THE USE OF THE FLUTTER FRAMEWORK IN THE DEVELOPMENT PROCESS OF HYBRID MOBILE APPLICATIONS

Slavimir Stošović
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
slavimir.stosovic@akademijanis.edu.rs
Dušan Stefanović
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
dusan.stefanovic@akademijanis.edu.rs
Milan Bogdanović
Academy of Applied Technical and Preschool Studies, Nis, Serbia, milanbogdanovic.nish@gmail.com
Nikola Vukotić
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
nikola.vukotic@akademijanis.edu.rs

Abstract: The digitization trend poses the challenge to development engineers to design and program software solutions compatible with different hardware platforms and operating systems in the shortest possible time. These differences are particularly prominent between Android and iOS mobile devices. For this reason, certain frameworks are created that enable the rapid creation of application software for different hardware platforms by writing only one code base. Applications created in this way are called hybrid applications. This paper was written to familiarize the reader with the current possibilities of applying modern web technologies for the creation of hybrid mobile applications. Through this paper, the principles of functioning of the Flutter working framework are presented to the reader, as well as the basics of the Dart programming language, used in the Flutter framework. The inner workings concepts of the Flutter framework are presented to the reader through practical examples of implementations of the application for purchasing and booking airplane tickets. The work also shows the internal system of widgets, which are used to create parts of the graphical interface within the application, as well as parts of the code of the project written in the Dart programming language.
Keywords: Flutter, Framework, Dart programming language, Widgets, Hybrid Apps, Web

KORIŠĆENJE FLUTTER RADNOG OKVIRA U RAZVOJU HIBRIDNIH MOBILNIH APLIKACIJA

Slavimir Stošović
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
slavimir.stosovic@akademijanis.edu.rs
Dušan Stefanović
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
dusan.stefanovic@akademijanis.edu.rs
Milan Bogdanović
Academy of Applied Technical and Preschool Studies, Nis, Serbia, milanbogdanovic.nish@gmail.com
Nikola Vukotić
Academy of Applied Technical and Preschool Studies, Nis, Serbia,
nikola.vukotic@akademijanis.edu.rs

Sadržaj: Trend digitalizacije postavlja pred razvojne inženjere izazov da u što kraćem vremenu projektuju i isprogramiraju softverska rešenja kompatibilna sa različitim hardverskim platformama i operativnim sistemima. Ove razlike su naročito istaknute između Android i iOS mobilnih uređaja. Zbog toga nastaju određeni radni okviri (*framework*) koji omogućavaju ubrzano kreiranje aplikativnog softvera za različite hardverske platforme pisanjem samo jedne baze koda. Ovako nastale aplikacije se nazivaju hibridne aplikacije. Ovaj rad napisan je kako bi čitaoca upoznao sa trenutnim mogućnostima primene modernih veb tehnologija za izradu hibridnih mobilnih aplikacija. Kroz ovaj rad čitaocu su predstavljeni principi funkcionisanja Flutter radnog okvira, kao i osnove programskog jezika Dart, koji se koristi. Kroz praktične primere realizacija aplikacije za kupovinu i rezervaciju avionskih karata,

čitaocu su predstavljeni koncepti funkcionisanja Flutter radnog okvira. Rad takođe prikazuje unutrašnji sistem vidžeta pomoću koga se i kreiraju delovi grafičkog interfejsa unutar aplikacije kao i delove koda projekta napisanog u Dart programskom jeziku.
Ključne reči: Hibridne aplikacije, Mobilne aplikacije, Flutter, radni okvir, Dart programski jezik, Vidžeti, veb tehnologije

1. UVOD

Savremene veb aplikacije omogućavaju korisnicima da bez ikakvih dodatnih instalacija na svom računaru, ili mobilnom telefonu pristupaju sofisticiranim softverskim rešenjima zbog kog se i samo korisničko tržište preorijentisalo od desktop aplikacija ka veb aplikacijama. Dodatni razvoj tehnologija na pametnim telefonima, takođe je doprineo trendu zapostavljanja desktop aplikacija. Mobilni uređaji predstavljaju glavnu platformu putem koje korisnici pristupaju internetu. Prema podacima iz 2021. godine, 56.53% internet saobraćaja ostvaren je preko pametnih telefona [1]. Upravo zbog svoje prenosivosti i dostupnosti pametni telefoni predstavljaju najveće tržište korisničkih aplikacija [2] [3].

Prvobitno su se aplikacije za mobilne telefone programirale u razvojnim okruženjima i korišćenjem alata podržanih od strane najvećih zastupnika mobilnih operativnih sistema, Android i iOS. Alate za izradu aplikacija za Android operativni sistem sadrži „Android Studio“ razvojno okruženje [4], a koristi se Java ili Kotlin programski jezik. Za izradu aplikacija za iOS operativni sistem koristi se „Xcode“ [5], razvojno okruženje sa integrisanom podrškom za pisanje aplikacija, a koristi se Objective-C ili Swift programski jezik. Aplikacije napisane uz pomoć prvobitnih alata i programskih jezika nazivaju se *Native* aplikacije (*Native eng.* Domaće).

Pored *Native* tehnologija, Veb tehnologije pružaju novi pristup izradi mobilnih aplikacija. Aplikacijama napisanim uz pomoć veb tehnologija je zajedničko to da u toku izrade aplikacije i njenog korišćenja, one pružaju mogućnost platformske nezavisnosti upravo zbog primenjenih alata u toku izrade aplikacije. Aplikacije napisane uz pomoć veb tehnologija, a koje se mogu instalirati konvencionalnim putem, preko zvaničnog distributera aplikacije za Android ili iOS nazivaju se hibridne aplikacije (*Hybrid apps eng.*). Njihova glavna odlika je to što predstavljaju kombinaciju *Native* tehnologija namenjenih za izradu aplikacija na određenom operativnom sistemu i veb tehnologija. Prednost hibridnih aplikacija u odnosu na *Native* aplikacije je to što programer poznavanjem samo jednog jezika ima mogućnost da stvara aplikacije za više različitih operativnih sistema. Sa druge strane održavanje aplikacije za različite platforme je olakšano jer se radi na izmeni samo jednog izvornog koda.

Postoji više radnih okvira koji omogućuju izradu hibridnih aplikacija, a koji se razlikuju u načinu rešavanja nezavisnosti od operativnih sistema i prilagođavanja izgleda ulazno-izlaznih funkcionalnosti aplikacije standardnim aplikacijama napisanim u svojim *Native* tehnologijama. Najpopularniji su Xamarin, React-Native, Apache Cordova (prethodno poznatiji kao "*PhoneGap*"), Ionic i NativeScript. Među mogućim rešenjima za izradu hibridnih višeplatformskih aplikacija nalazi se i najmlađi Flutter (*Trepet eng*.), koji je zapravo radni okvir predstavljen od strane Google-a, kao skup alata za izradu korisničkih aplikacija.

2. FLUTTER RADNI OKVIR

Flutter je višeplatformski alat za izradu korisničkih interfejsa, koji koristi programski jezik Dart. Dizajniran je tako da omogućava ponovnu upotrebu koda u operativnim sistemima kao što su iOS i Android, dok takođe omogućava aplikacijama da se direktno povezuju sa sistemskim uslugama izabrane platforme [6]. Svrha ovog radni okvira je da omogući programerima da isporuče aplikacije visokih performansi koje krajnjem korisniku pružaju osećaj korišćenja *Native* aplikacije na različitim platformama, prihvatajući sistemske razlike i funkcionalnosti u zavisnosti od ponašanja platforme na kojoj se izvršava, dok istovremeno dele što više zajedničkog koda, koji najčešće opisuje poslovnu logiku aplikacije.

2.1 Programski jezik Dart

Za rad sa Flutter radnim okvirom koristi se programski jezik Dart. Dart je osmišljen i dizajniran za razvoj klijentskih aplikacija, kao što su veb i mobilne aplikacije, takođe može se koristiti i za razvoj aplikacija za personalne računare na tri najpopularnija operativna sistema, Linux, Windows, i MacOS. Dart je objektno orijentisan programski jezik, zasnovan na klasama, jezik koji omogućava sakupljanje smeća sa sintaksom u stilu C programskog jezika.

Slika 1. Prikaz dela Dart koda primenjenog u projektom zadatku

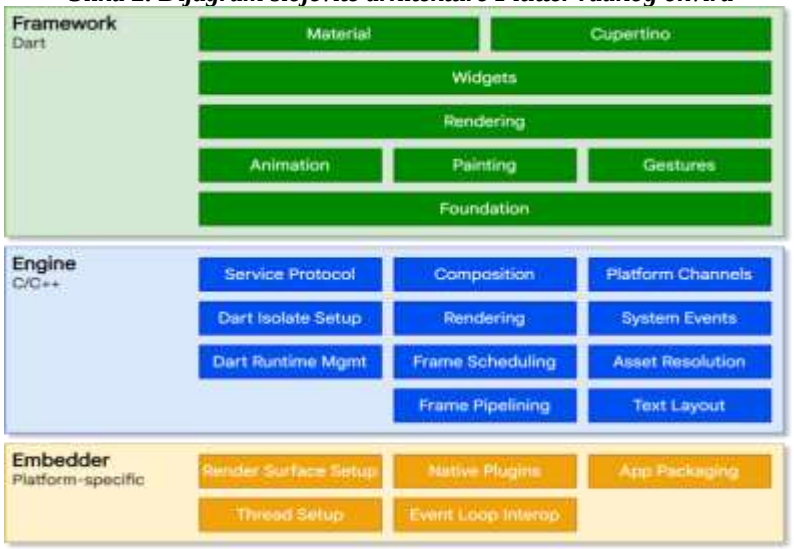
```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(const MyApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: primary,
      ), // ThemeData
      home: LoginScreen(),
    ); // MaterialApp
  }
}
```

Dart se može kompajlirati u mašinski kod za izabranu platformu ili JavaScript, podržava interfejse, apstraktne klase, generičke tipove i zaključivanje tipova na osnovu vrednosti promenljive. Po svojoj sintaksi Dart pripada porodici C programskih jezika. Dart ima mogućnost transpajliranja ("*source to source*" kompajlacije) u JavaScript, u kojem obliku se i izvršava unutar internet pretraživača. Pored internet pretraživača, postoji i Dart virtuelna mašina "DartVM", koja predstavlja okruženje u kojem se Dart može samostalno izvršavati.
Ovim je omogućeno izvršavanje i interakcija sa napisanim Dart programom preko komandne linije. Mogućnost prevremenog kompajliranja (*Ahead of Time Compilation – AOT Compilation eng.),* omogućuje Dart-u da se pre izvršavanja prevede u jezik nižeg nivoa čime se optimizuju performanse ovog jezika tokom izvršavanja [7]. Za pre-vremenu kompajlaciju, za različite operativne sisteme Dart koristi "dart2native" kompajler, pomoću kog se izvorni kod napisan u Dart-u transpajlira u odgovarajući mašinski kod[7]. Upravo uz pomoć ove funkcionalnosti, Dart programi napisani uz pomoć Flutter radnog okvira se mogu distribuirati kao *Native* aplikacije na veb prodavnicama zvaničnih distributera aplikacija za izabranu platformu pametnih telefona. "Play Store" za Android operativni sistem ili "AppStore" za Apple-ov iOS.

2.2. Arhitektura Flutter radnog okvira

Na površini, Flutter radnog okvira je reaktivni, pseudo-deklarativni radni okvir za pravljenje korisničkih interfejsa, u kome programer obezbeđuje mapiranje iz stanja same aplikacije i njenih podataka u stanje interfejsa, a radni okvir preuzima zadatak ažuriranja interfejsa u toku izvršavanja kada se stanje aplikacije promeni.

Slika 2. Dijagram slojevite arhitekture Flutter radnog okvira



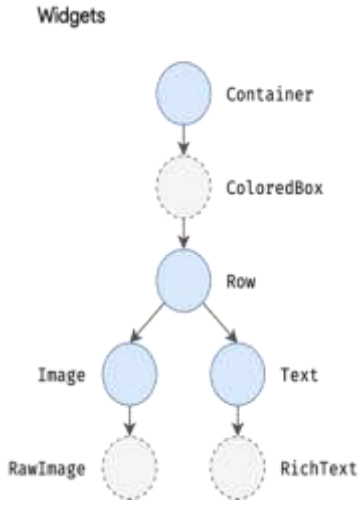
Flutter je dizajniran kao proširiv, slojeviti sistem i postoji kao niz međusobno nezavisnih biblioteka od kojih svaka biblioteka zavisi samo od sloja ispod. Nijedan sloj nema privilegovani pristup sloju ispod sebe, a svaki deo nivoa radnog okvira je dizajniran da bude opcioni i zamenljiv [6]. Dijagram spomenutih slojeva i biblioteka u ovoj sekciji prikazan je na slici **Slika 2** [8]. Iz perspektive operativnog sistema na kome se izvršavaju, Flutter aplikacije su upakovane na isti način kao i svaka druga *Native* aplikacija. Embeder, koji je specifičan za datu platformu, obezbeđuje ulaznu tačku u aplikaciju, takođe embeder zajedno sa operativnim sistemom platforme, reguliše pristup uslugama kao što su površine za iscrtavanje (ekran), pristupačnost (permisija) i unos podataka u aplikaciju, kao i upravljanje petljom događaja. Embeder je napisan na jeziku koji je odgovarajući za datu platformu: trenutno Java i C++ za Android, Objective-C/Objective-C++ za iOS i macOS i C++ za Windows i Linux. Koristeći embeder, Flutter kod se može integrisati u postojeću aplikaciju kao modul, ili može biti ceo sadržaj aplikacije.

Suštinski sloj Flutter-a je Flutter mašina, odnosno izvršno okruženje, koje je najvećim delom napisana u C++ programskom jeziku i podržava primitivne strukture i tipove podataka neophodne za podršku svih Flutter aplikacija. Flutter mašina je odgovorna za iscrtavanje kompozitnih scena kad god treba da se oslika novi ekran ili slika. Obezbeđuje implementaciju Flutter-ovog osnovnog API-ja na niskom nivou, uključujući grafiku (preko "*Skia*" grafičke biblioteke), izgled teksta, fajl i mrežni unos i izlaz podataka, takođe ima podršku za pristupačnost, arhitekturu dodataka (eng. *Plugins*), Dart mašine i lanac alata za kompajliranje. Flutter mašina, izložena je Flutter radnom okviru preko biblioteke zvane "*dart:ui*", koji umotava osnovni C++ kod u Dart klase. Ova biblioteka izlaže primitivne klase najnižeg nivoa, kao što su klase za pokretanje podsistema unosa, grafike i teksta.

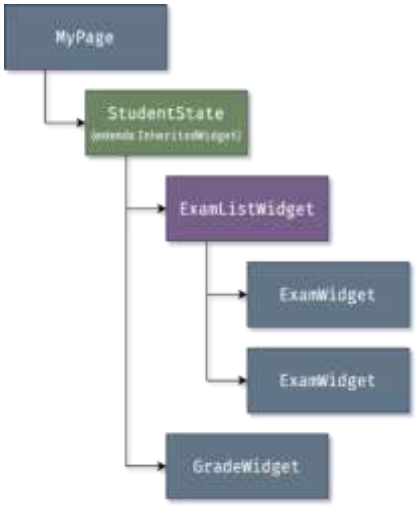
Poslednji sloj predstavlja sam Flutter Radni okvir, koji je napisan u Dart-u i preko kog uz pomoć osnovnih klasa programeri mogu da komuniciraju i pišu instrukcije za samu Flutter mašinu, odnosno za izvršno okruženje. Ovaj sloj sadrži biblioteke za animaciju, bojenje, registrovanje korisničkih pokreta i iscrtavanje samog stabla vidžeta (*widget tree eng.*). Krajnji proizvod primene ove arhitekture je skup vidžeta koji sačinjavaju "*Material*" ili "*Cupertino*" aplikaciju u zavisnosti od ciljanog ponašanja aplikacije. U Flutter-u, vidžeti, koji sačinjavaju gradivnu jedinicu korisničkog interfejsa, su predstavljeni nepromenljivim klasama koje se koriste za konfigurisanje stabla objekata. Ovi vidžeti se koriste za upravljanje odvojenim stablom objekata za raspored, koji se zatim koristi za upravljanje odvojenim stablom objekata za sastavljanje. Flutter je, u svojoj srži, niz mehanizama za efikasno korišćenje modifikovanih delova stabla, pretvaranje stabala objekata u stabla objekata nižeg nivoa i prikazivanje promena preko svih stabala [8].

3. SISTEM VIDŽETA

Vidžeti su gradivni blokovi korisničkog interfejsa (UI – *User Interface*) Flutter aplikacije, a svaki vidžet je nepromenljiva jedinica dela korisničkog interfejsa. Vidžeti formiraju hijerarhiju zasnovanu na kompoziciji. Svaki vidžet je ugnježden unutar svog roditelja vidžeta i može da primi kontekst okruženja od svog roditelja vidžeta. Ova struktura se prostire sve do osnovnog vidžeta. Osnovni vidžet je zapravo kontejner u kojem se nalazi sama Flutter aplikacija [8], obično *MaterialApp* na Android platformi ili *CupertinoApp* na iOS platformi. U Flutter-u, vidžeti, koji sačinjavaju gradivnu jedinicu korisničkog interfejsa, su predstavljeni nepromenljivim klasama koje se koriste za konfigurisanje stabla objekata. Dijagram jednog stabla vidžeta može se videti na **Slici 3.**



Slika 3. Primer Stabla vidžeta



Slika 4. Dijagram prosleđivanja stanja uz pomoć "Inherited Vidžet"-a

Vidžeti se obično sastoje od mnogih drugih malih, jednonamenskih vidžeta koji se kombinuju da bi proizveli moćne efekte. Gde god je to moguće, broj dizajnerskih koncepata je sveden na minimum, a istovremeno omogućava da ukupan rečnik bude veliki. Na primer, u sloju vidžeta, Flutter koristi isti osnovni koncept (vidžet) da predstavi crtež na ekranu, raspored (pozicioniranje i dimenzionisanje), korisničku interaktivnost, upravljanje stanjem, teme, animacije i navigaciju. Postoje vidžeti za dopunu, poravnanje, redove, kolone i mreže. Ovi vidžeti nemaju sopstveni vizuelni prikaz. Umesto toga, njihova jedina svrha je da kontrolišu neki aspekt izgleda drugog vidžeta. Flutter takođe uključuje uslužne vidžete koji koriste prednosti ovog kompozicionog pristupa. Na primer, "*Container*", najčešće korišćeni vidžet, sastoji se od nekoliko vidžeta odgovornih za raspored, slikanje, pozicioniranje i dimenzionisanje. Konkretno, kontejner se sastoji od vidžeta "*LimitedBox*", "*ConstrainedBox*", "*Align*", "*Padding*", "*DecoratedBox*" i "*Transform*", kao što možete videti čitajući njegov izvorni kod [8].

Odlikujuća karakteristika Flutter-a je da možete ući u deo koda za bilo koji vidžet i upoznati se sa njegovim načinom funkcionisanja. Hijerarhija klasa je namerno plitka i široka kako bi se maksimizirao mogući broj kombinacija, fokusirajući se na male vidžete koji se mogu sastaviti od kojih svaki dobro radi jednu stvar. Osnovne karakteristike su apstraktne, sa čak i osnovnim karakteristikama kao što su "*padding*" (prostor oko teksta) i poravnanje koje se implementiraju kao zasebne komponente, a ne ugrađene u jezgro. (Ovo je takođe u suprotnosti sa tradicionalnijim API-jima gde su funkcije kao što je "*padding*" ugrađene u zajedničko jezgro svake komponente rasporeda.) Tako, na primer, da bi se centrirao vidžet, umesto da se prilagođava zamišljeno svojstvo "*Align*", vidžet se može umotati u "*Center*" vidžet.

Mnogi vidžeti nemaju promenljivo stanje, jer nemaju svojstva koja se menjaju tokom rada aplikacije. Podklasa ovih vidžeta "*StatelessWidget*". Ako se jedinstvene karakteristike vidžeta moraju promeniti na osnovu interakcije korisnika ili drugih faktora, taj vidžet ima stanje. Na primer, ako vidžet ima brojač koji se povećava svaki put kada korisnik dodirne dugme, tada je vrednost brojača stanje za taj vidžet. Kada se ta vrednost promeni, vidžet treba ponovo da se izgradi da bi se ažurirao deo korisničkog interfejsa. Ovi vidžeti su podklase "StatefulWidget", (jer je sam vidžet nepromenljiv) oni čuvaju promenljivo stanje u posebnoj klasi koja je podklasa State klase. "StatefulWidget"-i nemaju metod izgradnje, umesto toga, njihov korisnički interfejs je izgrađen preko njihovog „State“ objekta[9].

Kad god se mutira objekat "State"(*Stanje eng.*) određenog vidžeta, mora se pozvati metoda "setState()" kako bi se signaliziralo radnom okviru da ažurira korisnički interfejs tako što će se ponovo pozvati metoda izgradnje State-a i time promeniti korisnički interfejs. Imajući odvojene objekte stanja i vidžeta, vidžeti tretiraju i vidžete bez stanja i vidžete sa stanjem na potpuno isti način, bez brige o gubitku stanja. Kako stabla vidžeta postaju dublja, prenošenje informacija o stanju gore-dole po hijerarhiji stabla postaje previše kompleksno. Radi smanjenja kompleksnosti uvodi se, treći tip vidžeta, "*InheritedWidget*", pruža jednostavan način za preuzimanje podataka od zajedničkog pretka [9]. Može se koristiti "*InheritedWidget*" kako bi se kreirao vidžet stanja koji obmotava zajedničkog pretka u stablo vidžeta, kao što je prikazano na **Slici 4**.

4. VIDŽETI U PROJEKTNOM ZADATKU

Projektni zadatak je osmišljen kao klijentska aplikacija za korisnike avio kompanije "ATVSS Airlines". Ideja je da kada korisnik postane mušterija ove zamišljene avio kompanije, dobije mogućnost da kroz aplikaciju vrši pregled i kupovinu dostupnih avionskih karata. Unutar same aplikacije takođe postoji mogućnost pretrage hotela i destinacija za putovanje, pomoću veb sajta kompanije "Tripadvisor", čiji su podaci dostupni unutar aplikacije pomoću "*url_launcher*" biblioteke koja je dostupna preko zvaničnog repozitorijuma biblioteka trećih partija za Flutter radni okvir "*pub.dev*"[10]. Funkcionalnost korisničkih naloga omogućena je pomoću Firebase-ovog servisa za autentikaciju korisnika, rad sa Firebase servisima je takođe omogućen pomoću biblioteke sa "*pub.dev*" repozitorijuma, pod nazivom "*firebase_auth*". Izgled određenih ekrana napravljene aplikacije, sa opisanim funkcionalnostima može se videti na slikama: **Slika 5**, **Slika 6**, **Slika 7.**

Slika 5. Početni ekran, sa Login funkcionalnošću, koji koristi Firebase auth. servis

Slika 6. "HomeScreen", omogućava pregled kupljenih karata i dostupnih hotela

Slika 7. "PurchaseTicketsScreen", omogućava pregled i kupovinu karata

Čuvanje podataka o kupljenim kartama omogućeno je preko "*shared_preferences*" biblioteke preuzete sa "*pub.dev*", repozitorijuma primer unošenja podataka može se primetiti na slici "**Slika 8**". Ovde se mogu primetiti metode koje se koriste za pristup i modifikaciju lokalne memorije telefona. Metoda "*getLocalData()*", je asinhrona metoda koja uz pomoć biblioteke "*shared_preferences*", može pristupiti podacima iz lokalne memorije mobilnog telefona. Metoda "*saveToLocalData(String key)*", koristi se za upis podataka u lokalnu memoriju, poslednja metoda "*noteTheBoughtTickets()*", na osnovu dostupnih podataka upravlja nizovima koji se koriste za čuvanje podataka o kupljenim kartama. Primer jednog takvog niza može se primetiti na **Slici 10**. ("*purchasedUpcomingTickets*"). Na slici **Slici 10**. se može videti kod koji opisuje grafički element aplikacije prikazane na **Slici 9.**
Kroz ovaj primer koda može se videti da je u projektnom zadatku primenjena teorija postavljanja vidžeta u određenoj formi stabla, gde se jedan vidžet smatra roditeljem ("*SingleChildScrollView*" vidžet), a unutar njega se mogu ugnježditi "deca" vidžeti koji u tom trenutku imaju ulogu jednog od elemenata unutar svog roditelja vidžeta. Ovo je praktični primer teorije objašnjene u sekciji 3. uz **Sliku 3.**

```
getLocalData() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  return prefs;
}

saveToLocalData(String key) async {
  if (localData != null) {
    await localData!.setBool(key, true);
    print("successfully saved to local data");
  }
}

noteTheBoughtTickets() {
  for (var ticket in availableTickets) {
    bool? tempBool = localData!.getBool("boughtTicketID" + ticket["id"]);
    if( tempBool != null && tempBool) {
      availableTickets.remove(ticket);
      print("removing a ticket from available tickets");
      purchasedUpcomingTickets.insert(0,ticket);
      print("Insertinga ticket into purchased tickets");
    }
  }
}
```

Slika 8. Metode korišćene za čuvanje podataka o kupljenim avionskim kartama

Slika 9. Grafički element koji prikazuje niz kupljenih avionskih karata

```
SingleChildScrollView(
  scrollDirection: Axis.horizontal,
  padding: const EdgeInsets.only(left: 20),
  child: Row(
    children:
      purchasedUpcomingTickets.map((e) => TicketView(ticket: e)).toList(),
  ), // Row
), // SingleChildScrollView
```

Slika 10. Kod koji opisuje funkcionalnost vidžeta prikazanog na prethodnoj slici

5. ZAKLJUČAK

Kroz ovaj rad čitaocu su prikazani teorijski principi koji se koriste sa Flutter radnim okvirom, takođe mogu se primetiti osnovna Dart sintaksna pravila, koja su primenjena u samom projektu. Kroz predstavljene teorijske

koncepte i praktične primere pružen je uvid u sposobnosti i prednosti primene Flutter radnog okvira za izradu hibridnih mobilnih aplikacija. Poznavanjem veb tehnologija programerima je omogućeno da pored standardnih veb stranica i veb aplikacija mogu da proizvode i *Native* aplikacije nezavisne od platforme za koju se prave.

Pružajući jednostavnost u procesu razvoja, visoke performanse u rezultujućoj mobilnoj aplikaciji, bogat i relevantan korisnički interfejs za Android i iOS platforme, Flutter radni okvir će sigurno omogućiti velikom broju programera da razviju mobilnu aplikaciju visokih performansi i puno funkcija koje zadovoljavaju kriterijume postavljenje od strane korisnika. Korišćenje sistema vidžeta za izgradnju korisničkih interfejsa, zajedno sa pseudo-deklarativnom paradigmom, koju koristi ovaj radni okvir, pokazalo se kao dobro rešenje za izgradnju korisničkih interfejsa pri izradi mobilnih aplikacija naročito, uzimajući i u obzir da ponašanje tih vidžeta i njihov izgled može da varira u zavisnosti od platforme na kojoj se aplikacija izvršava. Iako je Flutter od strane Google-a predstavljen kao skup alata koji služi za izgradnju korisničkih interfejsa, zajedno sa Dart programskim jezikom oni čine jako korisno okruženje za brz razvoj kvalitetnih klijentskih aplikacija. Flutter radni okvir pruža izvanredne mogućnosti time što omogućava odličan metod za pravljenje mobilnih aplikacija na način koji je zaista nezavisan od platforme.

LITERATURA

Arshad, W. (2021). *Managing State in Flutter Pragmatically*. Packt Publishing

Miola, A. (2020). *Flutter Complete Reference*. Independently published

Payne, R. (2019). *Beginning App Development with Flutter*. Apress

Senade, J., & Galloway, M. (2021). *Dart Apprentice (First edition): Beginning Programming with Dart Raywnderlich Tut.Team*. R R Bowker LLC

Zaccagnino, C. (2020). *Programming Flutter: Native, Cross-Platform Apps the Easy Way*. Pragmatic Bookshelf

Global stats Stat Counterm, https://gs.statcounter.com/platform-market-share/desktop-mobile/worldwide/#yearly-2011-2022, (visited on 20th September 2022)

European App Economy Report 2013 - Vision Mobile Research, https://actonline.org/wp-content/uploads/2013/07/The-European-App-Economy.pdf, (visited on 20th September 2022)

The App Economy in the European Union - Deloitte finance, branch of Deloitte network, https://actonline.org/wp-content/uploads/Deloitte-The-App-Economy-in-the-EU-2020.pdf, (visited on 20th September 2022)

https://developer.apple.com/xcode/ - Official Apple Developer website, (visited on 20th September 2022)

https://developer.android.com/studio - Official Android Developer website, (visited on 20th September 2022)

https://docs.flutter.dev/resources/architectural-overview - Official Flutter Documentation, (visited on 20th September 2022)