter.dart';

final futureCreateBoxProvider = FutureProvider<Box<String>>((ref) async {
  await Hive.initFlutter();
  final box = await Hive.openBox<String>('yearBox');
  ref.read(boxProvider.notifier).update((state) => box);
  return box;
});

final boxProvider = StateProvider<Box<String>?>(
  (ref) => null,
);
```

Sl. 4.12. Inicijalizacija Hive baze podataka

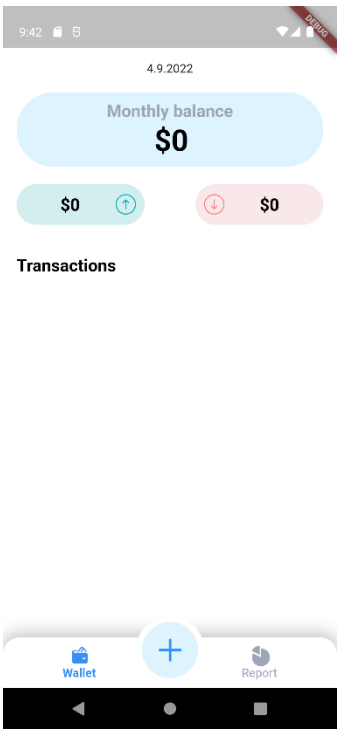
Slika 4.12. prikazuje kod servisa koji je zaslužan za inicijalizaciju baze, dok slika 4.13. prikazuje kako se aplikacija ponaša u različitim slučajevima. Prilikom uključivanja aplikacije prvi zaslon

koji se vidi jest zaslon učitavanja, a ovisno o rezultatu inicijalizacije Hive baze podataka ovisi koji će se zaslon prikazati sljedeći. Ako se u kojem slučaju baza ne inicijalizira korisniku se prikaže zaslon za pogreške.

```
class MyApp extends ConsumerWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    AsyncValue<Box<String>> state = ref.watch(futureCreateBoxProvider);
    return MaterialApp(
      theme: ThemeData(
        scaffoldBackgroundColor: Colors.white,
      ), // ThemeData
      home: state.when(
        data: (_) => const MyWalletPage(),
        error: (_, __) => const SplashPage(),
        loading: () => const SplashErrorPage(),
      ),
    ); // MaterialApp
  }
}
```

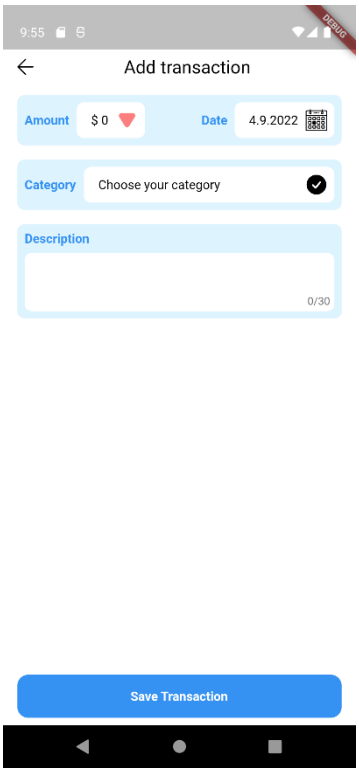
Sl. 4.13. Logika prikaza zaslona

Nakon što je završilo učitavanje i bilo je uspješno prikazuje se početni ekran kao na slici 4.14.



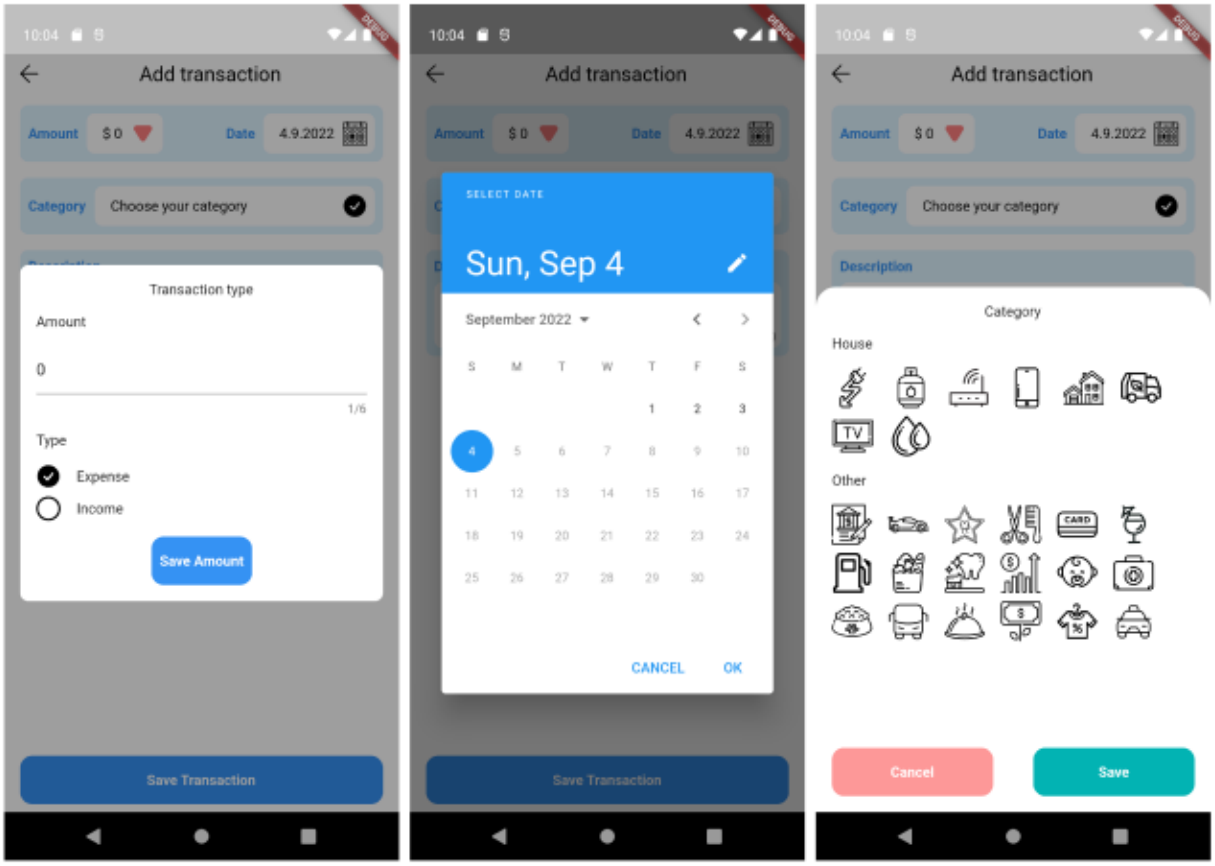
Sl. 4.14 Početni zaslon

Početni zaslon sadrži nekoliko ključnih funkcionalnosti, a to je prikaz trenutne mjesečne potrošnje, iznos troškova i zarade u trenutnom mjesecu i listu godišnjih transakcija. Na dnu zaslona nalazi se navigacijska traka koja ima dvije sekcije, a to su sekcija s općim prikazom transakcija i detaljni prikaz transakcija po mjesecima. Gumb koji se nalazi na sredini navigacijske trake služi za dodavanje novih transakcija. Pritiskom na njega otvara se zaslon za dodavanje transakcije kao na slici 4.15. zaslon se sastoji od četiri polja za unos, a to su polja za vrijednost, datum, kategorija i opis transakcije.



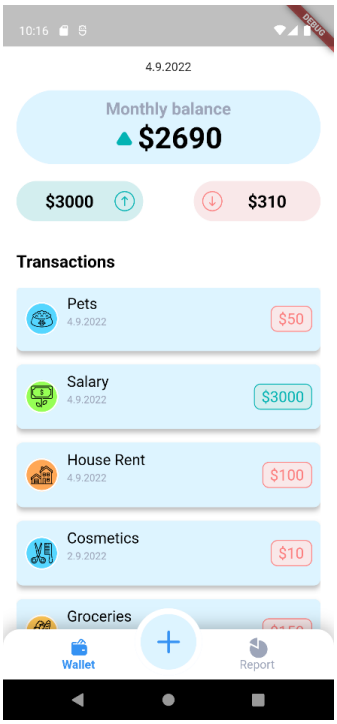
Sl. 4.15. Zaslon za dodavanje transakcija

Polja za unos vrijednosti, datuma i kategorije otvaraju zaseban prozor s elementima gdje korisnik upisuje ili odabire željenu vrijednost kao na slici 4.16.



Sl. 4.16. Prikaz zasebnih prozora

U slučaju da korisnik nije ispunio potrebna polja, a pokušava spremiti transakciju na zaslonu se prikazuje poruka da provjeri jesu li sva polja za unos ispunjena. Nakon što je korisnik uspješno popunio zadana polja i spremio transakciju, aplikacija vraća korisnika na početni zaslon gdje se sada nalazi novo dodana transakcija kao na slici 4.17.

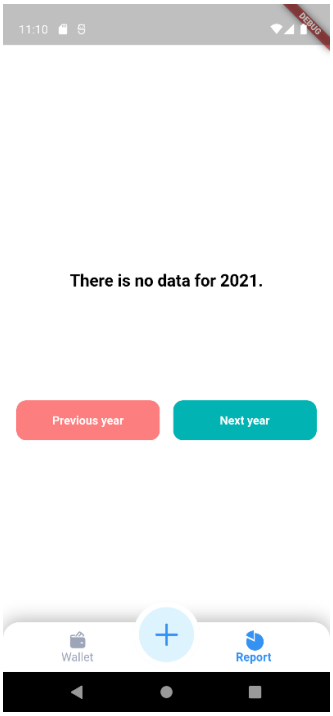


Sl. 4.17. Početni zaslon ispunjen transakcijama

Pritiskom na sekciju s detaljima transakcija, ako korisnik ima spremljene transakcije za tu godinu, na zaslonu će mu se prikazati graf za odabranu godinu. Korisnik također ima mogućnost da pritisne za željeni mjesec grafa gdje će mu se onda prikazati stanje, iznos troškova i zarade, kao što je prikazano na slici 4.18.

Sl. 4.18. Zaslon s detaljima transakcija

Ispod grafa nalaze se četiri gumba, od kojih prva dva gumba omogućuju korisniku da mijenja godine. U slučaju da ne postoji unos za izabranu godinu korisniku će biti prikazan zaslon kao na slici 4.19.



Sl. 4.19. Zaslon izvještaja kada nema unesene transakcije

Dok prvi od donja dva gumba omogućuje korisniku bolji uvid na kućansku potrošnju u obliku grafa s godišnjim popisom kućanskih transakcija, zadnji gumb nudi mogućnost prikaza svih transakcija za izabranu godinu prikazano na slici 4.20.

Sl. 4.20. Zaslon grafa kućanskih transakcija (lijevo) i zaslon svih transakcija (desno)

5. USPOREDBA RAZVOJA VIŠEPLATFORMSKIH I NATIVNIH APLIKACIJA

U ovom poglavlju bit će opisane neke od razlika kod višeplatformskih aplikacija u odnosu na nativne, te prednosti i nedostaci prilikom korištenja.

5.1. Proces razvoja

Držanjem najboljih praksi prilikom kodiranja osigurava programeru brz razvoj i jednostavno održavanje za različite platforme. Održavanje koda na jednome mjestu uvelike olakšava održavanje, također broj programera je smanjen jer s jednim Flutter programerom može se pokriti šest platformi dok u nativnom načinu proizvodnje bilo bi potrebno zaposliti jednog programera po platformi.

5.1.1. Odvajanje logike od korisničkog sučelja

Prilikom izrade aplikacije bitno je držati odvojeno elemente korisničkog sučelja i njihovu logiku. Posljedice ne bi bile vidljive na prvu, ali tijekom razvoja aplikacije i preglednost koda bi bio otežan. Aplikacija koja je napravljena bez odvajanja elemenata i logike bila bi jako teška za održavanje. Najmanja promjena na korisničkom sučelju mogla bi utjecati i na logiku, a ponovno iskorištavanje već postojećih elemenata i logike ne bi bilo moguće. Svaki element bi u sebi sadržavao svoju logiku i time ne bi nudio mogućnost prilagodbe. Svaka aplikacija ima svoj stil kojeg prati, tako da će se gumb za nastavak sigurno pojavljivati na nekoliko mjesta. Zbog pojavljivanja istog elementa s različitim zadaćama bitno je napraviti element koji je univerzalan. A to se postiže uređivanjem elemenata tako da budu lako prilagodljivi za bilo koju upotrebu. Slika 5.1. prikazuje gumb koji se može upotrijebiti svugdje u aplikaciji baš zbog tog što se logika ne odvija unutar elementa, nego ju programer određuje prilikom pozivanja elementa.

```dart
class PrimaryButton extends StatelessWidget {
  final Function function;
  final String title;
  final bool margin;
  final Color color;
  final double opacity;
  final double? width;
  const PrimaryButton({
    Key? key,
    required this.title,
    required this.function,
    this.margin = true,
    this.color = AppColors.primaryBlue,
    this.opacity = 1,
    this.width,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () => function(),
      child: Opacity(
        opacity: opacity,
        child: Container(
          width: width,
          margin: EdgeInsets.symmetric(
            vertical: margin ? 8 : 0,
          ), // EdgeInsets.symmetric
          padding: const EdgeInsets.symmetric(
            vertical: 16,
            horizontal: 8,
          ), // EdgeInsets.symmetric
          decoration: BoxDecoration(
            borderRadius: BorderRadius.circular(12),
            color: color,
          ), // BoxDecoration
          child: Text(
            title,
            style: caption.copyWith(color: AppColors.white),
            textAlign: TextAlign.center,
          ), // Text
        ), // Container
      ), // Opacity
    ); // GestureDetector
  }
}
```

Sl. 5.1. Prikaz koda univerzalnog gumba

Slika 5.2. prikazuje slučaj odustajanja ili spremanja transakcijske kategorije i pritom također prikazuje koliko je jednostavno koristiti jedan element za različite poslove. Kada element ne bi bio univerzalan onda bi svaki gumb bio zaseban element, ali ipak vrlo sličan i tada bi većina elementa bila jednaka, a samo logika unutar njih drugačija. U ovom slučaju dolazi do problema redundancije i gomilanja nepotrebnog koda.

```
Row(
  children: [
    Expanded(
      child: PrimaryButton(
        color: AppColors.primaryRed,
        title: 'Cancel',
        function: () => ref
            .read(showPopUpTypeProvider.notifier)
            .update((state) => PopUpType.none),
        opacity: 0.8,
      ), // PrimaryButton
    ), // Expanded
    const SizedBox(width: 40),
    Expanded(
      child: PrimaryButton(
        color: AppColors.primaryGreen,
        title: 'Save',
        function: () {
          TransactionSubType newSubType =
              ref.watch(subTypeProvider);
          ref
              .read(addTransactionProvider.notifier)
              .update(
                (state) => state.copyWith(
                    subtypeId: newSubType),
              );
          ref
              .read(showPopUpTypeProvider.notifier)
              .update((state) => PopUpType.none);
        },
      ), // PrimaryButton
    ), // Expanded
  ],
) // Row
```

Sl. 5.2. Korištenje univerzalnog gumba

5.1.2. Upravljanje promjenom stanja aplikacije

Jedna od najvažnijih stvari prilikom započinjanja projekta je odabir arhitekture koja će odvajati korisničko sučelje od logike aplikacije. Prije nekoliko godina razvojni okvir Flutter zbog svoje mladosti nije imao ustaljene arhitekture dok stariji jezici Swift i Kotlin već imaju razrađene arhitekture. Zbog toga su mnogi programeri pokušavali koristiti arhitekture iz različitih jezika ne bi li pronašli onaj koji najbolje odgovara tada novom razvojnom okviru. Koristile su se razne arhitekture poput MVC (engl. Model-View-Controller), MVP (engl. Model-View-Presenter), MVVM (engl. Model-View-ViewModel). Iako su te arhitekture odrađivale dobar posao ipak nisu bile najbolje prilagođene za trenutni razvojni okvir, s vremenom na pozornici su se pojavili novi načini odvajanja logike kao što su Provider, BLoC, Riverpod, GetX.

BLoC nudi jednostavan način za odvajanje logike i korisničkog sučelja, programeru nudi mogućnost da točno zna u svakom trenutku u kojem je stanju aplikacija. Jednostavan, snažan i lagan je za korištenje te pritom omogućava lagano testiranje takozvanih blokova u kojima se odvija promjena stanja. Glavni mu je cilj napraviti promjenu stanja predvidljivim reguliranjem tako da

se promjena stanja može učiniti na samo jedan način kroz cijelu aplikaciju [25]. Na slici 5.3. se može vidjeti dijagram promijene stanja, a jednostavni slučaj upotrebe ovog načina je da korisnik pritišće gumb koji šalje bloku događaj. Ovisno o poslanom događaju blok šalje zahtjev bazi podataka i nakon što dobije odgovor vraća novo stanje na korisničko sučelje koje okida ponovnu izgradnju zaslona.

Sl. 5.3. Način promijene stanja korištenjem BloC-a

Provider je zamišljen kao omotač oko naslijeđenog elementa i time mu omogućava jednostavnije korištenje i lagano višekratno upotrebljavanje. Omogućava korisniku jednostavnu alokaciju resursa, lijeno učitavanje [26]. Neki paketi poput Chewie paketa za prikaz video zapisa na mobilnim platformama ili Bloc-a ovise o Provideru.

Riverpod je nastao od istog kreatora kao i Provider, te postoji dosta sličnosti između njih dva, s tim da Riverpod ne ovisi o Flutter razvojnom okviru. Glavni ciljevi su mu sigurno kreiranje, praćenje i uništavanje stanja, jednostavno testiranje, bolja preglednost stanja pomoću Flutter programerskog alata i omogućiti aplikacijama s jednosmjernim protokom bolju skalabilnost [27]. U aplikaciji za praćenje režijskih troškova bit će korišten Riverpod jer se pokazao kao jednostavan alat bez dodatnog koda. U Riverpodu postoji nekoliko različitih tipova davatelja usluga, a to su budući davatelj usluga koji omogućava asinkrone pozive, običan davatelj usluga koji služi za spremanje podataka, davatelj usluga sa stanjima, davatelj usluga u obliku potoka. Svaki od njih ima svoju zadaću, kako bi element mogao koristiti Riverpod davatelje usluge bitno ga je označiti potrošačem kako bi imao pristup referenci, svaki davatelj usluge je postavljen kao globalna varijabla tako da je dostupan cijeloj aplikaciji. Slika 5.4. prikazuje na koji je način postavljen davatelj usluga za donju navigaciju. Navigacija ima dvije sekcije, gdje prva je općeniti prikaz transakcija i potrošnje dok druga sekcija nudi više detalja oko godišnje i kućne potrošnje. Prilikom postavljanja davatelja usluga u ovom slučaju potrebno mu je zadati početnu vrijednost odnosno zaslon koji će biti prikazan prilikom ulaska u aplikaciju.

```
enum BottomNavigation {
  wallet,
  report,
}

final bottomNavProvider = StateProvider<BottomNavigation>(
  (ref) => BottomNavigation.wallet,
);
```

Sl. 5.4. Postavljanje davatelja usluga

Slika 5.5. prikazuje donju navigaciju koja je označena kao element potrošač samim time ime mogućnost korištenja Riverpod reference. Na samome početku bloka za izgradnju postavljena je varijabla navProvider kojoj je zadaća slušati promjene na davatelju usluga i ako dođe do promjene ponovo izgraditi taj element. Što znači da prilikom pritiska na sekciju donje navigacije dolazi do ažuriranja stanja odnosno vrijednosti davatelja usluga, koji javlja svim potrošačima koji ga slušaju da je došlo to promijene i da se moraju ponovo izgraditi. Promjena stanja se izvršava pomoću čitanja reference gdje se poziva davatelj usluge. Nakon toga ide poziv na metodu za ažuriranje gdje se predaje novo stanje.

```dart
class HomeBottomNavigation extends ConsumerWidget {
  const HomeBottomNavigation({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final navProvider = ref.watch(bottomNavProvider);
    return Align(
      alignment: Alignment.bottomCenter,
      child: Stack(
        alignment: Alignment.bottomCenter,
        children: [
          Container(
            padding: const EdgeInsets.symmetric(horizontal: 60),
            decoration: const BoxDecoration( // BoxDecoration ···
            child: Padding(
              padding: const EdgeInsets.all(10),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  NavigationItem(
                    title: 'Wallet',
                    iconPath: AppAssetsPath.wallet,
                    isPicked: navProvider == BottomNavigation.wallet,
                    func: () {
                      ref
                          .read(bottomNavProvider.notifier)
                          .update((state) => BottomNavigation.wallet);
                      ref
                          .read(yearlyChartProvider.notifier)
                          .update((state) => null);
                    },
                  ), // NavigationItem
                  NavigationItem(
                    title: 'Report',
                    iconPath: AppAssetsPath.report,
                    isPicked: navProvider == BottomNavigation.report,
                    func: () => ref
                        .read(bottomNavProvider.notifier)
                        .update((state) => BottomNavigation.report),
                  ), // NavigationItem
                ],
              ), // Row
            ), // Padding
          ), // Container
          const AddTransactionButton(),
        ],
      ), // Stack
    ); // Align
  }
```

Sl. 5.5. Donja navigacija

Optimizacija pretraživača odnosno SEO (engl. search engine optimization) je najveći nedostatak na Flutter web platformi. SEO je proces poboljšavanja web stranice kako bi povećali njezinu vidljivost kada ljudi pretražuju usluge ili proizvode preko pretraživača kao što su Google, Bing i ostali. Povećavanjem vidljivosti također se povećava i vjerojatnost da će proizvod privući više pozornosti i veći broj kupaca.

Tab. 5.1 Prednosti i nedostaci u procesu razvoja višeplatformskih aplikacija

Prednosti	Nedostatci
Brz razvoj	Nedostatak najboljih praksi
Održavanje aplikacije pomoću jednog koda	Relativno mlada tehnologija
Cjenovno efikasan	Ovisnost o podršci trećih stranaka
Moguće integracije s nativnim kodom	SEO
Odlična dokumentacija	

5.2. Performanse

Svaka nova verzija razvojnog okvira Flutter donosi sve bolje i bolje performanse. Jedno od glavnih razloga zašto koristiti Flutter kada se ide na višeplatformski razvoj je zbog performansi koje su gotovo iste kao i na nativnim aplikacijama. Performanse aplikacije ne ovise samo o razvojnom okviru, nego i o programeru. Važno je znati koji su elementi zahtjevni po pitanju performansi i naučiti kada ih je u redu koristiti, a kada ih je bolje izbjegavati. Jednostavan primjer može se uzeti prilikom izrade ekrana na kojemu se nalazi lista s velikim brojem elemenata jedan ispod drugog. Ako bi se koristio kod sa slike 5.6. onda bi došlo do trzanja prilikom listanja transakcija zbog toga što bi svi elementi već bili izgrađeni i čuvani u memoriji što bi jako utjecalo na performanse.

```dart
class TransactionList extends ConsumerWidget {
  const TransactionList({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final box = ref.watch(boxProvider);
    return box != null
        ? ValueListenableBuilder(
            valueListenable: box.listenable(keys: ['${DateTime.now().year}']),
            builder: (context, __, _) {
              List<Transaction> listOfTransactions =
                  getThisYearBox(box)?.yearlyTransactions ?? [];
              listOfTransactions.sort((a, b) => b.date.compareTo(a.date));
              return ListView(
                shrinkWrap: true,
                children: [
                  //list of transactions
                ],
              ); // ListView
            },
          ) // ValueListenableBuilder
        : const Center(
            child: Text('No transactions for current year.'),
          ); // Center
  }
}
```

Sl. 5.6. Lista transakcija s lošijim performansama

Slika 5.7. prikazuje način koji je bolji kada trebamo prikazati listu koja ima puno elemenata, zbog toga što će na ovaj način biti izgrađeni samo elementi odnosno transakcije koje su vidljive na zaslonu. Sve ostale transakcije će biti izgrađene netom prije, nego što je njihov red za prikazivanje.

```dart
class TransactionList extends ConsumerWidget {
  const TransactionList({
    Key? key,
  }) : super(key: key);

  @override
  Widget build(BuildContext context, WidgetRef ref) {
    final box = ref.watch(boxProvider);
    return box != null
        ? ValueListenableBuilder(
            valueListenable: box.listenable(keys: ['${DateTime.now().year}']),
            builder: (context, __, _) {
              List<Transaction> listOfTransactions =
                  getThisYearBox(box)?.yearlyTransactions ?? [];
              listOfTransactions.sort((a, b) => b.date.compareTo(a.date));

              return ListView.builder(
                itemCount:
                    listOfTransactions.isEmpty ? 0 : listOfTransactions.length,
                shrinkWrap: true,
                itemBuilder: (context, index) => Container( // Container ···
              ); // ListView.builder
            },
          ) // ValueListenableBuilder
        : const Center(
            child: Text('No transactions for current year.'),
          ); // Center
  }
}
```

Sl. 5.7. Lista transakcija s boljim performansama

Jedan od poznatih problema su animacije prijelaza između zaslona na mobilnim platformama osobito na iOS platformi, zbog toga što je se animacije učitavaju tek kada dođe red na njih što je prekasno kod mobitela koji nemaju najbolje specifikacije. Na službenoj stranici razvojnog okvira Flutter može se pronaći način kako smanjiti zastajkivanje animacija [28].

Tab. 5.2. Prednosti i nedostaci performansi višeplatformskih aplikacija

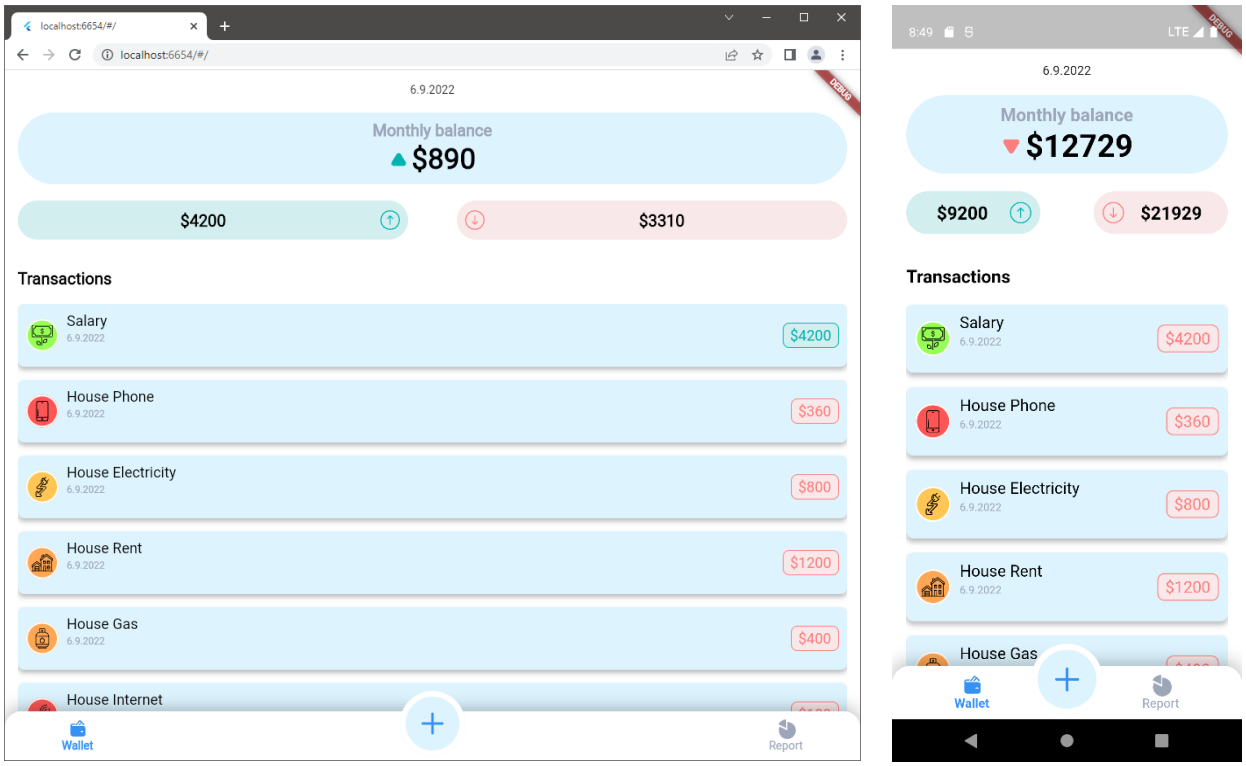
Prednosti	Nedostatci
Nativni osjećaj korištenja	Nisu sve platforme jednako optimizirane
	Učitavanje animacije prijelaza
	Performanse kod zahtjevnih aplikacija (npr. aplikacije proširene stvarnosti, igre,)

5.3. UI – korisničko sučelje

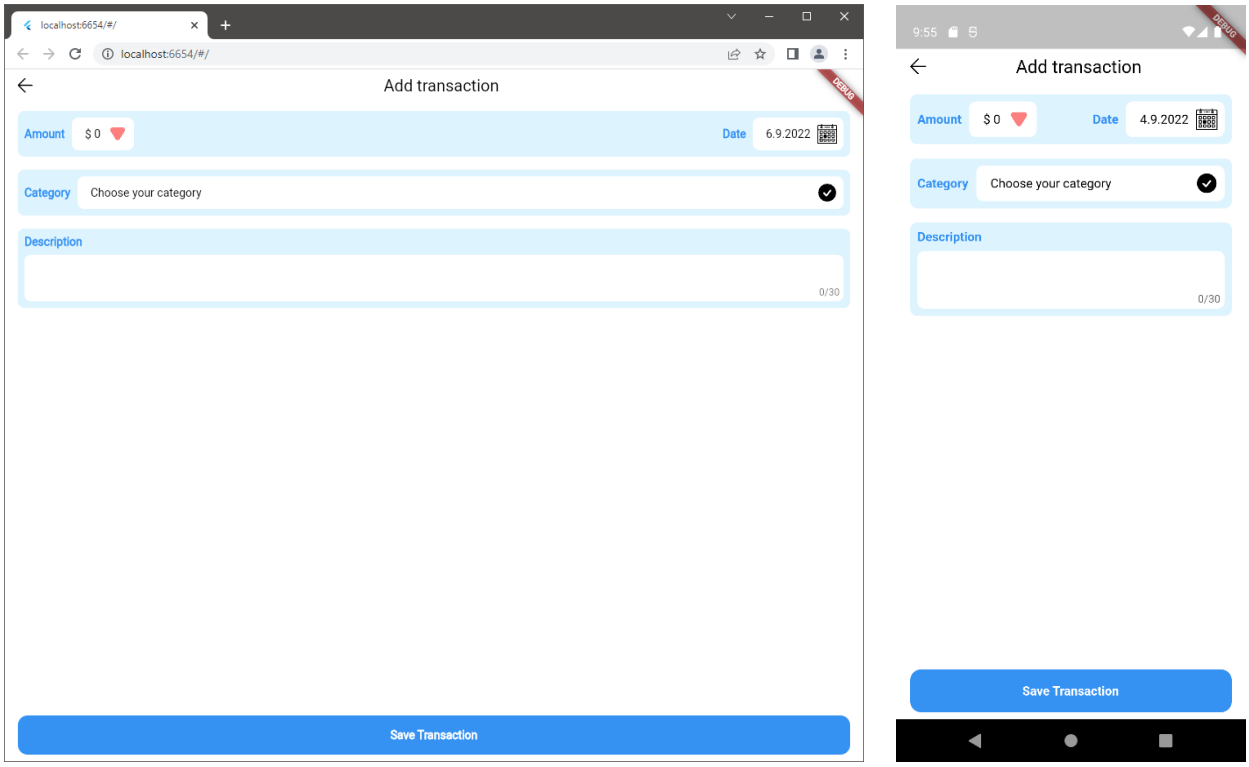
Flutter razvojni okvir nudi dizajnerima veliku slobodu oko kreiranja korisničkog sučelja, ali zadaje im i izazove kada sučelje treba prilagoditi na nekoliko različitih platformi. Aplikacija za praćenje režijskih troškova ista je izgledom i funkcionalnostima na web i mobilnoj platformi, samo što su na web platformi elementi izduženiji. Kako bi dizajn s web platforme bio što sličniji mobilnoj potrebno je napraviti nekoliko izmjena tako što bi se utjecalo na ograničavanje visine i širine elemenata ili promjene rasporeda elemenata. Slika 5.8. prikazuje kako izgleda aplikacija koja je pokrenuta na dvije različite platforme, gdje vidimo da je jedina razlika između platformi je oblik elementa dok je sve ostalo isto. Slike 5.9. - 13. prikazuju ostale zaslone u odnosu na isti taj zaslon u drugoj platformi. Bitno je imati na umu da su platforme različite što znači ako se na mobilnoj platformi neki dio može listati pomicanjem prsta ne znači da će se isto to moći i na webu pomicanjem pokazivača. To je moguće izvesti, ali treba napraviti par malih prilagodbi. Zbog načina bojanja odnosno stvaranja piksela na zaslonu Flutter aplikacija je ista na svakoj platformi i time nudi održavanje dizajna ili brenda jednakim na nekoliko različitih platformi. Iako Flutter Svg (engl. Scalable Vector Graphics) paket podržava običnu verziju Svg-a koji sadrži vektore, ukoliko Svg sadrži gradijent u sebi on se neće prikazati. To znači da programer mora napraviti gradijent na drugi način kako bi pratio zadani dizajn.

Tab. 5.3. Prednosti i nedostaci prilikom izrade korisničkog sučelja višeplatformskih aplikacija

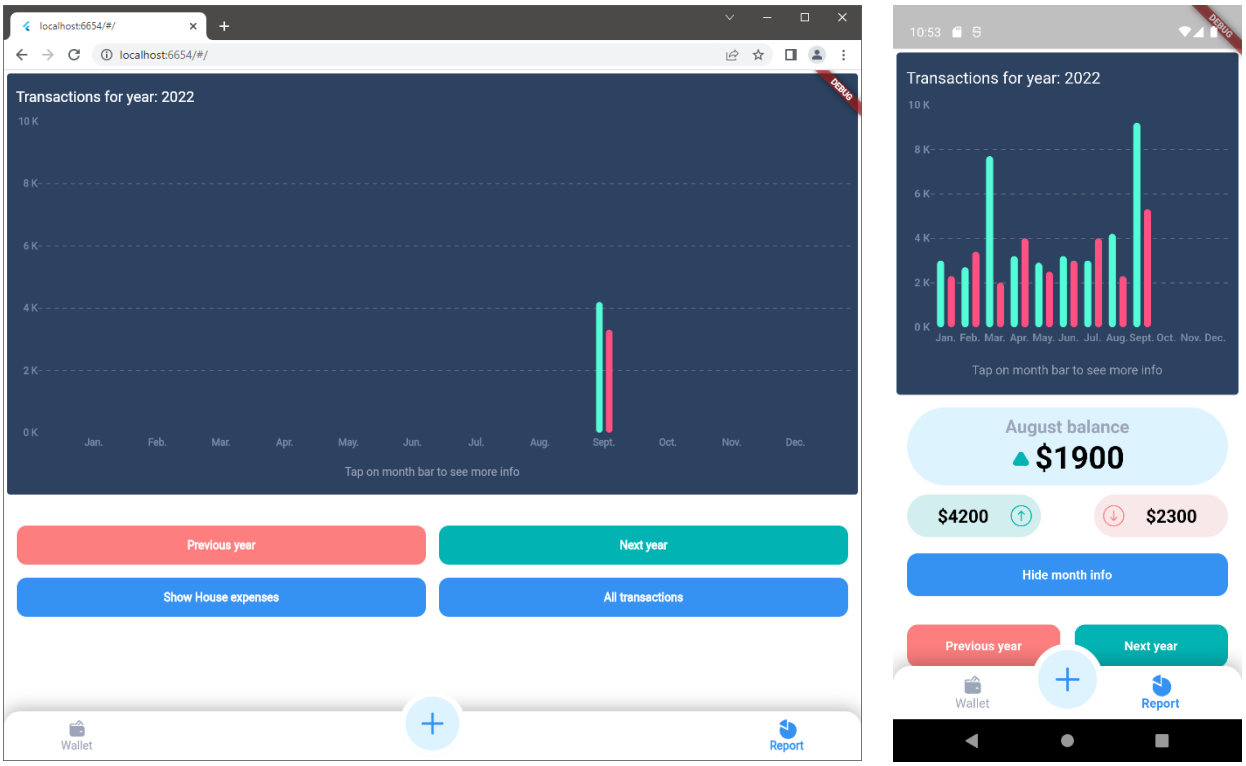
Prednosti	Nedostatci
Jednostavna izrada korisničkog sučelja	Svg gradijent nije podržan
Prilagodba elemenata	



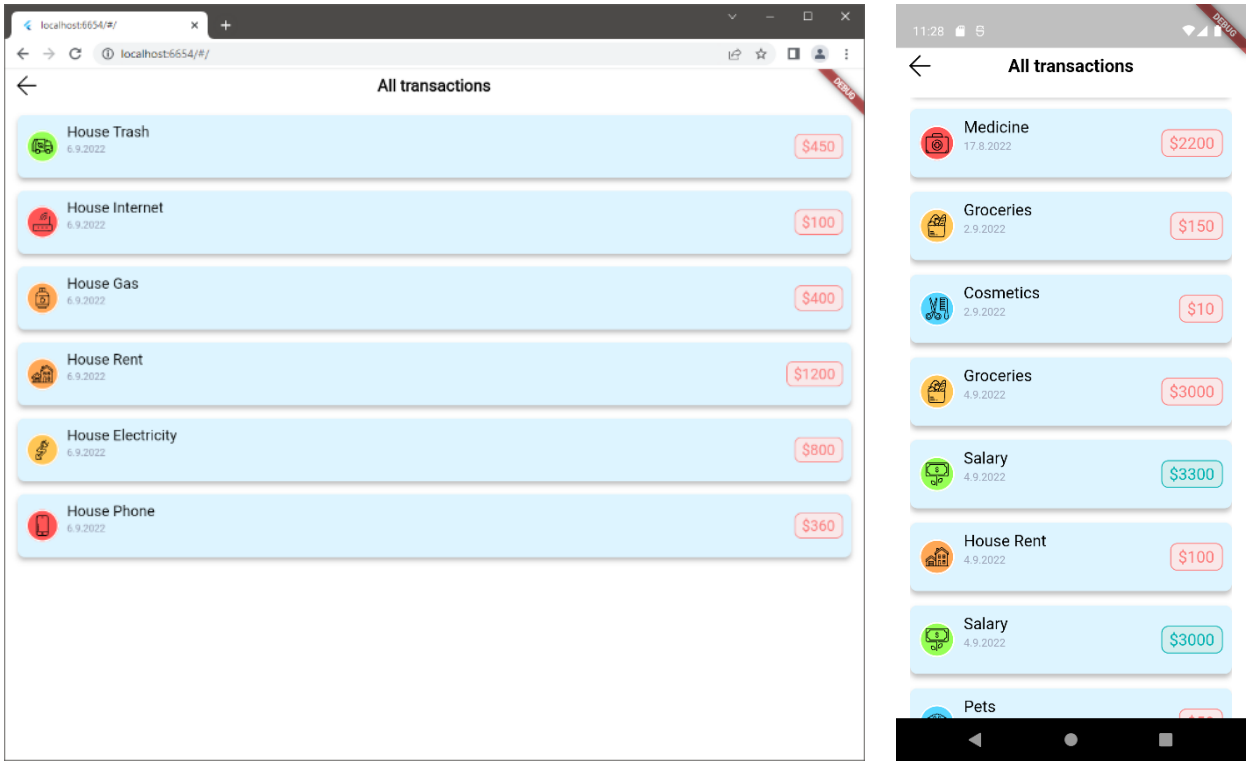
Sl. 5.8. Usporedba početnog zaslona na web i mobilnoj platformi



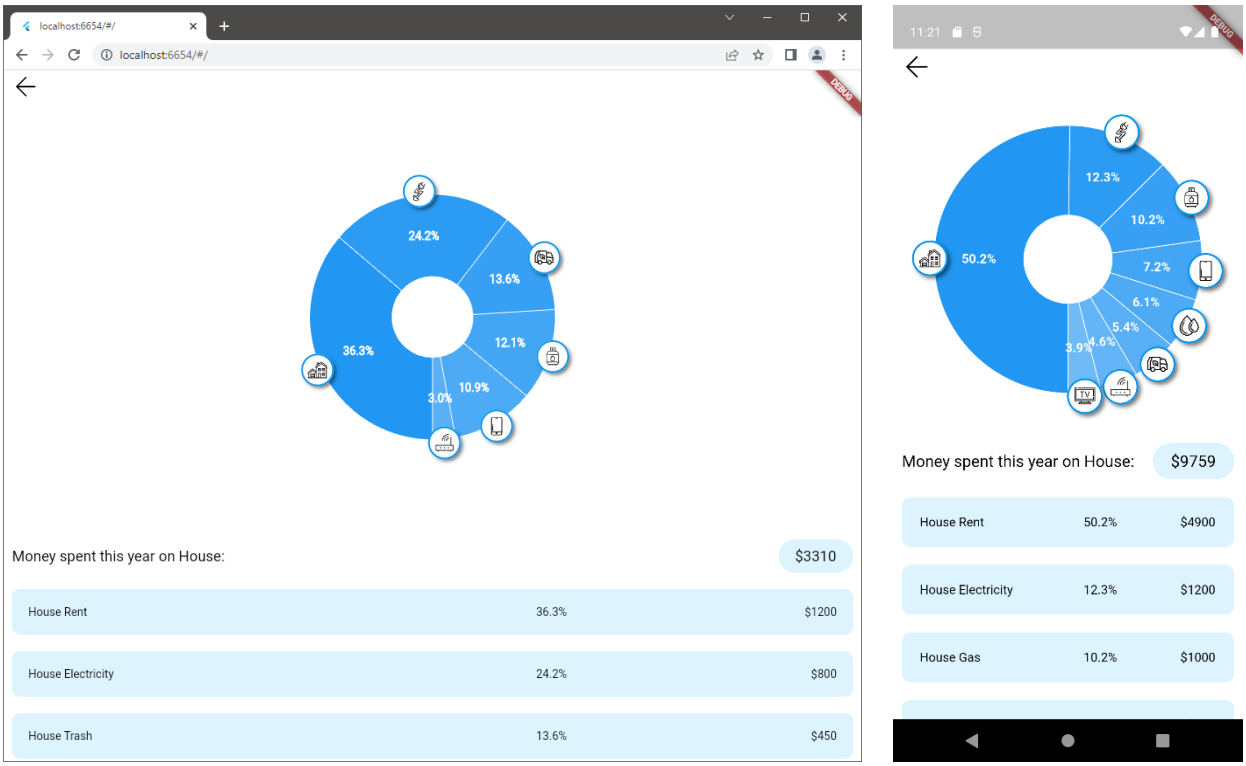
Sl. 5.9. Usporedba zaslona za dodavanje transakcije na web i mobilnoj platformi



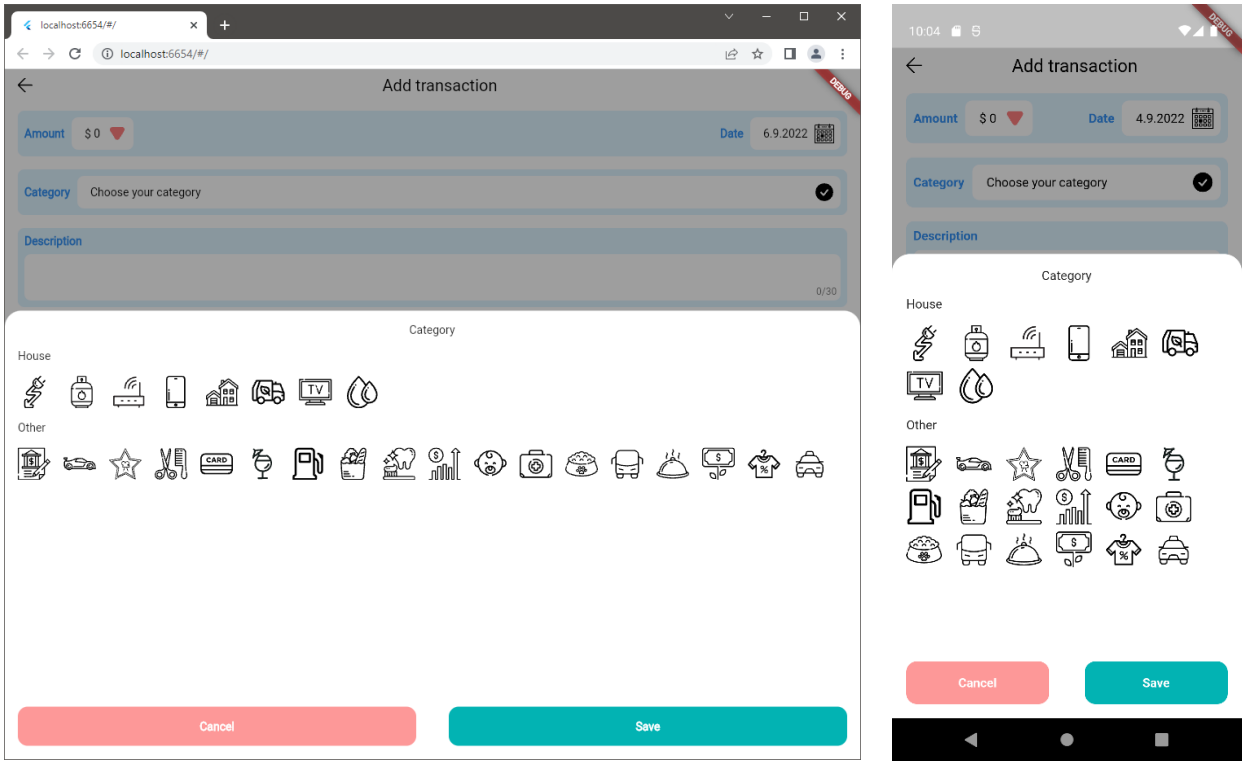
Sl. 5.10. Usporedba zaslona detaljnog prikaza transakcija na web i mobilnoj platformi



Sl. 5.11. Usporedba zaslona svih transakcija na web i mobilnoj platformi



Sl. 5.12. Usporedba zaslona kućne potrošnje na web i mobilnoj platformi



Sl. 5.13. Usporedba prozora za dodavanje kategorije između web i mobilne platforme

6. ZAKLJUČAK

Brzim napretkom tehnologije i stalnim novim otkrićima, alati koji se koriste danas i oni koji će se koristiti za pet godina više neće biti isti. Tako nitko nije mogao ni zamisliti 2018. godine da će se Flutter razvojni okvir ukorijeniti u mobilnom razvoju i postati jedan od najkorištenijih razvojnih okvira za izradu višeplatformskih aplikacija. U slučaju Flutter razvojnog okvira korisnici imaju ogromnu ulogu jer ne samo da šire popularnost koristeći ga u aplikacijama, nego oni također pomažu u razvoju i zajedno oblikuju budućnost razvojnog okvira.

Važno je napomenuti da Flutter nije u mogućnosti pokriti sve tipove aplikacija koje se nalaze ili dolaze na tržište. Na primjer aplikacije proširene stvarnosti zahtijevaju pristup različitim senzorima i paketima koji trenutno nisu dostupni u Flutter razvojnom okviru. Također, ako su glavni ciljevi aplikacije da bude brza i kompaktna onda je nativni razvoj najbolji odabir zbog toga što nudi sigurnost, ali i pristup najnižim dijelovima operacijskog sustava prilikom programiranja. Iako su sve platforme stabilne valja naglasiti da su mobilne platforme najbolje optimizirane za produkcijski razvoj, dok web još zaostaje jer još nije riješen jedan vrlo važni izazov, a to je SEO optimizacija u Flutteru. SEO ključan na web platformi ako klijent želi da njegov proizvod bude predložen kao prvi i najbolji odabir prilikom pretrage na web stranicama.

Ipak, ako je bitno napraviti aplikaciju u kratkom vremenu koja izvršava svoju svrhu bez ikakvih problema i pritom ima laku proširivost na ostale platforme onda je Flutter pravo rješenje. Svakom godinom Flutter razvojni okvir se sve više približava nativnom razvoju. Možda ga nikada neće u potpunosti zamijeniti, ali omogućit će mnogim programerima da uđu u svijet izrade aplikacija. Napraviti aplikaciju koja se nalazi na nekoliko platformi više nije problem, važno je dobro isplanirati što će se i kada koristiti, kako bi se proširivanje na ostale platforme obavilo bez većih problema.

LITERATURA

[1] Meta Platforms, „React Native · Learn once, write anywhere“, *Meta Platforms*, 2022. https://reactnative.dev/ (pristupljeno 26. ožujak 2022.).

[2] R. Arlin, „Testing“, *Medium*, 2020. https://medium.com/@johanes123renaldi/testing-5a6b57339f5d (pristupljeno 07. rujan 2022.).

[3] M. Frachet, „Understanding the React Native bridge concept | HackerNoon“, *Hackernon*, 2017. https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8 (pristupljeno 26. ožujak 2022.).

[4] B. Drzewiński, „How to Build Cross-platform Apps with React Native Bridges“, *DeSmart*, 2021. https://desmart.com/business/how-to-build-cross-platform-apps-with-react-native-bridges/ (pristupljeno 07. rujan 2022.).

[5] N. Chrzanowska, „13 Great Examples of React Native Apps in 2021 [Updated]“, *Netguru*, 2019. https://www.netguru.com/blog/react-native-apps (pristupljeno 26. ožujak 2022.).

[6] T. Altexsoft, „The Good and The Bad of Xamarin Mobile Development“, *AltexSoft*, 2020. https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/ (pristupljeno 26. ožujak 2022.).

[7] A. Michaeli, „Microsoft Goes All-In On React Native For Their Mobile Apps“, *Appfigures*, 2019. https://appfigures.com/resources/insights/microsoft-goes-all-in-on-react-native (pristupljeno 26. ožujak 2022.).

[8] Microsoft, „Xamarin customers showcase | .NET“, *Microsoft*, 2022. https://dotnet.microsoft.com/en-us/platform/customers/xamarin (pristupljeno 26. ožujak 2022.).

[9] Xamarin, *English: Logo of Xamarin.* 2010. Pristupljeno: 07. rujan 2022. [Na internetu]. Dostupno na: https://commons.wikimedia.org/wiki/File:Xamarin-logo.svg

[10] T. Altexsoft, „The Good and the Bad of Ionic Mobile Development“, *AltexSoft*, 2019. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-development/ (pristupljeno 26. ožujak 2022.).

[11] Wikipedia, „Apache Cordova“, *Wikipedia*, 2022. https://en.wikipedia.org/w/index.php?title=Apache_Cordova&oldid=1070983233 (pristupljeno 26. ožujak 2022.).

[12] The Apache Software Foundation, „Artwork - Apache Cordova“, *Apache Cordova*, 2022. https://cordova.apache.org/artwork/ (pristupljeno 07. rujan 2022.).

[13] L. Sujay Vailshery, „Cross-platform mobile frameworks used by global developers 2021“, *Statista*, 2021. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/ (pristupljeno 26. ožujak 2022.).

[14] Flutter, „Flutter - Build apps for any screen“, *Flutter*, 2022. https://flutter.dev/ (pristupljeno 13. rujan 2022.).

[15] Wikipedia, „Dart (programming language)“, *Wikipedia*, 2022. https://en.wikipedia.org/w/index.php?title=Dart_(programming_language)&oldid=1091485829 (pristupljeno 02. rujan 2022.).

[16] Dart, „Dart overview“, *Dart*, 2022. https://dart.dev/overview.html (pristupljeno 30. kolovoz 2022.).

[17] S. Sinha, *Quick Start Guide to Dart Programming: Create High-Performance Applications for the Web and Mobile*. Apress, 2019.
[18] S. Ford, „Dart 2 for C# and Java Developers“, *Toptal Engineering Blog*, 2019. https://www.toptal.com/dart/dartlang-guide-for-csharp-java-devs (pristupljeno 30. kolovoz 2022.).
[19] T. Bailey i A. Biessek, *Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter 2.5 and Dart, 2nd Edition*. Packt Publishing Ltd, 2021.
[20] Flutter, „Widget catalog“, *Flutter*, 2022. https://docs.flutter.dev/development/ui/widgets (pristupljeno 29. kolovoz 2022.).
[21] Flutter, „AnimatedContainer class - widgets library - Dart API“, *Flutter*, 2022. https://api.flutter.dev/flutter/widgets/AnimatedContainer-class.html (pristupljeno 07. rujan 2022.).
[22] T. Sneath, „Introducing Flutter 3“, *Flutter*, 2022. https://medium.com/flutter/introducing-flutter-3-5eb69151622f (pristupljeno 31. kolovoz 2022.).
[23] „Pinterest“, *Pinterest*, 2022. https://www.pinterest.com/search/pins/?q=expenses%20app (pristupljeno 07. rujan 2022.).
[24] Code with Andrea, „Flutter Project Structure: Feature-first or Layer-first?“, *Code With Andrea*, 2022. https://codewithandrea.com/articles/flutter-project-structure/ (pristupljeno 30. kolovoz 2022.).
[25] Bloc, „Bloc State Management Library“, *Bloc library*, 2022. https://bloclibrary.dev/#/whybloc (pristupljeno 01. rujan 2022.).
[26] R. Rousselet, „Provider“, *GitHub*, 2022. https://github.com/rrousselGit/provider (pristupljeno 01. rujan 2022.).
[27] Riverpod contributors, „Riverpod | Dart Package“, *Dart packages*, 2022. https://pub.dev/packages/riverpod (pristupljeno 01. rujan 2022.).
[28] Flutter, „Reduce shader compilation jank on mobile“, *Flutter*, 2022. https://docs.flutter.dev/perf/shader (pristupljeno 06. rujan 2022.).

SAŽETAK

Glavni cilj ovog diplomskog rada bio je utvrditi koje su prednosti i nedostaci prilikom višeplatformskog programiranja. U radu su opisani različiti višeplatformski razvojni okviri, ali glavni razvojni okvir koji je uzet za usporedbu s nativnim razvojem je bio Flutter. Opisane su glavne značajke i postavljanje višeplatformskog razvojnog okvira Flutter i kakvo je trenutno stanje. Razvijena je i aplikacija za praćenje troškova koja služi kao ogledni primjer razvoja višeplatformske aplikacije u Flutteru. Uspoređena su različita područja koja su bitna i utječu na odabir načina razvoja te su navedeni prednosti i nedostaci višeplatformskog razvoja.

Ključne riječi: aplikacija, dart, flutter, višeplatformski razvoj

ABSTRACT

Title: Advantages and disadvantages of developing cross-platform applications

The main goal of this thesis was to determine the advantages and disadvantages of multi-platform programming. Various cross-platform development frameworks are described in the paper, but the main development framework taken for comparison with native development was Flutter. The main features and setup of the Flutter cross-platform development framework are described and the current state of the framework. An application for monitoring costs was also developed, which serves as an example of the development of a cross-platform application in Flutter. Different areas that are important and influence the choice of development methods are compared, and the advantages and disadvantages of multi-platform development are listed.

Keywords: application, cross-platform development, dart, flutter

ŽIVOTOPIS

Antonio Krušlin rođen je 22. srpnja 1998. godine u Đakovu. Pohađao je Osnovnu školu Bilje, te zatim upisuje III. gimnaziju Osijek 2013. godine. Nakon završetka srednje škole 2017. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Nakon završetka preddiplomskog studija upisuje diplomski studij na istom fakultetu smjer programsko inženjerstvo. Tijekom trajanja diplomskog studija radio je u Osječkoj tvrtki COBE gdje je razvijao mobilne aplikacije u Flutter razvojnom okviru.

Antonio Krušlin

Potpis autora