Praćenje fizičke aktivnosti i medicinskih terapija na mobilnim uređajima

Đerđ, Mislav

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

Preddiplomski studij

PRAĆENJE FIZIČKE AKTIVNOSTI I MEDICINSKIH TERAPIJA NA MOBILNIM UREĐAJIMA

Završni rad

Mislav Đerđ

Osijek, 2021

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 20.09.2021.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime studenta:	Mislav Đerđ
Studij, smjer:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3914, 26.07.2016.
OIB studenta:	06979539727
Mentor:	Izv. prof. dr. sc. Josip Balen
Sumentor:	
Sumentor iz tvrtke:	
Naslov završnog rada:	Praćenje fizičke aktivnosti i medicinskih terapija na mobilnim uređajima
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Predložena ocjena završnog	Vrlo dobar (4)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 1 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	20.09.2021.
Datum potvrde ocjene Odbora:	26.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

IZJAVA O ORIGINALNOSTI RADA

Osijek, 29.09.2021.

Ime i prezime studenta:	Mislav Đerđ
Studij:	Preddiplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	R3914, 26.07.2016.
Turnitin podudaranje [%]:	5

Ovom izjavom izjavljujem da je rad pod nazivom**: Praćenje fizičke aktivnosti i medicinskih terapija na mobilnim uređajima**
izrađen pod vodstvom mentora Izv. prof. dr. sc. Josip Balen

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

IZJAVA

o odobrenju za pohranu i objavu ocjenskog rada

kojom ja __Mislav Đerđ_____, OIB: __06979539727_____, student/ica Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek na studiju _Preddiplomski sveučilišni studij Računarstvo_, kao autor/ica ocjenskog rada pod naslovom: _Praćenje fizičke aktivnosti i medicinskih terapija na mobilnim uređajima**,**

dajem odobrenje da se, bez naknade, trajno pohrani moj ocjenski rad u javno dostupnom digitalnom repozitoriju ustanove Fakulteta elektrotehnike, računarstva i informacijskih tehnologija Osijek i Sveučilišta te u javnoj internetskoj bazi radova Nacionalne i sveučilišne knjižnice u Zagrebu, sukladno obvezi iz odredbe članka 83. stavka 11. *Zakona o znanstvenoj djelatnosti i visokom obrazovanju* (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15).

Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog ocjenskog rada. Ovom izjavom, kao autor/ica ocjenskog rada dajem odobrenje i da se moj ocjenski rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti

b) studentima/icama i djelatnicima/ama ustanove

c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem ocjenskom radu, podnosi se obrazloženi zahtjev nadležnom tijelu Ustanove.*

__Osijek, 29.09.2021._____ _____

(mjesto i datum) (vlastoručni potpis studenta/ice)

SADRŽAJ

1. UVOD

Mobilne aplikacije su veoma popularne u današnje vrijeme i trend broja preuzimanja mobilnih aplikacija je takav da svake godine raste. Svaka osoba koja posjeduje pametni telefon koristi njegove aplikacije. Trenutno dominiraju dva operacijska sustava, iOS i Android. Ukoliko mobilna aplikacija želi biti uspješna, ona mora biti dostupna na oba operacijska sustava. Klasičan razvoj aplikacija za spomenute operacijske sustave je takav da je potrebno napraviti dvije mobilne aplikacije koje rade istu stvar na specifičan način koji ovisi o sustavu. To rezultira povećanjem troškova koji su potrebni za razvoj takve aplikacije.

Osmišljena je mobilna aplikacija koja će omogućiti korisniku da prati svoju fizičku aktivnost, unos hrane i njezinih nutritivnih informacija te praćenje i obavještavanje korisnikovih medicinskih terapija. Cilj aplikacije je da korisniku nudi jasnu statistiku njegovih aktivnosti i statistiku praćenja unosa medicinske terapije i dodataka prehrani za trening.

Za izradu mobilne aplikacije korišten je više-platformski radni okvir Flutter. Flutter je modernije višeplatformsko razvojno okruženje koji nudi mnoštvo opcija s kojima se razvijaju aplikacije za iOS i Android na veoma brz način. Programski jezik pomoću kojeg je pisan programski kôd je Dart. Dart je objektno-orijentirano programski jezik. Kao bazu podataka korišten je izvan mrežni sustav *hive*.

1.1. Zadatak završnog rada

Cilj završnog rada je u teorijskom dijelu opisati tehnologiju za izradu hibridnih mobilnih aplikacija Flutter i programski jezik Dart. U praktičnom dijelu rada izraditi mobilnu aplikaciju koja će omogućiti automatsko praćenje fizičke aktivnosti korisnika, računanje unesenih kalorija i podsjetnike na terapije. Mobilna aplikacija treba omogućiti spremanje podataka u bazu podataka te kreiranje izvještaja.

2. PREGLED PODRUČJA TEME RADA

Na tržištu aplikacija nalazimo nekoliko sličnih aplikacija. Najpoznatija i najkorištenija aplikacija u kategoriji *Zdravlje i Fitnes* je FitBit. FitBit aplikaciju nalazimo na iOS i Android uređajima te se koristi najviše u svrhu praćenja fizičke aktivnosti. Glavna karakteristika FitBit aplikacije je njezina povezanost s istoimenim fitnes uređajima. MyFitnessPal je aplikacija koja se bavi praćenjem unosa hrane i poticanjem korisnika na zdraviji život. Prednost MyFitnessPal aplikacije je bar kôd čitač hranidbenih proizvoda i velika baza podataka hrane. Samsung Health, Google Fit i Apple Health su sve fitnes aplikacije pojedinih tvrtki koje se bave praćenjem korisnikovog zdravlja. Navedene aplikacije rađene se u Swiftu i Javi/Kotlinu. Medisafe je mobilna aplikacija na obje platforme koja se bavi praćenjem korisnikovih lijekova. Mobilna aplikacija završnog rada će pokriti neke od funkcionalnosti navedenih aplikacija. Ono u čemu će se jasno razlikovati od ostalih je tehnologija izrade pomoću radnog okvira Flutter i dodana funkcionalnost praćenja medicinskih terapija i dodataka prehrani.

2.1. Android operacijski sustav

Android je operacijski sustav otvorenog kôda zasnovan na modificiranoj jezgri Linux i stvoren od američke tvrtke Google. Ima velik broja aplikacija većinom napisane u Java programskom jeziku. Prvenstveno dizajniran za mobilne uređaje poput pametnih telefona i tableta, danas se nalazi i u televizorima, pametnim satovima, autima i hladnjacima.

2.2 iOS operacijski sustav

iOS je operacijski sustav kojeg je razvila američka tvrtka Apple Inc. za iPhone. Predstavljen je 2007. godine istovremeno s iPhoneom. Mobilne aplikacije na spomenutom sustavu su pretežito razvijene Swift programskim jezikom, koji je specifično napravljen za iOS sustav.

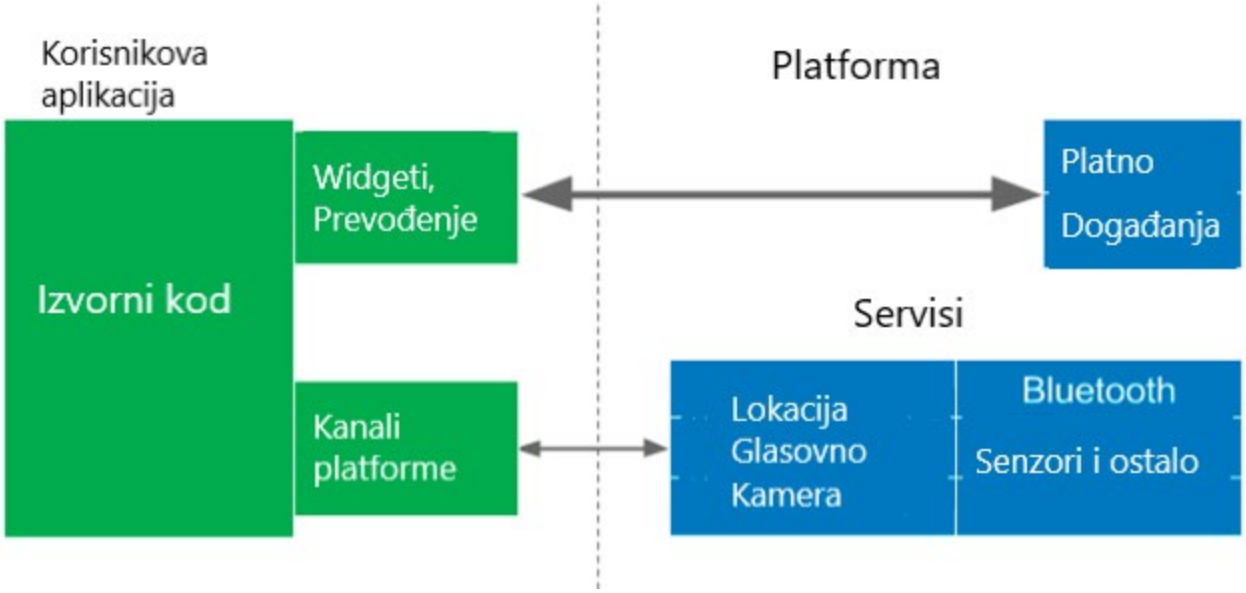
3. PREGLED KORIŠTENIH TEHNOLOGIJA

U ovome poglavlju se detaljno prikazuju i objašnjavaju programski slojevi radnog okvira Flutter, te prikazuju njegove prednosti, mane i različitosti od drugih višeplatformskih okruženja. Isto tako se objašnjava korišteni objektno orijentirani programski jezik Dart. Postavljen je fokus na sustav Flutter razvojnog okruženja, koji će biti detaljno opisan. Kao razvojno okruženje korišten je Microsoft Visual Studio(VS Code) koji nudi veliku prilagodbu korisnikovim željama i veoma je memorijski lagan za korištenje.

3.1. Radni okvir Flutter

Google definira Flutter kao set razvojnih alata otvorenog kôda s korisničkim sučeljem za izgradnju izvorno sastavljenih aplikacija za mobilne uređaje, web i radnu površinu iz jedne baze kôdova. Predstavljen je 2015. godine a službeno je pokrenut u prosincu 2018. godine. Trenutno je ugrađena podrška za Android i iOS mobilne platforme. Razvoju višeplatformskih aplikacija Flutter pristupa tako što pruža vlastiti skup objekata sučelja, iscrtavanja i motor koji implementira animaciju, grafiku, datoteke i mnoge druge programske knjižnice [1] .

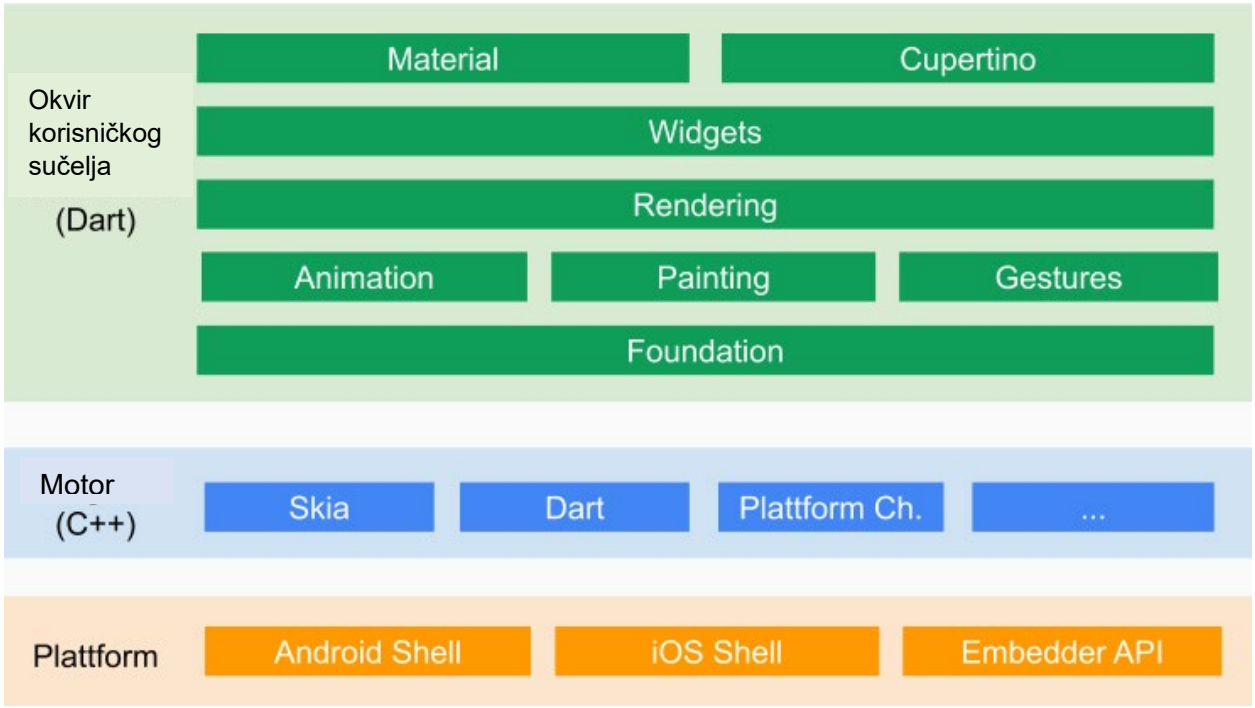
Arhitektura Flutter sustava sastoji se od više slojeva. Polazeći od visoke razine, glavna grafička komponenta korisničkog sučelja zove se Widget. Widgeti su građevni blokovi svih Flutter aplikacija. Na službenoj stranici, Google ističe brz razvoj, lijepo i fleksibilno korisničko sučelje, i odlične performanse na bilo kojoj platformi kao tri glavne prednosti u odnosu na slične radne okvire. Zbog takvih prednosti Flutter se ističe naspram drugih sličnih radnih okvira. Navedene prednosti nastale su zbog korištenja Dart jezika i implementiranja nove arhitekture koja koristi lijepe, brze, prilagodljive i proširive Widgete. Flutter sam kreira svoje grafičke komponente koje su identične onima iz izvornih(*engl. native)* aplikacija.



Slika 3.1 Arhitektura iscrtavanja Flutter aplikacija

Flutter prilazi arhitekturi iscrtavanja na način da se cijeli proces iscrtavanja prenese u aplikaciju. Rendering prolazi kroz svoj *engine* i koristi svoje Widgete koji se i dalje nalaze u aplikaciji. Na dijelu platforme nalazi se platno (engl. *canvas*) na kojem se iscrtavaju Widgeti kako bi se prikazali na ekranu uređaja, pristup događajima (poput dodira i tajmera) i uslugama (poput mobilne kamere, GPS senzora i slično).

Prijenosom većine radnja u aplikaciju Flutter je omogućio veću brzinu izvođenja zbog manje potrebe komunikacije s platformom preko kanala koji inače usporava aplikaciju. Mana takvog pristupa je da to utječe na datotečnu veličinu aplikacije u odnosu na izvorne poput Jave ili Kotlina na Androidu, i Swifta na iOSu. Najjednostavnija „Hello World!“ aplikacija u Javi ili Kotlinu na Androidu ima oko 0.5 MB-a, dok bi takva vizualno ista aplikacija napravljena pomoću Flutter okvira imala oko 5.5 MB-a veličinu datoteke [2].



Slika 3.2 Arhitektura Flutter sustava

Prema slici 3.2 arhitektura Flutter sustava raspodijeljena je u tri sloja:

1. Okvir korisničkog sučelja koji pišemo Programskim jezikom Dart
2. Engine sloj smješten ispod sloja okvira
3. Sloj ugradbenog sučelja (engl. *Embedder*)

Flutter je dizajniran kao rastegljivi, slojeviti sustav. Postoji kao niz neovisnih knjižnica od kojih svaka ovisi o temeljnom sloju. Nijedan sloj ima privilegiran pristup donjem sloju, dok je svaki dio razine okvira osmišljen kao zamjenjiv i opcionalan. Navedeni slojevi sustava su objašnjeni u idućim poglavljima.

3.2. Programski jezik Dart

Prilikom odabira programskog jezika za radni okvir Flutter bilo je razmatrano desetak jezika. JavaScript je bio prvi izbor i nakon godinu dana razvijanja projekta Flutter tim se odlučio na promjenu jezika u Dart. Dart je objektno-orijentirani programski jezik razvijen od tvrtke Google koji se koristi u razvijanju Flutter mobilnih aplikacija. Sintaksa jezika slična je stilu

mnogim programskim jezicima kao što su C++, Java i C# te zbog toga većina programera Dart nauči brzo [3].

Jedna od posebnosti Fluttera je značajka „vrućeg ponovnog učitavanja“ (engl. *hot reload*). Ta značajka omogućuje programerima uvid u trenutne izmjene u kôdu na simulatorima i emulatorima, bez ponovnog pokretanja aplikacije. Ta karakteristika je rezultat Dart VM-a (engl. *Virtual Machine*) i njegovog različitog načina rada u Debug i Release modu. Dart VM je skup komponenti za izvorno izvršavanje Dart kôda. Pri „načinu za uklanjanje pogrešaka“ (engl. *Debug mode*) sposoban je raditi u JIT(*engl. Just in time)* načinu kompilacije koji dinamički učitava i prevodi izvorni kôd. S time se olakšava značajka *hot reload* i uklanjanje pogrešaka te ubrzava proces razvoja mobilne aplikacije.

U „načinu otpuštanja“ (engl. *Release mode*), Dart je sastavljen preko AOT(*engl. Ahead of time)* prevoditelja. AOT prevoditelj omogućava prevođenje u kôd niske razine za produkcijske verzije aplikacije što znatno poboljšava performanse. Također, Dart ima *garbage collector* koji omogućava brzo brisanje starih i tvorbu novih objekata. Rezultat toga su glatke animacije i stalna brzina od 60 okvira u sekundi (engl. *frames per second*) [4]. Dobro je napomenuti kako postoje slučajevi u kojima će se aplikacija trebati kompletno restartirati kako bi vidjeli promjene na simulatorima. Neki od slučajeva su mijenjanje fonta, pisanje kôda u izvornim jezicima, promjene vrijednosti u lijenom instanciranju, promjena modela objekta i generiranje JSON(*engl. JavaScript Object Notation)* formata radi povezivanja s API(*engl. Application programming interface)* klijentom i slično [5].

3.3. Sloj okvira korisničkog sučelja

Najviši sloj Fluttera je Dart UI „okosnica“ (engl. *Framework*). Taj sloj je najjasniji i najrelevantniji programerima mobilnih aplikacija jer sadrži sve s čime će programer komunicirati prilikom razvoja aplikacije. Sloj sadrži teme, iscrtavanja i Widgete te *dart:ui.* U ovom sloju programer koristi Dart programski jezik koji mu omogućava korištenje Flutter sučelja. Podsloj okosnice, *dart:ui* služi kao temelj za knjižnicu i iscrtavanja Widgeta i on upravlja komunikacijom sve do Flutter *engine.*

Slika 3.3 - Podjela Dart UI sloja

Framework dolazi sa već napravljenim *Cupertino* (za iOS) i *Material* (za Android) paketima. Ti paketi sadrže elemente korisničkog sučelja, boje i geste koje su jedinstvene temi ili specifičnoj platformi. U većini slučajeva će se koristiti jedna tema jer obje platforme dijele veći broj ponašanja i izgled. Kako bi aplikaciju napravili da izgleda izvorno (engl. *native*), lagano rješenje bi bilo da se provjeri na kojoj platformi se nalazi aplikacija i zatim se pošalje jasno određen Widget/Tema za tu platformu. Provjera mobilnog operacijskog sustava se koristi s metodom *Platform.isIOS*(ili *isAndroid)*.

Widgeti su elementi koji utječu i upravljaju s izgledom i sučeljem aplikacije. Bilo koji tekst, slika, tipka ili animacija u aplikaciji jest Widget. Čak je i sama aplikacija Widget. Uz već velik broj dokumentiranih Flutter Widgeta, oni se mogu i prilagođavati. Glavna podjela Widgeta je na *Stateful* i *Stateless*. Najčešći tip Widgeta je *Stateless*. Njih karakterizira nepromjenjivost bilo kojih veličina nakon inicijalizacije. Stateless Widget se najčešće sastoji od *build* metode prikazane u slijedećem isječku kôda. Kako *build* metoda može biti više puta pozvana u jednoj sekundi, praksa je da bude što lakša u smislu veličine zadatka. Flutter tim preporučuje da se *Stateless* Widgeti koriste gdje god je moguće [6].

```
class MyWidget extends StatelessWidget {
  //Zove se više puta u sekundi
  //Treba biti što lakši za izvođenje
  //
  @override
  Widget build(BuildContext context) {...}
}
```

Programski kôd 3.3.1 - Primjer *Stateless* Widgeta

Stateful widgeti su odgovorni za čuvanja promjenjivih stanja u Flutter aplikaciji. Sastoje se od dva dijela: nepromjenjivi widget objekt i promjenjivi objekt stanja. Obično se instanciraju preko ugradnje konstruktora. Widget je odgovoran za inicijalizaciju promjenjivog objekta i skladištenje stanja. Objekt stanja se zadržava u memoriji i nakon promjene u aplikaciji. Njegov ciklus je duži i složeniji nego widgetov. Ciklusi nude razne mogućnosti pozivanja metoda u Widgetu. Metoda *initState* je prva metoda zvana nakon kreiranja Widgeta te je često korištena. *setState* metodu često pozivamo u kôdu kada želimo obnoviti Widget novim podacima ili ažurirati ekran. Metoda *dispose* je korištena kada se objekt urušava, te se većinom koristi kao brisač predmemorije(*engl. cache memory)*. Pruža funkciju inicijalizacije, *build* metodu i *dispose* metodu kao što je prikazano u sljedećem isječku kôda. Po preporuci Flutter tima, *Stateful widgeti* se trebaju što manje koristiti iz razloga boljeg izvođenja. Postoje prvenstveno tri razloga korištenja *Stateful widgeta* nad *Stateless widgetima*:

1. Widget treba čuvati podatak koji će se tijekom radnog vijeka mijenjati.
2. Widget treba napraviti neku vrste inicijalizacije na početku svog vijeka trajanja.
3. Widget treba očistiti memoriju nakon svog vijeka trajanja [7].

```
class MyWidget extends StatefulWidget {
  //Poziva se odmah prilikom građe prvog Stateful widgeta
  @override
  State<StatefulWidget> createState() => _MyState();
}

class _MyState extends State<MyWidget>{
  //Poziva se prilikom kreiranja
  @override
  void initState() { }

  //Zove se više puta u sekundi
  //Treba biti lagan za izvođenje
  //Ovdje se gradi korisničko sučelje
  @override
  Widget build(BuildContext context) {...}

  //Zove se jednom prije završetka seanse(gašenje aplikacije)
  dispose() {}
}
```

Programski kôd 3.3.2 - Životni vijek *Stateful* widgeta [8]

3.4. Sloj motora

Smješten ispod *framework* sloja, Flutterov Engine je prijenosni *runtime* koji poslužuje aplikaciju. Preuzima osnovne tehnologije Dart *runtime*, Skia (2D grafička knjižnica otvorenog kôda), datotečni i mrežni I/O, podršku za pristup i ostale osnovne knjižnice. Sloj koristi programske jezike C i C++ koji su prevođeni na iOS i Android platforme prikladnim tehnologijama prevođenja za pojedinu platformu. Dart UI *framework* i izvorni kôd aplikacije sastavljani su u ARM knjižnice i delegirani su za obradu petlje događaja, prikazivanje i druge zadatke. Takva implementacija slična je implementaciji u video igrama i pruža kvalitetan korisnički doživljaj.

API (napisan u C-u) ne sadrži ovisnosti o platformama i može se pronaći na GitHubu. Imajući Engine koji nema ovisnost o temeljnoj platformi omogućava stvaranje prilagođenih ugrađivača za platforme koje nisu dio proizvoda. Pošto je Flutter otvorenog kôda, postoje knjižnice koje pružaju Windows, Linux i macOS podršku za Flutter okvir [9].

3.5. Sloj platforme

Na razini platforme (iOS ili Android), Flutter pruža *Shell*, softverski program koji interpretira naredbe od korisnika kako bi ih sustav mogao razumjeti i izvršiti odgovarajuće funkcije. Shell, specifičan pojedinoj platformi, pruža pristup API-ima i uspostavlja platno karakteristično platformi. Ljuska, isto tako pomaže kod komunikacije s tipkovnicama.

Pristup svim API-ima specifične platforme nije ugrađeno u sam radni okvir, jer bi se time povećala veličina Fluttera i stvorilo bi se mnogo *bugova*. Jedan od glavnih izazova s kojima se susreću rješenja za više platformi okruženja je omogućavanje da aplikacijski kôd direktno komunicira s izvornim API-jevima. Flutter rješava taj problem preko kanala platforme. *Platform channel* funkcionira na principu slanja i primanja poruka bez generacije kôda. Komunikacija je dvosmjerna i asinkrona. Slojevi više razine su iOS/Android neovisni jer se sva važna međudjelovanja s izvornim API-ima, kao pristup platnu, odvijaju na ovom sloju [10].

3.6. Nedostatak Fluttera

Iako se čini da je Flutter budućnost razvoja aplikacija s kontinuiranim razvojem i specifičnim značajkama, postoje ograničenja koja se trebaju poboljšati. Knjižnice i Widgeti trećih strana prilično su mali. Štoviše, neki Widgeti su dostupni samo na jednoj operacijskoj platformi. Nedostatak *code pusha*, mogućnosti da odmah stavimo zakrpe u već izdanu aplikaciju, čime se štedi vrijeme i rad. Kako Flutteru nedostaje ta mogućnost, sva ažuriranja moraju proći kroz tradicionalni postupak izdavanja nove verzije aplikacije, što može potrajati danima za iOS. Na GitHubu stoji više od 7000 tisuća neriješenih problema otvorenih od strane programera. Još jedna mana bi bila već spomenuta datotečna veličina aplikacije koja je veća negu u Javi ili Kotlinu.

4. IMPLEMENTACIJA APLIKACIJE

U sklopu završnoga rada izrađena je aplikacija za brojanje dnevnih koraka, praćenje unosa hrane, i osobni podsjetnik za lijekove. Također, implementiran je i pregled povijesti unosa lijekova poredan po datumu. Aplikacija je razvijena s tehnologijom Flutter te ju je moguće pokrenuti na iOS i Android mobilnim uređajima. Dizajn svih ekrana je tematski sličan i koristi iste boje, fontove i animacije [11].

Implementirana aplikacija sastoji se od četiri glavna ekrana:

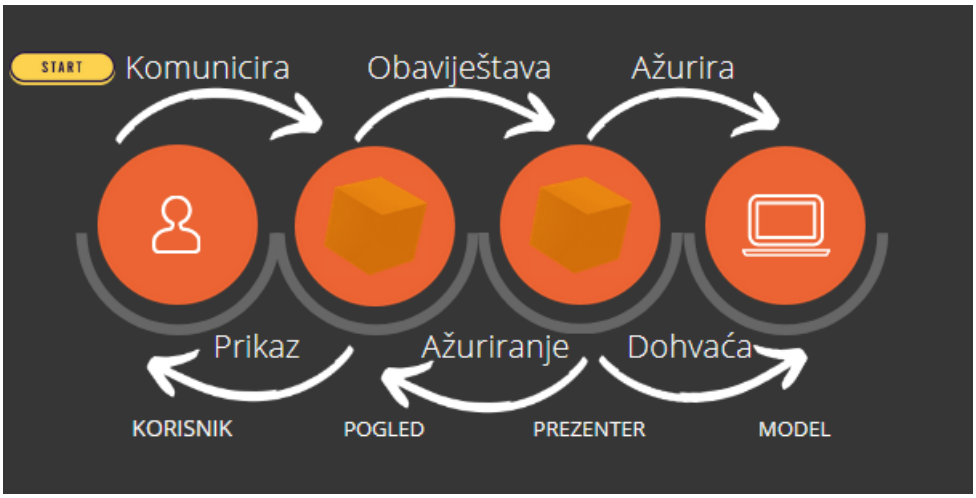
1) Uvodni ekran
2) Ekran brojača koraka
3) Ekran unosa hrane
4) Ekran podsjetnik za terapiju

Uvodni ekran sadrži sažete informacije o ostalim ekranima sklupčane u pojedine kartice. Pritiskom na pojedinu karticu dovodi nas do novih ekrana. Ekran brojača koraka sastoji se od pregleda dnevno napravljenih koraka, pregleda koraka od prošlih dana, izračunatu potrošnju kalorija, i gumb za dodavanje dnevnog cilja. Ekran unosa hrane sastoji se od pregleda nutritivnih vrijednosti unosa hrane po danu i gumba koji pritiskom na njega otvara manji ekran (sporedni) u kojemu korisnik može unositi hranu. Ekran podsjetnik za terapiju sadrži pregled prošlih terapija i kazalo koje prikazuje da li je korisnik konzumirao lijek, te dodatan gumb koji nas vodi do manjeg ekrana za dodavanje novih podsjetnika.

Važna komponenta aplikacije su notifikacije za podsjećanje na medicinske terapije. Korisnik odabire vrijeme i upisuje tekst lijek koji će biti prikazan u notifikaciji. Pritiskom na pridošlu obavijest se otvara aplikacija koja zabilježi lijek kao odrađen.

Kao arhitekturu projekta korišten je Model-View-Presenter arhitektura(skraćeno MVP). U MVP-u, *Presenter* sadrži poslovnu logiku korisničkog sučelja za *View*. Svi pozivi iz *View*a delegiraju se izravno prema *Presenteru* koji je odvojen od *View*a ali razgovora s njim preko sučelja. Zajednički atribut MVP-a je postojanje mnogo dvosmjernog slanja. Na primjer, kada korisnik pritisne gumb „Spremi“ voditelj događaja delegira metodu *Presenteru* koji komunicira s modelom i dobiva odgovor. Nakon što je metoda završena, *Presenter* zove *View* kroz sučelje i *View* obnavlja ili ažurira sučelje i s time pokazuje da je metoda dovršena. Arhitektura programa

predstavlja apstrakciju sustava koju programeri mogu koristiti kao osnovu za međusobno razumijevanje i identičnu komunikaciju u projektu. [12]



Slika 4.1 - Vizualan prikaz MVP arhitekture [13]

Spremanje podataka je ostvareno *hive* paketom, laganom i vrlo brzom NoSQL(*eng. Non structured query language)* bazom podataka zasnovano uparivanjem preko vrijednosti ključa i šifrirana s AES-256 standardom. U sklopu projekta napravljeno je 3 modela koja su definirana klasama. Model *step_count, food* i *medicine*. Za upravitelj stanja u aplikacije korišten je Provider, jednostavan menadžer stanja koji funkcionira kao omotač oko naslijeđenog Widgeta. Kada se izvede određena metoda i obavi promjena u bazi podataka, poziva se metoda *notifyListeners* koja obavještava naslijeđeni Widget omotan oko Providera. Spremanje primitivnih tipova u *hive* je lagano. Samo se upari ključ i doda npr. *integer*. Kada želimo spremiti složeniji tip podatka, poput objekta *medicine* modela, moramo ručno indeksirati svaku varijablu modela. Nakon što je model indeksiran, korištenjem *pub run build_runner build* naredbe u terminalu, dart će izgenerirati adapter datoteku koja će biti jasna *hive* paketu.

```
@override
void write(BinaryWriter writer, Medicine obj) {
  writer
    ..writeByte(3)
    ..writeByte(0)
    ..write(obj.dateTime)
    ..writeByte(1)
    ..write(obj.medicine)
    ..writeByte(2)
    ..write(obj.dose);
}
```

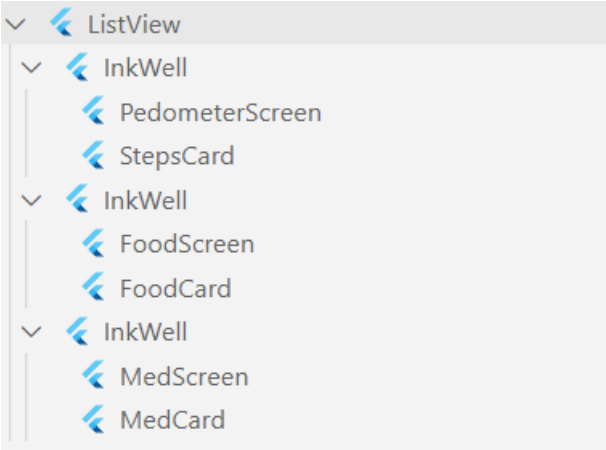
Programski kôd 4.1. – Automatski izgeneriran kod koji *hive* može pročitati

```
Hive.registerAdapter(StepCountAdapter());
Hive.registerAdapter(MedicineAdapter());
Hive.registerAdapter(FoodAdapter());
```

Programski kôd 4.2. – Registriranje *hive* adaptera

4.1. Implementacija uvodnog ekrana

Uvodni ekran se iscrtava prilikom otvaranja aplikacije. U gornjem dijelu ekrana je smješten naslov ekrana. Ispod njega se nalazi kutija čiji su kutovi kružno obrubljeni pomoću svojstva *borderRadius*. Kutija prikazuje ikonu notifikacije te podsjetnik za idući lijek. Glavni dio ekrana su tri kartice. Pritiskom na svaku karticu vodi nas na drugi ekran. Svaka kartica sadrži sliku koja prikazuje što korisnik može očekivati pritisne li karticu. Također, kartica sadrži tekst koji je povezan s usklađenim ekranom te tako prikazuje glavnu informaciju tog ekrana kako bi korisniku bilo jednostavnije pratiti. Kartice su poredane preko *ListView* widgeta u stupce. Kao svojstvo svake kartice, korištena je klasa *BoxShadow* koja baca sjenu na rubove čineći karticu podignutom. Rezultat toga je poboljšano korisničko iskustvo jer se kartica čini dodirljivom. U lijevi kut kartice je umetnuta manja kartica druge boje koja korisniku daje do znanja da je interaktivna.



Slika 4.2. – Stablasti raspored Widgeta na uvodnom ekranu

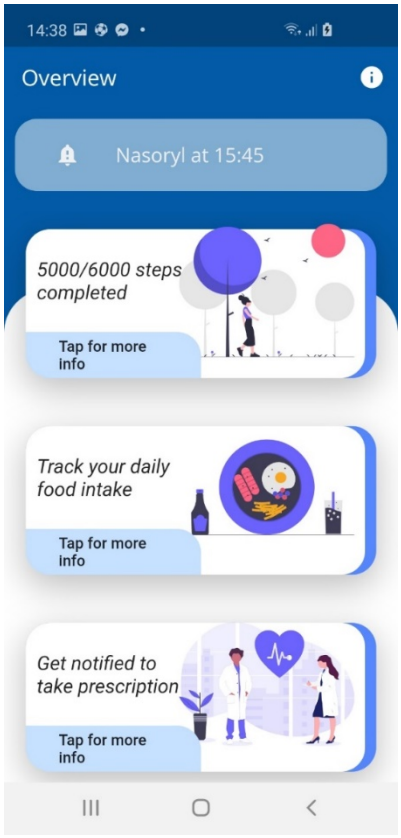
Za prikaz slike koristila se komponenta *SvgPicture* uvezena iz programske knjižnice *flutter_svg*. Spomenuta knjižnica služi za iscrtavanje vektorske grafike iz datoteka tipa *SVG(engl. Scalable vector graphics)*, koja se nalazi na svim ekranima. Prikaz takvih datoteka zauzima manje memorijskog prostora i same slike se bolje skaliraju nego bitmap ili png formati.

```
            '/AddFoodScreen': (context) => AddFoodScreen(),
            '/AddMedicineScreen': (context) => AddMedicineScreen(),
            '/FoodScreen': (context) => FoodScreen(),
            '/OverviewScreen': (context) => OverviewScreen(),
```

Programski kôd 4.3. – Navigacija u aplikaciji preko imenovanih ruti

Mobilne aplikacije svoj sadržaj obično otkrivaju putem elemenata na cijelom zaslonu koji se nazivaju „*screens*“ ili „*pages*“. U Flutteru se ti elementi nazivaju *routes* i njima upravlja widget *Navigator*. Navigator upravlja hrpom objekata *routes* i nudi dva načina za upravljanje hrpom, deklarativni i imperativni. Navigacija na druge ekrane se obavlja preko *pushNamed* metode u koju kao argument upisujemo jedno od imena ruta iskazanih u primjeru programskog kôda 4.3. Vraćanje na prošli ekran obavlja se s pozivom Navigator.*pop*(). Na uvodnom ekranu postoje dvije stvari koje su promjenjive, idući podsjetnik za lijek i broj koraka u danu. Provjeru da li postoji dolazeća notifikacija se vrši dolaskom na uvodni ekran bilo putem otvaranja aplikacije ili prelaska s drugog ekrana. Zovemo metodu *showScheduledMedicine* koja preko Providera zove *getNextMedicineInfoIfScheduled* metodu. Ta metoda otvara *hive* kutiju s važećim ključem i uzima sve podatke te ih privremeno sprema u varijablu. Nakon toga provjerava objekte da li postoji datum i vrijeme koje još nije uslijedilo. Ako postoji, uzima taj objekt i vraća ga

uvodnom ekranu. Ako ne postoji, vraća *string* s obrazloženjem. Na sličan način ponaša se i brojač koraka, zove metodu koja preko Providera zove metodu koja gleda u lokalnu bazu podataka broj koraka i vraća odgovor.



Slika 4.3. Izgled početnog ekrana implementirane aplikacije

4.2. Implementacija ekrana brojača koraka

Na ekran brojača koraka dolazimo pritiskom kartice koraka na uvodnom ekranu. Gornji dio ekrana se sastoji od naslova, ikone koja simbolizira šetanje, broj današnjih ostvarenih koraka (iskazano i u kilometrima) te potrošene kalorije. Velik dio ekrana zauzima SVG slika šetanja koja je identična onoj s uvodnog ekrana. Donji dio ekrana sastoji se od liste *ListView* widgeta, retka fiksne visine koji obično sadrži neki tekst, kao i vodeću i zadnju ikonu. Time je implementiran jasan pregled broja koraka i potrošene kalorije prošlih dana.

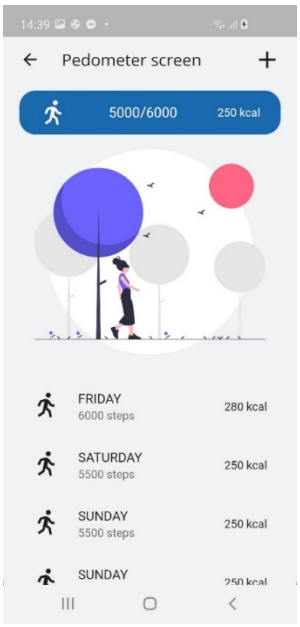
Model *step_count* sadrži dvije varijable, datumska vrijednost koja se sprema u tip *DateTime* i broj koraka koji se sprema u primitivni tip *integer.* Objekti tog modela se spremaju u *hive* kutiju gdje su poredani u listu.

```
@HiveType(typeId: 0)
class StepCount {
  @HiveField(0)
  DateTime dateTime;
  @HiveField(1)
  int steps;

  StepCount({
    this.steps,
    this.dateTime,
  });
}
```

Programski kôd 4.4. – Definiranje StepCount modela

Mjerenje koraka implementirano je s *pedometer* dodatkom koji omogućuje kontinuirano brojanje koraka pomoću već ugrađenih API senzora u iOS i Android mobilnim uređajima. Dobiveni broj koraka predstavlja ukupan broj učinjenih koraka od zadnjeg podizanja sustava. Na ekranu se obavlja pozadinska provjera unesenosti broja koraka za današnji dan, i pošto je dobiven ukupan broj koraka, on se oduzima od broja koraka prethodnog dana i ispisuje na predviđeno mjesto. Ako se u međuvremenu mobilni telefon ugasio te ponovno pokrenuo, događa se slučaj kada je ukupan broj koraka manji od broja koraka prošlog dana. Tada se prošli dan zanemaruje i upisuje se ukupan broj koraka za taj dan.



Slika 4.4. Izgled ekrana broja koraka

4.3. Implementacija ekrana unosa konzumirane hrane

Na ekran praćenja dnevno konzumirane hrane dolazimo pritiskom na karticu hrane s uvodnog ekrana. Gornji dio ekrana se sastoji od naslova ekrana, ikone koja simbolizira hranu, te kartice s natpisom „Report“. U gornjem desnom rubu se nalazi ikona hrane koja nas pritiskom na nju, prebacuje na sporedan ekran u kojemu korisnik upisuje i putem interneta dobiva tražene podatke hrane. Dio ekrana zauzima i slika hrane koja je identična onoj s uvodnog ekrana. Donji dio ekrana sastoji se od *ListView* Widgeta koji prikazuje ikonu jela, ukupnu kaloričnu vrijednost konzumiranu u jednom danu i količinu predmeta. Pritiskom pločice proširuje se ili urušava pogled njegove djece. Njegova djeca su *ListTile* widgeti koji prikazuju ikonu jela, tekst unesene hrane i kaloričnu vrijednost hrane.

Model *food* sadrži mnogo varijabli koje čine nutritivne informacije o proizvodu, kao i datum unosa i ime proizvoda. Objekti *food* modela se spremaju u *hive* kutiju u listu. Dolaskom na ekran hrane se zove metoda *getFoodList* koja otvara kutiju i uzima sve objekte te vraća objekte kao listu. Ekran se ažurira i prikazuje sve dobivene objekte kroz Widget *ListTile*. U navedenom programskom kôdu *_boxName* je *string* konstanta koja sadrži ključ otvaranja određene kutije.

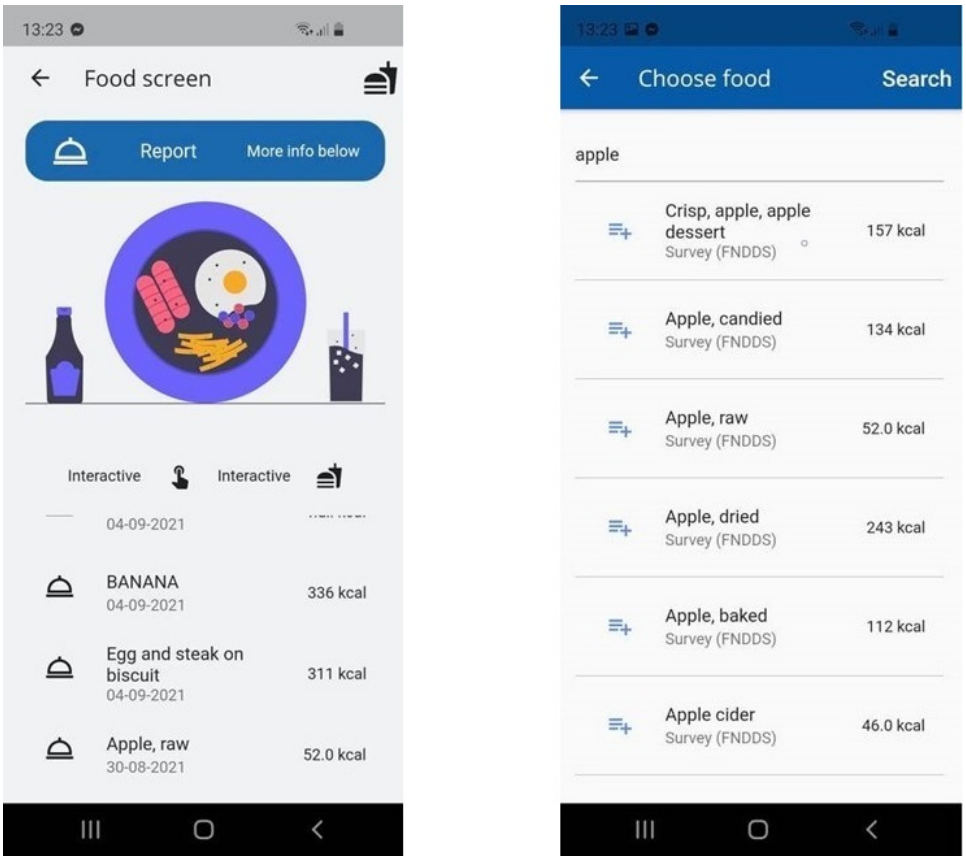
```
Future<List<Food>> getFoodList() async {
    var box = await Hive.openBox<Food>(_boxName);

    _foodList = box.values.toList();
    return _foodList;
  }
```

Programski kôd 4.5. – Metoda dohvaćanja liste *food* objekata iz hivea

Pritiskom na „Report“ karticu prikazuje se dijalog koji uzima sve unesene podatke čiji datum je identičan s datumom pokretanja dijaloga. Dijalog se sastoji od zbroja svih nutritivnih vrijednosti filtriranih podataka. U slučaju da za taj dan ništa nije uneseno, korisnik biva obaviješten.

Već spomenuti sporedni ekran služi kao tražilica hrane i može se koristiti ako korisnik ima omogućenu vezu s internetom. Dolaskom na ekran, automatski se otvara tipkovnica uređaja i korisnik može upisati traženu hranu. Dodana je provjera broja upisanih slova u tražilicu. Tražilica je omogućena ako su najmanje 3 slova upisana zbog kvalitete pretrage. Ukoliko je korisnik upisao manje od 3 slova u pretragu i pritisnuo na gumb traženja, na ekranu se pojavljuje *FlutterToast* Widget koji nagovještava korisniku da je najmanji broj upisanih slova tri. Nakon što je korisnik upisao više od dva slova i pritisnuo gumb „Search“ ili „Done“ u mobilnoj tipkovnici, poziva se metoda *searchFood* s argumentom korisnikovog upisa. *searchFood* metoda šalje argument API-u koji uparuje vrijednosni ključ i okida poziv prema bazi podataka. Ako se korisnik odluči vratiti na prošli ekran, to može učiniti s dodanim gumbom za vraćanje ili s već ugrađenim „nazad“ gumbom. Kada korisnik odlazi s ekrana, kontroler koji je pratio upisivanje teksta treba biti odložen [14]. U Flutteru se to postiže naredbom *dispose* koja je osnovni dio životnog ciklusa Flutter Widgeta.



Slika 4.5. – Izgled ekrana praćenja hrane i tražilice hrane

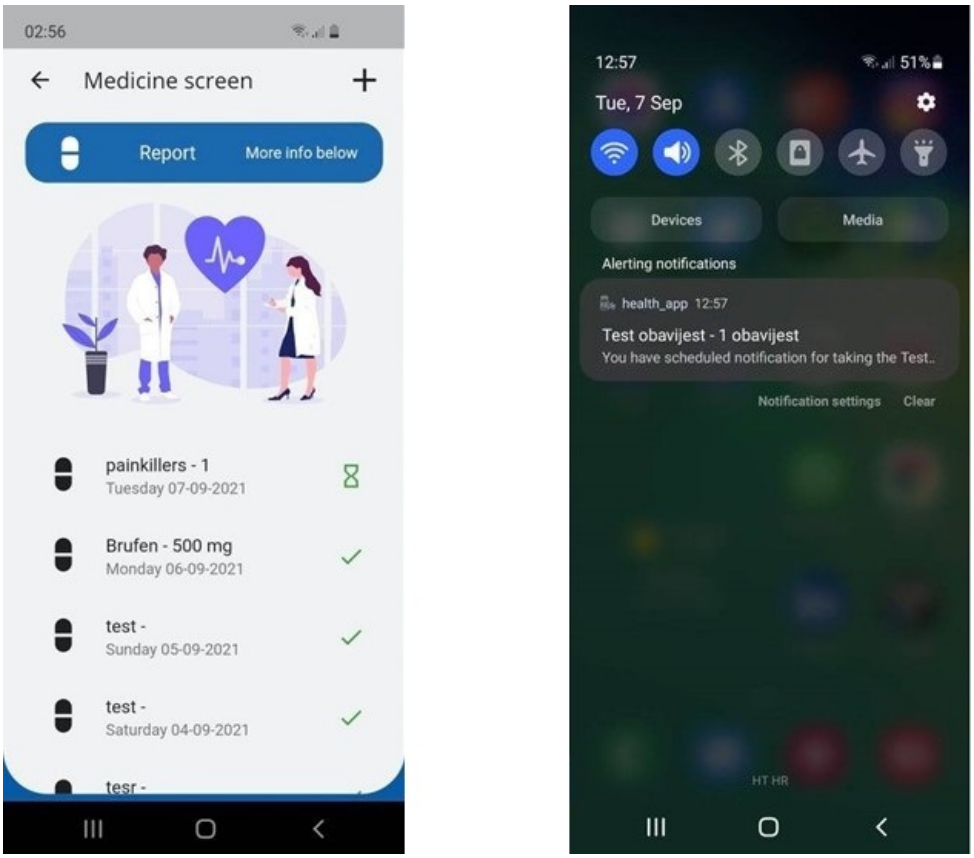
Tražilica hrane je implementirana uz pomoć *FoodData Central* API paketa i njihove baze podataka s preko milijun prehrambenih proizvoda. Korišten je POST poziv koji u glavnici ima URL(*engl. Uniform Resource Locator)* i vrijednosni ključ koji dopušta pogled u bazu, dok tijelo pôziva sadrži unesenu vrijednost od strane korisnika aplikacije. Nakon što je korisnik unio vrijednost(npr. čokolada) i potvrdio svoj odabir, okida se mrežni poziv koji nakon nekog vremena vraća 20 najsličnijih rezultata unesene vrijednosti. Rezultati sadrže razne detaljne informacije o tom proizvodu, poput naziva marke, broja kalorija po 100 grama, ugljikohidrata, proteina i slično. U slučaju da se poziv izveo a nakon nekog vremena se i dobio odziv s nekim od HTTP(*engl. Hypertext transfer protocol)* kôdova grešaka poput 400, 404, 500 i slično koji mogu sugerirati kako korisnik nije imao upaljenu mrežnu vezu ili se vrši održavanje datotečne baze, u donjem dijelu ekrana će se prikazati dijalog s nagovještajem korisniku gdje je nastao problem.

Nakon što su korisniku prikazani rezultati pretrage, omogućene su dvije funkcionalnosti za svaki vraćeni objekt. Uz svaki Flutterov Widget dolazi poneka funkcionalnost, tako i *ListTile* ima nekoliko, od kojih su dvije korištene. Pritiskom na jedan od 20 rezultata odrađuje se metoda koja je uparena na svojstvo *onTap*. U ovom slučaju se prikazuje centrirani dijalog postavljen

preko ekrana koji sadrži informacije pritisnutog rezultata poput imena, energije u kalorijama, količine ugljikohidrata i ostalih nutritivnih informacija. Tada korisnik ima dvije opcije: dodati hranu u lokalnu bazu ili se vratiti na sporedan ekran gdje može odabrati drugi rezultat. Drugo korišteno svojstvo je *onLongPress* gdje korisnik nakon dužeg pritiska rezultata sprema odabranu hranu u lokalnu bazu i vraća se na glavni ekran hrane gdje je odabrana hrana vizualno spremljena u listu.

4.4. Implementacija ekrana medicinskih terapija

Na ekran medicinskih terapija dolazimo pritiskom na zadnju karticu u uvodnom ekranu. Gornji dio ekrana se sastoji od naslova ekrana i medicinske kartice. U gornjem desnom rubu se nalazi ikona znaka plus koja nas pritiskom na nju, prebacuje na sporedan ekran u kojem korisnik dodaje lijekove za koje želi da ga aplikacija podsjeti. Donji dio ekrana sastoji se od liste *ListTile* widgeta koji prikazuje detalje o spremljenim podacima u lokalnu bazu podataka. Model *medicine* ima 3 svojstva: zakazani datum konzumiranja lijeka, ime lijeka i količina unesene medicine. Objekti modela se spremaju u *hive* kutiju u listu. Podatci koje pločica prikazuje su: ikona tablete, ime terapije, vrijeme uzimanja terapije i stanje koje ovisi o vremenu da li se terapija dogodila ili će se dogoditi. Ovakvim prikazom korisnik jasno može vidjeti povijest uzimanja medicine i zakazane buduće tretmane.



Slika 4.6. – Izgled ekrana medicinske terapije i dobivene obavijesti

Hijerarhija dizajna ekrana je takva da je sve prvo omeđeno *column* Widgetom, koji svoju djecu poreda od gore prema dolje. Svaki ekran možemo konfigurirati već ugradbenim *Scaffold* Widgetom, koji dolazi s napravljenim podijeljenim stavkama ekrana poput glavnice, tijela ekrana i podnožja. Glavnica u ovom slučaju posjeduje gumb za nazad, ime ekrana i gumb za prelazak na sporedan ekran dodavanja podsjetnika. Prvo dijete je „Report“ kartica, ispod toge je SVG pozadinski okvir i slika. Ispod slike je *ListView* widget koji gradi slične widgete u listu. Kao argument koliko će se puta ponoviti građa widgeta, *ListView* dobije od zvanja metode *Provider.of<MedicineData>(context).medicineListLength.*

```
Container(
                  height: size.height * 0.45,
                  width: size.width * 0.9,
                  alignment: Alignment.center,
                  child: ListView.builder(
                    reverse: true,
                    scrollDirection: Axis.vertical,
                    shrinkWrap: true,
                    itemBuilder: (context, index) {
                      return MedicineTile(tileIndex: index);
                    },
                    itemCount: Provider.of<MedicineData>(context).medicineList
Length,
                  ),
```

Programski kod 4.6. – Donji dio ekrana

Sporedan ekran sastoji se od tri polja gdje korisnik unosi ime terapije, dozu ili količinu terapije i vrijeme kada želi da ga se obavijesti. Ime terapije i vrijeme su obvezna polja za unos i ukoliko korisnik pritisne gumb za spremanje podsjetnika dobit će odgovor kako treba unijeti polja koja su obvezna. Datum i vrijeme se unosi tako što korisnik pritisne gumb i otvori se *DatePicker* Widget gdje korisnik odabire vrijeme i datum, bez korištenja tipkovnice. Pritiskom na gumb „Remind“, okine se metoda *_addMedicine* koja provjerava popunjenost polja i ukoliko su polja popunjena, preko *Providera* zove *addMedicine* metodu koja vrijednosti sprema u *hive* kutiju. Nakon završetka funkcije, korisnik biva obaviješten kako je podsjetnik terapije uspješno dodan. Paket *flutter_local_notifications* omogućava prikaz i zakazivanje lokalnih obavijesti na iOS i Android operativnim sustavima.

```
//localNotifications
  var initializationSettingsAndroid = AndroidInitializationSettings('medicine'
);

  var initializationSettingsIOS = IOSInitializationSettings(
    requestSoundPermission: true,
    requestBadgePermission: true,
    requestAlertPermission: true,
    onDidReceiveLocalNotification: (int id, String title, String body, String
payload) async {},
  );
  var initializationSettings = InitializationSettings(
    initializationSettingsAndroid,
    initializationSettingsIOS,
  );
  await flutterLocalNotificationsPlugin.initialize(
    initializationSettings,
    onSelectNotification: (String payload) async {
      if (payload != null) {
        debugPrint('notification payload: ' + payload);
      }
    },
  );
```

Programski kôd 4.7. – Inicijalizacija paketa za lokalne obavijesti

Android aplikacija ne traži odobravanje pristupa notifikacijama, dok iOS traži. Ovaj dio kôda je obavezan jer u suprotnom obrada obavijesti dok je aplikacija ugašena ne bi radila. Android aplikacija koristi menadžer alarma(*engl. Alarm Manager)* za posao raspoređivanja obavijesti, dok se u iOS aplikaciji to obavlja preko kreiranja *UNNotificationRequest* objekta. Iako programski jezik Dart koristimo kao svojstvo pisanja jednog kôda za oba operacijska sustava, stvari poput reguliranja obavijesti iziskuju dodatne provjere operacijske platforme korisnikovog mobitela i unikatno rješenje za svaku.

5. ZAKLJUČAK

Iako je Flutter relativno mlad radni okvir, na mene je ostavio dobar dojam. Lakoća razvoja jednostavnih aplikacija je vrlo brza zbog razvojnih alata i *Hot Reload* značajke. Uz sve to, jedan kôd i za Android i za iOS aplikaciju znatno smanjuje vrijeme potrebno za razvitak aplikacije. Nove verzije Fluttera poput 2.2 okrenule su se prema optimizaciji *web* platforme. Idući korak nadogradnje aplikacije bi bio da se sve podigne na internetsku cjelinu preko *Firebase* platforme i napravi web dio, gdje bi korisnik mogao pratiti i upisivati vrijednosti preko web preglednika.

Izrada završnog rada zahtijevala je proučavanje i istraživanje radnog okvira Flutter, te odabir pogodne arhitekture. Nadalje, praćenje pravila čistog i održivog programskog kôda i odvajanje korisničkog sučelja od funkcionalnosti same aplikacije. Mobilna aplikacija za praćenje fizičke aktivnosti i medicinskih terapija na mobilnim uređajima olakšava korisnicima praćenje konzumirane hrane, daje informacije o postignutom broju koraka po danu i obavještava korisnika kada je vrijeme za medicinsku terapiju.

LITERATURA

[1] Flutter, https://flutter.dev/, 2021

[2] Rap Payne, Beginning App Development with Flutter: Create Cross-Platform Mobile Apps, Apress, Dallas, Texas, 2019

[3] "Flutter: How we're building a UI framework for tomorrow at Google" by Eric Seidel, https://www.youtube.com/watch?v=VUiVkDpikDI

[4] W. Leler, 'What's Revolutionary about Flutter', hackernoon, https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514, 2021

[5] Hot Reload, https://flutter.dev/docs/development/tools/hot-reload, 2021

[6] Flutter Dev Team, 'Stateless Widget class', https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html, 2021

[7] Dart Team, 'Performance best practices', https://flutter.dev/docs/testing/best-practices, 2021

[8] Flutter example of many lifecycle states in stateful widget, https://github.com/flutter/flutter/blob/master/examples/layers/services/lifecycle.dart, 2021

[9] The Engine Architecture, https://github.com/flutter/flutter/wiki/The-Engine-architecture, 2021

[10] Writing Custom Specific Code, https://flutter.dev/docs/development/platform-integration/platform-channels, 2020

[11] Steve Krug, Don't Make Me Think, revisited: a common sense approach to Web usability third edition, New Riders Press, San Francisco, California, 2013

[12] Software Architecture in Practice, second edition Why is Software Architecture important?, https://people.ece.ubc.ca/matei/EECE417/BASS/ch02lev1sec4.html, 2021

[13] Presentation Model, https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff921080(v=pandp.20), 2021

[14] Joshua Block, Effective Java second edition, Addison-Wesley, Stoughton, Massachutes, 2009

SAŽETAK

Cilj ovog rada je istražiti radni okvir Flutter kao moguće rješenje kod izrade višeplatformskih mobilnih aplikacija. Ovaj rad pruža teorijski pregled razvojnog okruženja i predočavanje njegovih prednosti i mana. U prvom dijelu su objašnjeni platformski slojevi Flutter okvira i Dart programski jezik. U praktičnom dijelu rada osmišljena je višeplatformska aplikacija kroz koju je demonstriran proces izrade pojedinih komponenti i objašnjena logika razvoja same aplikacije. Napravljena aplikacija koja prati dnevnu fizičku aktivnost, daje korisniku mogućnost praćenja dnevno konzumirane hrane i podsjetnike za unos medicinskih terapija je brza, robusna i u aspektu dizajna moderna, uz čiju će pomoć korisnik jasno i lako pratiti svoj dnevni izvještaj.

Ključne riječi: Android, flutter, medicinska terapija, višeplatformski okvir, widget

ABSTRACT

MONITORING PHYSICAL ACTIVITIES AND MEDICAL THERAPIES ON MOBILE DEVICES

The goal of this assignment was to explore Flutter framework as a possible solution in the development of multi-platform mobile applications. This paper provides a theoretical overview of the development environment and a presentation of its advantages and disadvantages. The first part explains the platform layers of the Flutter framework and the Dart programming language. In the practical part of the assignment, a multi-platform application was designed through which the process of making individual components was demonstrated and the logic of the development of the same applications was explained. The created application that monitors daily physical activity, gives the user the ability to track daily food consumption and reminders to input medicine therapy is fast, robust, and modern in terms of design, with which the user will clearly and easily track his daily report.

Keywords: Android, flutter, medicine therapy, multi-platform framework, widget

ŽIVOTOPIS

Mislav Đerđ rođen je 09.11.1997. u Osijeku. Osnovnu školu je pohađao u Dardi. Nakon završetka osnovne škole upisuje Prvu gimnaziju Osijek, opći smjer. 2016. godine polaže maturu, te iste godine upisuje preddiplomski sveučilišni studij na Fakultetu elektrotehnike, računarstva, i informacijskih tehnologija Osijek, smjer računarstvo. Tijekom visokoškolskog obrazovanja pokazuje interes za programiranje mobilnih i web aplikacija. Trenutno radi kao programski developer u IT tvrtki u kojoj razvija Android, iOS, i web aplikacije. Posjeduje određeno znanje o programskim jezicima Dart i C# te znanje o razvijanju Android i iOS aplikacija.

Mislav Đerđ