Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

Horvat, Ivan

Undergraduate thesis / Završni rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Virovitica University of Applied Sciences / Veleučilište u Virovitici**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:165:532633

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Veleučilište u Virovitici

Repository / Repozitorij:

Virovitica University of Applied Sciences Repository - Virovitica University of Applied Sciences Academic Repository

zir.nsk.hr

dabar

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

VELEUČILIŠTE U VIROVITICI

Ivan Horvat

Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

ZAVRŠNI RAD

Virovitica, 2022.

VELEUČILIŠTE U VIROVITICI

Preddiplomski stručni studij Računarstva, smjer Programsko inženjerstvo

Ivan Horvat

Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

ZAVRŠNI RAD

podnesen Veleučilištu u Virovitici
radi stjecanja akademskog zvanja
stručnog prvostupnika
računarstva

Virovitica, 2022

Veleučilište u Virovitici

Preddiplomski stručni studij Računarstva - Smjer Programsko inženjerstvo

OBRAZAC 1b

ZADATAK ZAVRŠNOG RADA

Student/ica: **IVAN HORVAT** **JMBAG:** **0307016658**

Imenovani mentor: **Marko Hajba, mag. math., pred.**

Imenovani komentor: **-**

Naslov rada:

Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

Puni tekst zadatka završnog rada:

Izraditi mobilnu aplikaciju koja služi za administraciju osobnih i poslovnih zadataka i rutina. Za izradu aplikacije potrebno je koristiti Firebase kao backend tehnologiju, Flutter i Dart kao frontend, Firebase Cloud Storage za spremanje datoteka i Cloud Firestore za bazu podataka. Bazu je potrebno napuniti testnim podacima.

Mobilna aplikacija treba imati sljedeće funkcionalnosti.

- Omogućiti registraciju i prijavu korisnika.
- Administratori brinu o održavanju funkcionalnosti aplikacije, primaju prijedloge i rješavaju probleme koje mogu prijaviti ostali korisnici.
- Korisnici mogu kreirati ažurirati privatne i poslovne zadatke i rutine. Potrebno je omogućiti zadavanje ponavljanja zadataka i rutina po želji –npr. dnevno, tjedno, mjesečno, godišnje. Kreiranje i ažuriranje zadataka i rutina se vrši putem kalendara, gdje su sve obveze vizualno prikazane. Potrebno je omogućiti dodavanje lokacije i dokumenata za određeni zadatak.
- Aplikacija treba prikazati napredak korisnika u izvršenim zadacima.

Osim programskog rješenja, u pisanom dijelu završnog rada opišite ukratko korištene tehnologije te detaljno opišite arhitekturu sustava i odabrane procese i/ili funkcije unutar same aplikacije. Prilikom opisivanja, osim neformalnih koristite i neke od formalnih metoda koje poznajete.

Datum uručenja zadatka studentu/ici: 28.07.2022.
Rok za predaju gotovog rada: 09.09.2022.

Mentor:

Marko Hajba, mag. math., pred.

Marko Hajba

Dostaviti:
1. Studentu/ici
2. Povjerenstvu za završni rad - tajniku

VELEUČILIŠTE U VIROVITICI **Završni rad**

Temeljna dokumentacijska kartica

Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

Sažetak

Aplikacija je napravljena za dvije platforme Android i iOS koristeći jedan kod. Aplikacija omogućuje korisnicima organizaciju osobnih i poslovnih zadataka i rutina. Dodavanje i ažuriranje zadataka i rutina odvija se kroz jednostavnu formu, a dostupan je i njihov kalendarski prikaz. Glavni problem predstavljalo je preklapanje osobnih i poslovnih zadataka. Ponavljanje zadataka je varijabilno tako da se može odrediti ponavljanje za dan, tjedan, mjesec ili godinu. Moguće je dodati podsjetnik kod kreiranja zadataka koji putem push obavijesti podsjeća na nadolazeće zadatke. Rutine sadrže navike, a navike su jednostavna vrsta zadataka koja se ponavlja svaki dan. Statistika predstavlja osobnu mjeru uspješnosti osobnih i poslovnih zadataka. Korisnici mogu slati zahtjeve, pohvale i pritužbe administratorima, što omogućava daljnji razvoj aplikacije.

(26 stranica, 15 slika, 14 referenci)

Ključne riječi: Firebase, Flutter, mobilna aplikacija, navika, rutina, zadaci

Mentor: Marko Hajba, mag.math., pred.

VIROVITICA UNIVERSITY OF APPLIED SCIENCES **Bachelor's Thesis**

Basic documentation card

Mobile application for organizing personal and business tasks and routines

Abstract

The application is made for two platforms Android and iOS using only one codebase. The application allows users to organize personal and business tasks and routines. Adding and updating tasks and routines is done through simple form and can be displayed from calendar view. The main problem was overlapping of personal and business tasks. Recurrence of tasks is variable and can be determined for a day, week, month or year. It is possible to add reminder when creating tasks that reminds you of upcoming tasks via push notifications. Routines contains, and habits are simple types of tasks that are repeated every day. Statistics is a personal measure of the success of personal and business tasks. Users can send requests, compliments and complaints to administrators, which enables further development of the application.

(26 pages, 15 figures, 14 references)

Keywords: Firebase, Flutter, mobile application, habbit, routine, tasks

Supervisor: Marko Hajba, mag. math., lecturer

Sadržaj

Sadržaj ..... *v*

1. Uvod *1*

2. Operacijski sustavi *2*

2.1. Android **2**

2.1.1. Android arhitektura 2

2.2. iOS operacijski sustav **4**

2.2.1. iOS arhitektura 4

3. Flutter *7*

3.1. Widget **7**

3.2. Flutter arhitektura **9**

4. Firebase *11*

5. Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina *13*

5.1. Arhitektura aplikacije **13**

5.2. Baza podataka **16**

5.3. Prijava u aplikaciju **17**

5.4. Registracija korisnika **18**

5.5. Zaslon s rutinama **19**

5.6. Zaslon sa zadacima **23**

6. Zaključak *26*

6. Popis literature *27*

7. Popis slika *28*

1. Uvod

Mobilni telefon je u početku služio za primanje i upućivanje poziva. Porastom zahtjeva korisnika, tvrtke su počele uvoditi inovacije i mijenjati načina na koji živimo. Počeli su se proizvoditi mobilni telefoni sa zaslonom na dodir s pregršt funkcija nazvani kao pametni telefoni. Pametni telefoni koriste operacijski sustav za rad, a dva najpoznatija konkurentna operacijska sustava su Android i iOS. Statcounter proveo je istraživanju u kolovozu 2022. godine i zaključio da je Android tržišno najprodavaniji operacijski sustav sa 71,54%, koji znatno nadmašuje iOS, koji je drugi najpopularniji operativni sustav na globalnoj razini sa 27,81% [1]. Mobilne aplikacije omogućuju proširivanje primjene mobilnih uređaja. Na primjer, mobilno bankarstvo omogućuje slanje novaca drugim osobama, plaćanje računa i praćenje troškova, a organiziranje vremena uz pomoć kalendara omogućuje planiranje i samim time efikasnije izvođenje posla.

Klasični pristup izrade mobilnih aplikacija je zasebno pisanje koda za ciljanu platformu, takve aplikacije ujedno se zovu nativne aplikacije. Moderne aplikacije koriste višeplatformski pristup odnosno koriste isti programski kod za izvršavanje na raznim platformama. Flutter je nastao u svibnju 2017, dok je prva stabilna verzija izašla 4. prosinca 2018. godine. On nudi rješenje za izradu višeplatformskih aplikacija. Za razlika od nativnih aplikacija Flutter omogućuje lakše održavanje, smanjeno vrijeme izrade aplikacija i konzistentan dizajn kroz platforme.

2. Operacijski sustavi

2.1. Android

Android je operativni sustav za mobitele temeljen na Linuxu koji je razvio Google. Najpopularniji je i najčešće korišten mobilni operativni sustav s više od 2.6 milijuna aplikacija dostupnih na Google Play Storeu. Android se, osim u mobilnim uređajima, koristi u pametnim satovima, televizorima i automobilima [1]. Otvorenog je koda pa ne postoje ograničenja za što se može koristiti

2.1.1. Android arhitektura

Arhitektura Androida sadržana je u šest slojeva za podršku potrebama bilo kojeg Android uređaja – slika 1. Biblioteke su napisane u C/C++ programskom jeziku, ali nativne aplikacije razvijaju se u Java i Kotlin programskom jeziku.

Temelj Android platforme je Linux jezgra (engl. *Linux kernel*) na kojoj se temelje ostali slojevi ujedno je i najniži sloj. Osnovne funkcije koje pruža jesu upravljanje memorijom i dretvom. Dretve služe za paralelno izvršavanje zadaća unutar jednog procesa. Sadrži ključne sigurnosne značajke: izolaciju procesa, proširivi mehanizam za sigurnu međuprocesnu komunikaciju i mogućnost uklanjanja nepotrebnih dijelova Linux kernela [2]. Linux kernel sloj pruža apstrakciju na hardveru. Sadržani su upravljački programi (engl. *Drivers*) kao što su upravljački programi za zaslon, upravljački programi za kameru, upravljački programi za Wi-Fi, upravljački programi za zvuk i mnogi drugi.

Hardverski sloj apstrakcije (engl. *Hardware Abstraction Layer*) sadrži module biblioteka i osnovan je na Linux kernelu. Svaki modul implementira sučelje za određenu vrstu hardverske komponente iz najnižeg sloja. Kada se podnese zahtjev za pristup hardveru uređaja sustav Android učitava modul biblioteke za zatraženu hardversku komponentu.

Nativne C/C++ biblioteke (engl. *Native C/C++ Libraries*) je sloj koji sadrži izvorne biblioteke napisane u C i C++ programskom jeziku. Pruža API okvire (engl. *API framework*) preko kojih viši slojevi komuniciraju s nižim slojevima. Neki primjeri svrha biblioteka su pristup bazi podataka, implementiranje web poslužitelja u aplikaciju, komunikaciju između aplikacija, prikazivanje.

Izvršavanje Android aplikacije (*ART,* engl. *Android runtime*) je sloj dizajniran da uređaj može neometano raditi s malo radne memorije. Potreban je Android 5.0 ili noviji, kako bi se

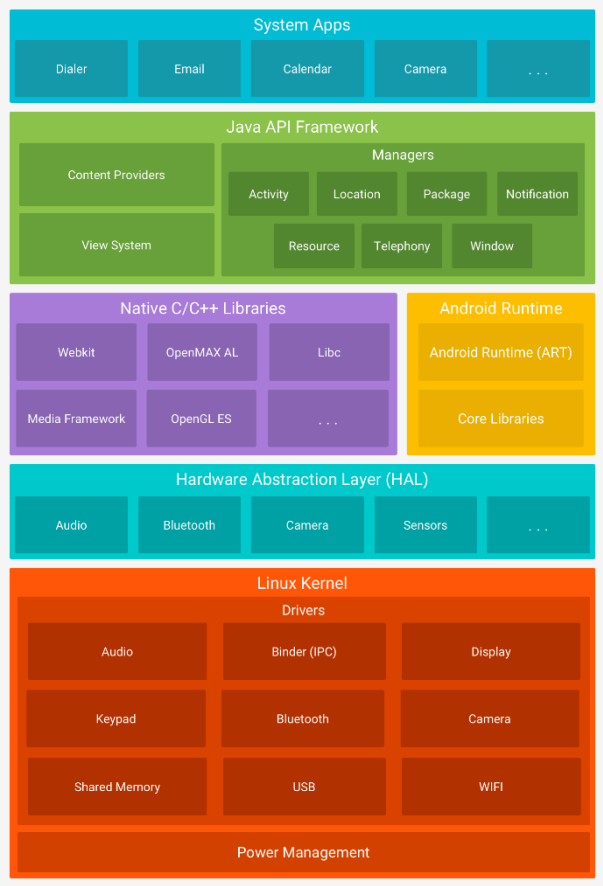
aplikacije mogle pokretati u zasebnom procesu što pospješuje rad platforme. ART je virtualni stroj koji izvršava DEX datoteke[1]. Brine se o oslobađanju resursa koji se ne koriste (engl. *garbage collection*), druga glavna uloga je otkrivanje i otklanjanje pogrešaka.

Java razvojno okruženje (engl. *Java API Framework*) sloj sadrži servise koji se pristupaju putem API-ja napisanih u Java programskom jeziku. Kod izrade nativne aplikacije koriste se servisi iz ovog sloja. Servisi su:

- Upravitelj resursa (engl. *Resource Manager*): pristupa datotekama unutar aplikacije
- Upravitelj obavijestima (engl. *Notification Manager*): omogućuje prikaz i kreiranje obavijesti
- Upravitelj aktivnostima (engl. *Activity Manager*): upravlja životnim ciklusom aplikacije
- Pružatelji sadržaja (engl. *Content providers*): Omogućava pristup podataka iz drugih aplikacija, npr. pregled slika iz galerije.
- View sustav (engl. *View system*): koristi se za iscrtavanje i interakciju korisničkog sučelja aplikacije

Najviši sloj jesu aplikacije sustava (engl. *System Apps*). To su osnovne aplikacije koje dolaze s operacijskim sustavom, npr. kalendar, pozivi, kalkulator, web preglednik, poruke i drugi.

[1] DEX datoteka – binarni zapis koda aplikacije



Slika 1. Android arhitektura [13]

2.2. iOS operacijski sustav

iOS (engl. *iPhone Operating System*) je drugi najpopularniji mobilni operacijski sustav poslije Androida. Razvijen je od strane Apple Inc. Može se koristiti samo na uređajima tvrtke Apple [4]. Ima zatvoreni ekosustav koji pruža stabilnost za različite Apple uređaje kao što su iPhone, iPad i iPod. Zatvoreni ekosustav znači da kod nije javno dostupan za pregled i uređivanje.

2.2.1. iOS arhitektura

Isto kao Android, iOS arhitektura je slojevita. Slika 2 prikazuje iOS arhitekturu sačinjenu od: aplikacijskog sloja (engl. *Cocoa Touch*), sloja upravljanja medijima (engl. *Media*), sloja servisa (engl. *Core Services*), sloja operacijskog sustava (engl. *Core OS*) te upravljačkih programa uređaja i jezgre (engl. Kernel and Device Drivers).

Najniži sloj upravljački programi uređaja i jezgre služi za upravljanje hardvera. Jezgra upravljanja temeljena je na *Mach 3.0* [2] koji omogućuje izoliranje jezgre te pristup datotečnim sustavima i upravljanje mrežom [5].

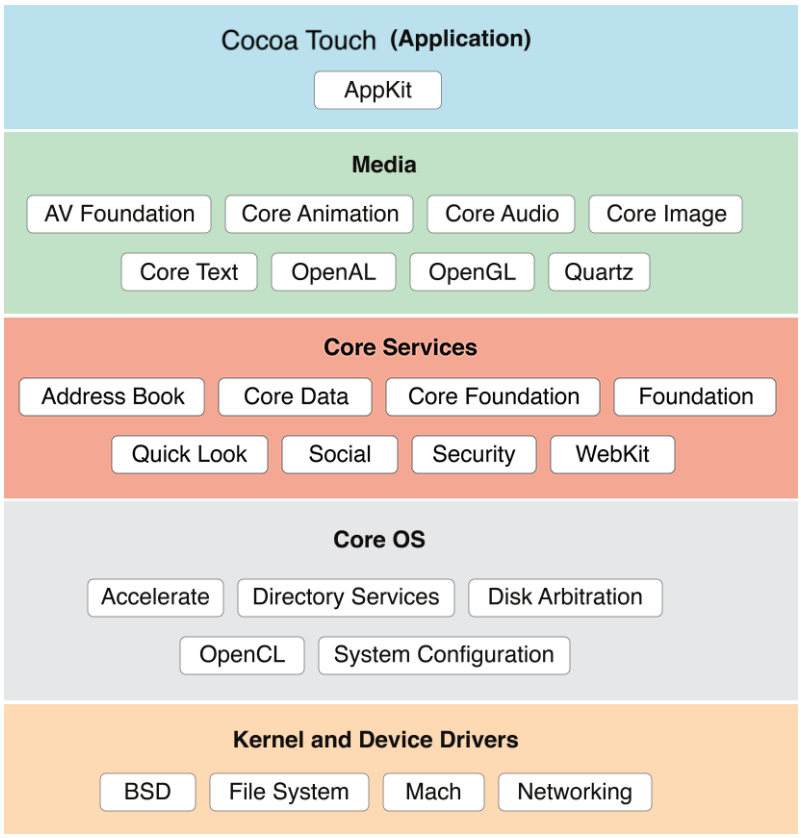
Sloj operacijskog sustava sastoji se od tehnologija i okvira koji pružaju servise za upravljanje hardverom i mrežom. Na ovom sloju izvode se aplikacije te kako je zatvoreni sustav, nije otvorenog koda zbog toga mnoge biblioteke se ne mogu koristiti zbog zaštite ekosustava. Biblioteke na koje se može pristupiti jesu *Accelerate*, *Directory services*, *Disk Arbitratzion*, *OpenCL* i *System Configuration*.

Sloj servisa je sloj koji se nadovezuje na operacijski sloj. Sadrži dodatne biblioteke koje pospješuju izradu aplikacije, te omogućuje pristup resursima potrebnim za normalan rad aplikacije. Primjerice, biblioteka *Social* služi za dohvaćanje podataka o korisniku iz raznih društvenih mreža.

Sloj upravljanja medijima služi za reproduciranje i prikazivanje medija. Prikazuje 2D i 3D grafike, animacije te reproducira audio i video sadržaj.

Aplikacijski sloj prvenstveno je odgovoran za izgled aplikacije. Omogućuje pristup glavnim servisima poput kamere, *push* obavijesti, kontakata. Sadrži elemente za izgradnju korisničkog sučelja.

[2] Mach – proširiva komunikacijska mikrojezgra



Slika 2. iOS arhitektura [14]

3. Flutter

Flutter je Googleov razvoji okvir (engl. *framework*) otvorenog koda za izradu nativno kompiliranih aplikacija za više platformi iz jedne baze koda [6]. Flutter je po prvi puta bio dostupan 2017. godine, dok je 2018. godine izašla prva stabilna verzija. Razlika između Androida, iOS i Fluttera je to što su je Flutter razvojni okvir za izradu višeplatformskih aplikacija, a Android i iOS su platforme.

React Native, *Xamarin*, i *Ionic* su još neki razvojnih okviri za razvijanje višeplatformskih aplikacija. Glavna razlika između Fluttera i nabrojanih razvojnih okvira je da se Flutter ne koristi omotač (engl. *Wrapper*) kako bi pristupio servisima nego ima svoj prevoditelj za svaku platformu koji direktno pristupa resursima uređaja [8]. Takav pristup Flutteru omogućuje stabilnije i brže izvođenje aplikacija od ostalih programskih okvira.

Dart programski jezik koji je razvio Google koristi se u Flutteru. Dart je objektno orijentirani programski jezik optimiziran za razvoj mobilnih aplikacija, web aplikacija i aplikacija za Windows, Linux i MacOS [7]. Gradivni element Fluttera su widgeti, koji će biti objašnjen u poglavlju 3.1.

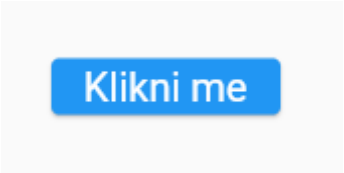
3.1. Widget

Svaki element koji se koristi u Flutter aplikaciji je widget. Widget predstavlja opis korisničkog sučelja. Widgeti nisu samo gradivni element poput gumba, teksta ili slike nego pomažu u definiraju dizajn i omogućuju korisničku interakciju [9]. Kako bi razdvojili elemente koristi se *Margin widget*, za centriranja koristimo *Center widget*. Osim toga, redovi i stupci koji definiraju okvir korisničkog sučelja su također widgeti. Zaključak je da su widgeti nacrti izgleda korisničkog sučelja.

Isječak programskog koda 1: Primjer jednostavnog widgeta

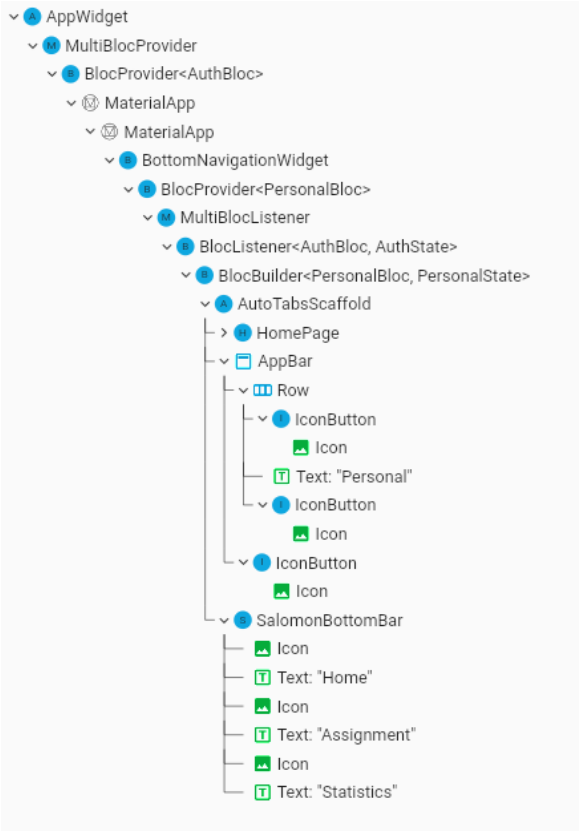
```
1.  Center(
2.    child:  ElevatedButton(
3.            style:  ElevatedButton.styleFrom( textStyle: const TextStyle(fontSize: 20)),
4.              onPressed: () {
5.                // poziva se klikom na gumb
6.              },
7.              child: const Text('Klikni me'),
8.            )
9.   );
```

Programski isječak koda 1 predstavlja primjenu widgeta. Glavni widget je Center koji vertikalno i horizontalno centrira widgete koji se nalaze unutar njega. Unutar Center widgeta nalazi se gumb kojemu su dodijeljeni widget za tekst i widget za povećanje veličine slova na 20. Slika 3 prikazuje kako taj widget izgleda.



Slika 3. Centrirani gumb

Widgeti se spremaju u strukturu stabla, ugniježđeni jedan u drugi formiraju aplikaciju prikazano slikom 4. Svaki novi widget koji dodajemo u aplikaciju dodaje se na kraj stabla. Svi widgeti koji se nalaze u istom čvoru poprimaju vrijednosti definirane u prethodnom čvoru, npr. ako je prethodni čvor imao centriranje po vertikalnoj i horizontalnoj osi onda će svi widgeti unutar tog čvora poprimiti isto to centriranje.



Slika 4. Prikaz widget stabla u aplikaciji

3.2. Flutter arhitektura

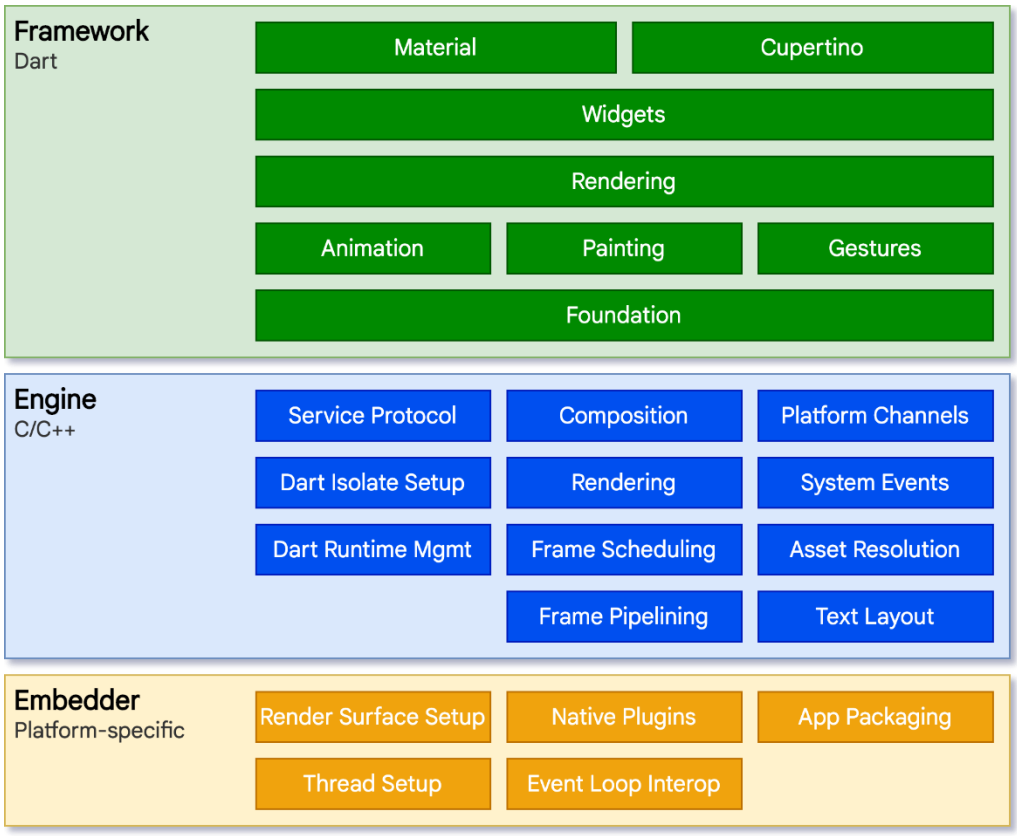
Flutter arhitektura prikazana je na slici 5, temeljena na tri sloja. Prvi sloj je upravljanje platformom (engl. *Embedder*), zatim sloj koji se nadovezuje na Embedder je sloj pokretača (engl. *Engine*) te zadnji sloj razvojni okvir (engl. *Dart Framework*) napisan u Dart programskom jeziku.

Embedder sloj je najniži sloj te ujedno predstavlja ulaznu točka za pojedinu platformu. Ovaj sloj ostvaruje direktnu komunikaciju s uređajem putem sučelja tako da aplikacija koja se izvodi ne gubi performanse. Programski jezik u kojem je Embedder napisan ovisi o platformi, a može biti Java i C++ za Android, Objective-C/Objective-C++ za iOS i MacOS, C++ za Windows i Linux i JavaScript za web.

Glavne zadaće sljedećeg Engine sloja su mrežni zahtjevi, ulazi i izlazi podataka. Osim toga, on je odgovoran i za iscrtavanje piksela koji se prikazuju na ekranu pomoću grafičke biblioteke Skia[3].

Framework sloj sadrži komponente za izradu korisničkog sučelja koje pružaju moderan, responzivan dizajn napisan u Dart programskom jeziku. Sadrži veliki broj unaprijed definiranih widgeta, temeljnih biblioteka i platformi te značajke poput kamere i Google karte koje je potrebno dodatno uključiti u projekt.

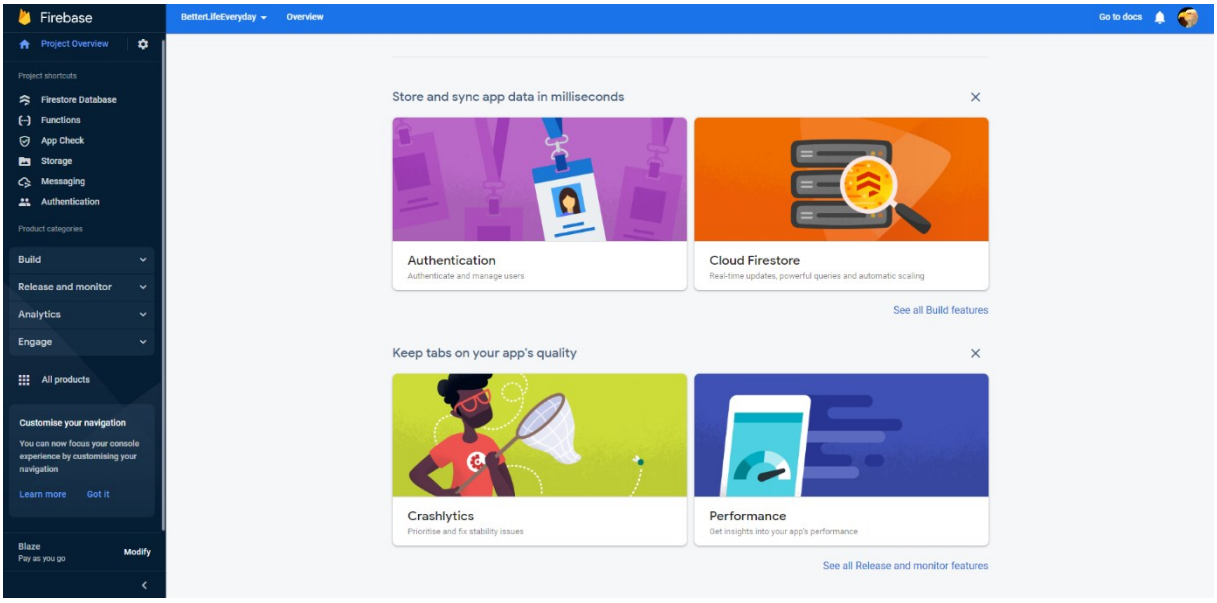
[3] Skia - biblioteka otvorenog koda za prikaz 2D grafike



Slika 5. Flutter arhitektura [6]

4. Firebase

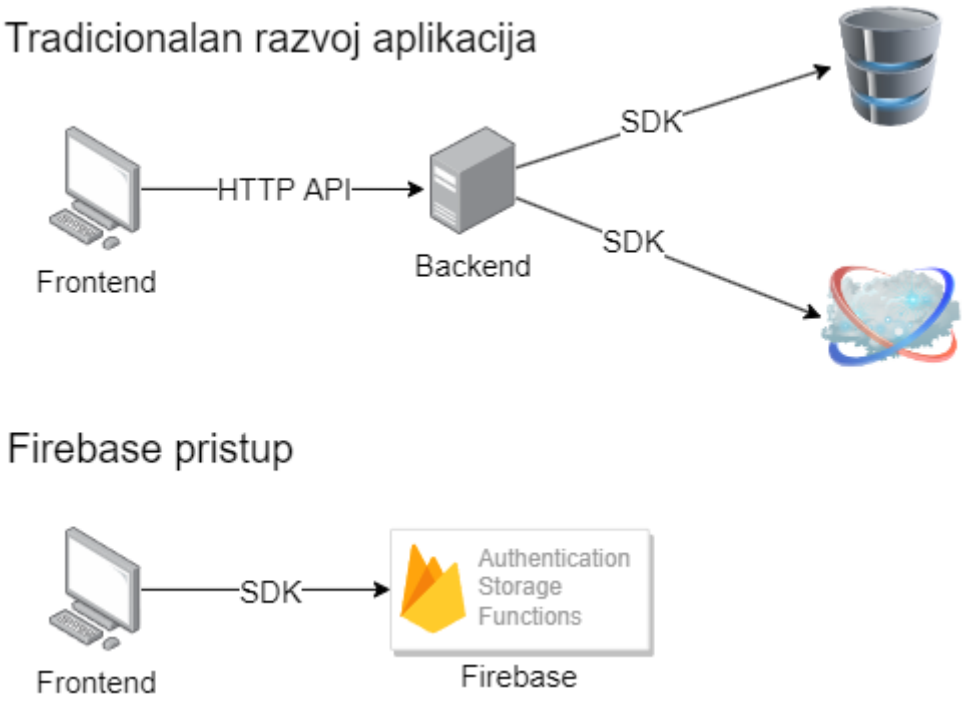
Firebase je Googleov Backend-as-a-Service (BaaS) platforma za razvoj dinamičnih aplikacija [3]. BaaS je usluga u oblaku koja sadrži niz servisa koji mogu raditi zajedno ili svaki posebno. Servisi su smješteni u oblaku, što znači da je Google odgovoran za održavanje servisa. Servise administriramo pomoću Firebase konzole prikazane na slici 6. Na primjer, Firebase konzola nam omogućava pregled i dodavanje novih servisa.



Slika 6. Firebase konzola

Klijentski alati za razvijanje softvera ili kraće SDK (engl. *Software Development Kit*) pružaju izravno komuniciranje s Firebase servisima, bez potrebe korištenja međuprograma između aplikacije i servisa. Takav pristup omogućava dohvaćanje podataka iz baze pisanjem upite na bazu u klijentskoj aplikaciji.

Tradicionalni razvoj aplikacija obično zahtjeva pisanje frontenda i backenda. Frontend poziva krajnju točku API-ja koji se nalazi na backendu te backend obrađuje podatke. Međutim pomoću Firebase zaobilazimo backend i podatke obrađujemo na klijentu. Razliku između tradicionalnog razvoja i razvoja pomoću Firebase možemo vidjeti na slici 7.



Slika 7. Komunikacija između tadicionalnog razvoja i korišenje firebasea

Trenutno Firebase nudi 24 servisa koji omogućuju razvijanje, nadogradnju i održavanje aplikacija.

U aplikaciji korišteni su sljedeći Firebase servisi:

- Authentication koji služi za autentifikaciju korisnika, pruža intuitivan i siguran postupak prijave pomoću Google, Twitter, Facebook ili GitHub računa.
- Cloud Firestore koji pruža bazu podataka gdje se spremaju podaci u obliku dokumenta koji su organizirani u kolekcije. Sinkronizira i sprema podatke u stvarnom vremenu. Pruža izvan mrežnu podršku.
- Cloud Functions omogućuje programiranje okidača koji izvršavaju određenu napisanu funkciju, kao što je izmjena podataka svaki dan u određeno vrijeme.
- Cloud Storage služi za pohranu bilo kakvih vrsta datoteka.

5. Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

Aplikacija je izrađena u Flutter razvojnom okviru te podržava Android i iOS platformu. Aplikacija je napravljena prema principu oblikovanja vođenog domenom (DDD, engl. *Domain-Driven Design*). DDD arhitektura je strukturirana da omogućuje vrlo lako održavanje aplikacije, bilo da se radi o dodavanju nove funkcionalnosti, zamjeni baze podataka ili ispravak već postojećeg koda.

5.1. Arhitektura aplikacije

Dizajn vođen domenom usmjeren je rješavanju složenost razvoja softvera. Kako se aplikacije nadograđuju tako složenost raste samim time i problemi pri razvoju. Dizajn usmjeren na domenu temelji se na domeni poslovanja. Rješava složene modele domene povezujući se s osnovnim poslovnim konceptima [11].

Arhitektura aplikacije (slika 8) jasno definira granice svakog sloja. Strelice obojenom istom bojom prikazuju tijek podataka. Tijek podataka može biti jednosmjeran ili dvosmjeran. Domenski sloj potpuno je neovisan o ostalim slojevima i sadrži samo čistu poslovnu logiku i podatke [12].

Prezentacijski sloj sadrži widgete i njihova stanja koji imaju jednu svrhu, a to je isporuka vrijednosti widgetima. Događaji se okidaju na temelju neke korisničke aktivnosti, primjerice, klikom na gumb. Glavna svrha događaja je pokretanje funkcija unutar aplikacijskog sloja. Ovisno da li funkcija prima parametre moguće je proslijediti neobrađene podatke iz polja u aplikacijski sloj na daljnju obradu. Prezentacijski sloj ne obrađuje nikakve podatke te služi samo za korisničku interakciju.

Aplikacijski sloj koristi BLoC za upravljanje stanjima unutar aplikacije i određuje kuda se podaci prosljeđuju. Ne izvodi nikakvu složenu poslovnu logiku, umjesto toga samo osigurava da je korisnički unos ispravan. Poziva sučelja koja se nalaze u domenskom sloju ili upravlja pretplatama na tokove podataka iz infrastrukture, indirektno korištenjem principa inverzije ovisnosti (engl. *dependency inversion*). Robert C. Martin izjavio je načelo inverzije ovisnosti navodi da moduli visoke razine ne bi trebali ovisiti o modulima niske razine.

Domenski sloj predstavlja središte aplikacije. Potpuno je samostalan i neovisan o drugim slojevima, dok svi ostali slojevi ovise o domenskom sloju. Ovaj sloj sadrži definirana

sučelja koji ostali slojevi koriste, a samostalnost pokazuje mogućnošću promijene baze podataka bez da se mijenja poslovna logika. Promjene u drugim slojevima ne utječu na domenski sloj, međutim promjene u domenskom sloju utječu na svaki drugi sloj.

Glavne značajke domenskog sloja:

- Provjerava valjanost podataka i održava podatke pomoću objekata s validiranom vrijednošću, primjerice, umjesto korištenja s*tringa* za naslov zadatka koristimo zasebnu klasu „*Title*“. Klasa „*Title*“ enkapsulirat će s*tring* vrijednost i pobrinuti se da nema više od 100 znakova i da nije prazna vrijednost.
- Transformira podatke
- Grupiranje i jednostavno identificiranje podataka
- Izvođenje složene poslovne logike - nije nužno uvijek slučaj u klijentskim Flutter aplikacijama, budući da se složena logika treba izvršavati na poslužitelju.

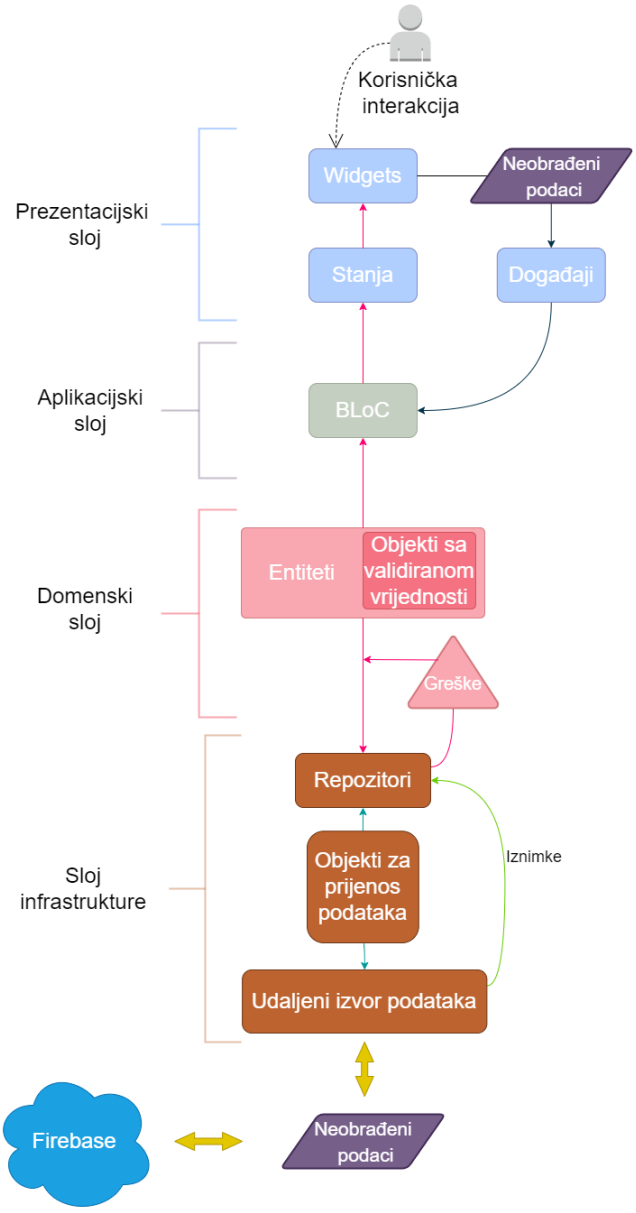
Upravljanje greškama je izazovan posao, greške se mogu dogoditi bilo gdje u kodu. Domenski sloj je središte cijele logike pa je time i središte za greške. Napravljeno je tako da vraća tip unije. Unija predstavlja dvije različite vrijednosti, nazvane lijevo i desno. Lijeva vrijednost sadrži greške dok desna sadrži vrijednosti ako nema grešaka. Takav princip nam osigurava da znamo u kojoj funkciji se dogodila greška.

Sloj infrastrukture omogućuje pozivanje API-ja, koristi Firebase servise, upravlja lokalnim bazama podataka i senzorima uređaja. Sloj infrastrukture sastoji se od dva dijela, a to su izvor podataka niske razine i repozitorija visoke razine. Dodatno, ovaj sloj sadrži objekte za prijenos podataka.

Objekti za prijenos podataka su klase čija je jedina svrha pretvaranje podataka između entiteta i vrijednost objekata iz sloja domene i običnih podataka koje dobivamo izvana. Firestore baza podataka može pohraniti samo osnovne tipove kao što je *int*, *string*, polje, itd. Zbog toga koristimo objekte s validiranom vrijednosti, jer ne želimo neprovjerene podatke u aplikaciji. Objekti za prijenos podataka mogu se serijalizirati i deserijalizirati.

Izvor podataka djeluje na najnižoj razini. Podaci s udaljenog izvora podataka su JSON (engl. *JavaScript Object Notation*) tipa koji se obrađuju i pretvaraju u objekte pomoću objekata za prijenos podataka. Također pretvaraju objekte u JSON tip.

Repozitorij obavlja važnu zadaću kao granica između domene i vanjskog svijeta. Poziva vanjske servise bilo da se radi o dohvaćanju podataka ili slanju podataka koristeći objekte za prijenos podataka.

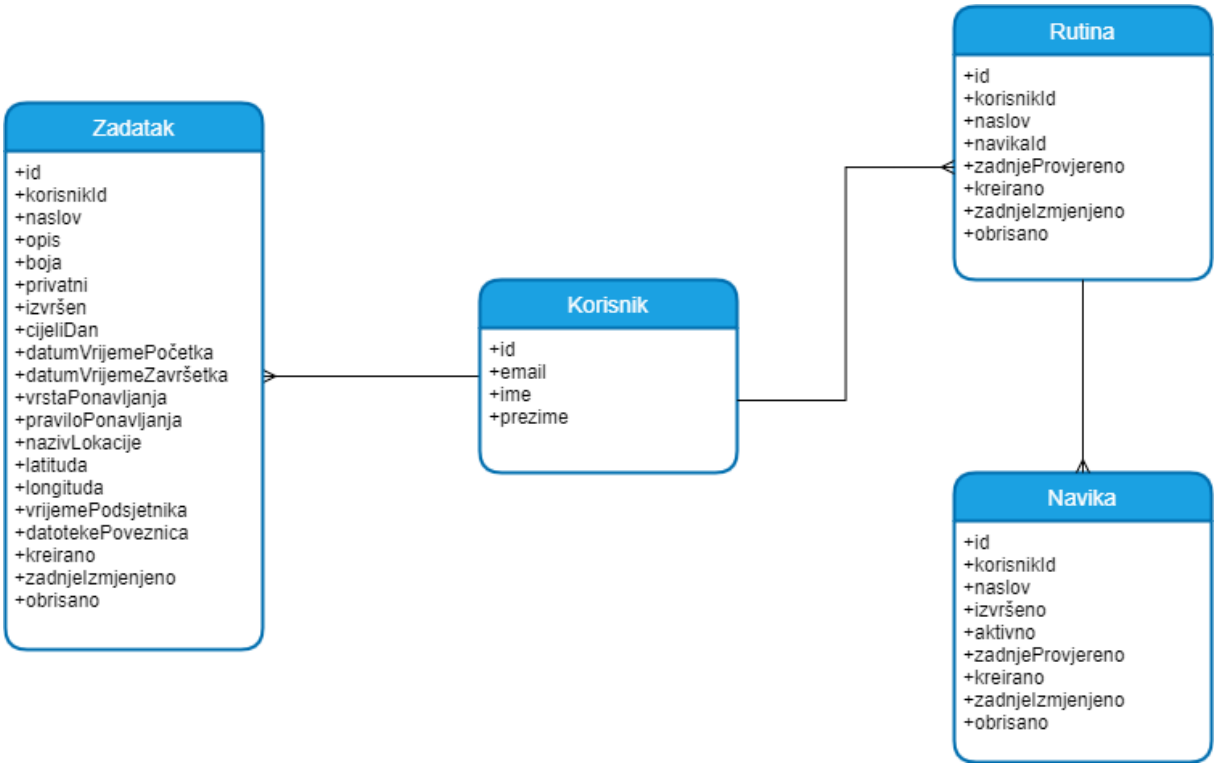


Slika 8. Arhitektura aplikacije.

5.2. Baza podataka

Za bazu podataka koristi se Firestore koji je dio Firebase servisa. Firestore je *NoSQL* baza podataka koja pohranjuje podatke u dokumente u obliku ključ-vrijednost. Ti dokumenti pohranjeni su u kolekciju koja je indeksirana i nad njom se mogu raditi upiti za dohvaćanje podataka.

Baza podataka korištena u aplikaciji sastoji se od četiri kolekcije korisnik, zadatak, rutina i navike – slika 9. Za pristup bazi koristi se Firestore *SDK*. *NoSQL* baza ne koristi relacije za povezivanje nego u dokumentima se navode polja identifikatora (*id*) po kojima se pretražuju dokumenti. Na primjer, da bi pretražili sve navike koje spadaju u rutinu, prvo moramo dohvatiti rutinu i njezin identifikator zatim dohvaćamo kolekciju s navikama i uspoređujemo svaki dokument da li u sebi sadrži taj identifikator rutine.

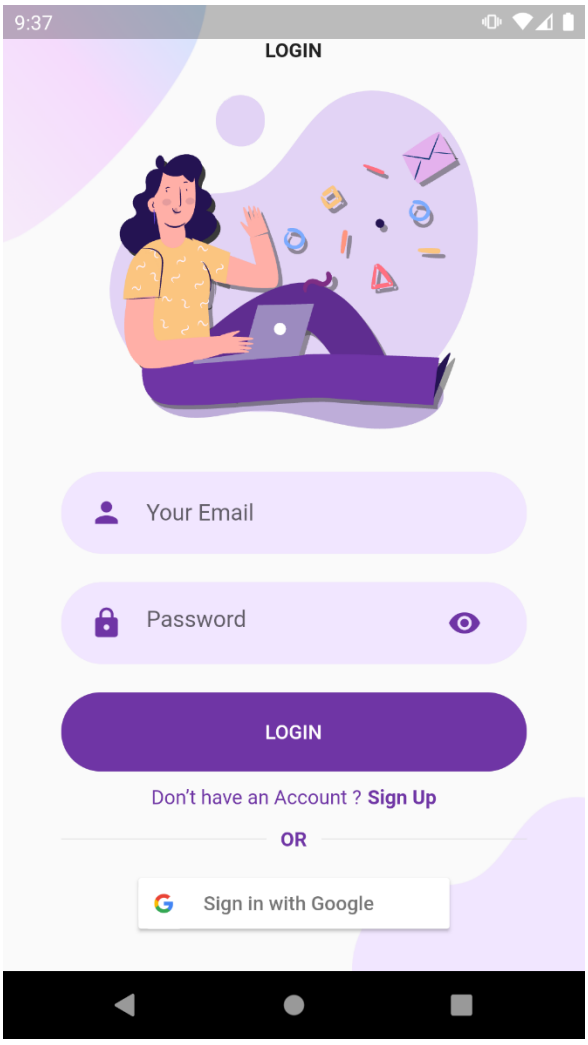


Slika 9. Shema baze podataka korištena u aplikaciji.

5.3. Prijava u aplikaciju

Aplikacija omogućuje korisnicima organizaciju osobnih i poslovnih zadataka i rutina. Nakon instalacije aplikacije prikazan je zaslon za prijavu. Slika 10 prikazuje zaslon za prijavu koja sadrži dva polja, email i lozinku (engl. *password*). Ispod polja nalazi se gumb za prijavu (engl. *login*). Moguće se prijaviti i putem brze prijave s Google računom. Uvjeti za prijavu putem forme su da email adresa mora biti ispravna i lozinka ne smije sadržavati manje od 6 znakova. Ukoliko se upiše kriva email adresa ili lozinka ispod neispravnog polja prikazuje se poruka koja daje korisniku do znanja da je unio krive podatke.

Nakon unesenih ispravnih podataka korisnik nastavlja s prijavom. Klikom na gumb *Login* okida se događaj u BLoCu i prosljeđuje podatke za prijavu. Kreira se objekt s validiranom vrijednosti koji je tipa *EmailAddress* za email i tipa *Password* za lozinku. Zatim poziva implementaciju sučelja koji poziva Firebase Authentication servis.



Slika 10. Prijava u aplikaciju

Isječak programskog koda 2 prikazuje prijavu pomoću Firebase Authentication servisa. Ima dva ulazna parametra *EmailAddress* i *Password*, izlazni parametar je predefinirana greška ili Unit, ne mogu biti dvije izlazne vrijednosti u isto vrijeme. Unit je tip koji predstavlja vrijednost koja ne pohranjuje nikakve informacije. Rezultat metode je Unit ukoliko se uspješno prijavio ili predefinirana greška ako je došlo do unosa krive kombinacije email i lozinke ili da se dogodila neka greška na strani poslužitelja. Nakon uspješne prijave prikazuje se ekran sa rutinama.

Isječak programskog koda 2: prijava sa Firebase Authentication servisa

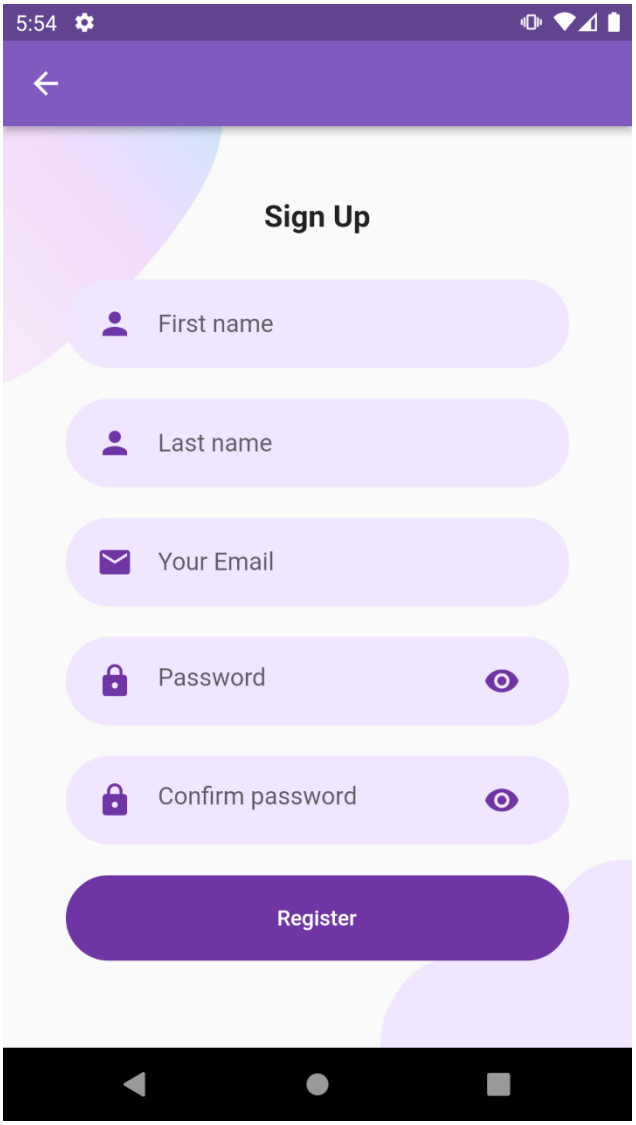
```
1.  Future<Either<AuthFailure, Unit>> signInWithEmailAndPassword(
2.        {required EmailAddress emailAddressObject, required Password passwordObject}) async
    {
3.      final emailAddressValue= emailAddressObject.getOrCrash();
4.      final passwordValue = passwordObject.getOrCrash();
5.      try {
6.        await _firebaseAuth.signInWithEmailAndPassword(
7.            email: emailAddressValue, password: passwordValue);
8.        return const Right(unit);
9.      } on FirebaseAuthException  catch (e) {
10.       if (e.code == 'wrong-password' || e.code == 'user-not-found') {
11.         return left(const AuthFailure.invalidEmailAndPasswordCombination());
12.       } else {
13.         return left(const AuthFailure.serverError());
14.       }
15.     }
16.   }
```

5.4. Registracija korisnika

Ukoliko se korisnik ne želi prijaviti putem Google servisa može se registrirati. Registrirani korisnik ima ista prava kao onaj koji se prijavio putem Google servisa. Dolazak na zaslon za registraciju moguće je isključivo preko zaslona za prijavu kikom na „*Sign Up*“. Klikom na „*Sign Up*“ otvara se poseban zaslon prikazan na slici 11, sadrži formu koju je potrebno popuniti za registraciju. Ikona sa lijevom strjelicom u gornjem lijevom kutu na slici 11 služi za vraćanje na zaslon prijave.

Za uspješnu registraciju potrebno je popuniti formu sa sljedećim poljima: ime (engl. *first name*), prezime (engl. l*ast name*), email, lozinka (engl. *password*) i potvrdna lozinka (engl. c*onfirm password*). Ukoliko korisnik ne ispuni nijedno od navedenih polja, ispuni samo neka polja ili neispravno ispuni polja, ispisat će se poruka crvenim slovima. Poruka korisniku daje do znanja koja polja nije ispunio ili ih je ispuno krivom vrijednošću. Osim što sva polja moraju biti ispunjena, kod email adrese potrebno je unijeti ispravnu email adresu. Lozinka mora sadržavati minimalno 6 znakova i mora se podudarati sa potvrdnom lozinkom. Kada su

zadovoljena pravila korisnik se registrira pomoću Firebase Authentication. Podaci o korisniku spremaju se u Firestore bazu podataka u kolekciju „*users*“. Kada se uspješno spreme podaci u bazu, korisnik se automatski prijavljuje u aplikaciju.



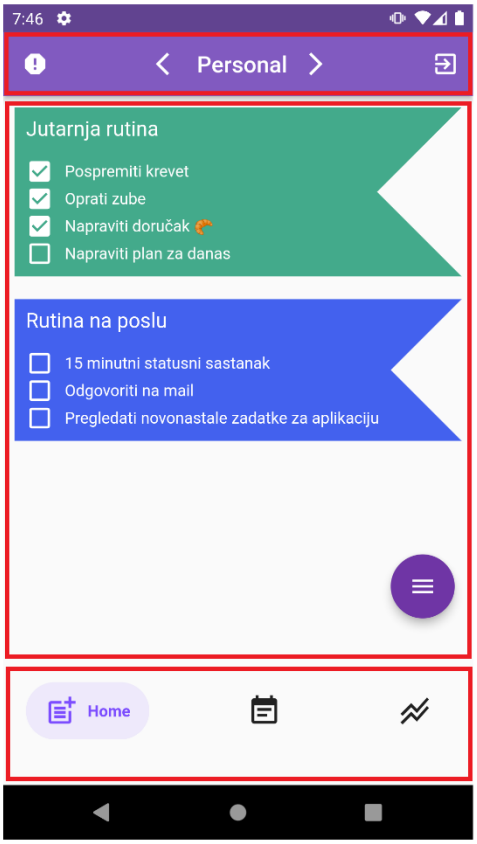
Slika 11. Registracija korisnika

5.5. Zaslon s rutinama

Nakon uspješne prijave prikazuje se početni zaslon, zaslon rutina – Slika 12. Zaslon se sastoji od tri dijela koje čine gornja traka na vrhu, sadržaj u sredini i navigacijskom trakom na dnu ekrana. Gornja traka i navigacijska traka uvijek je vidljiva kada se mijenjaju zasloni. Dok se u sadržaju prikazuje sadržaj ovisno o zaslonu na kojem se nalazi.

Gornja traka sastoji se od tri interaktivna elementa. Skroz lijevi element je ikona s uskličnikom, služi za slanje zahtijeva, pohvala i pritužba administratorima. Klikom na ikonu otvara se aplikacija za slanje email-a s unaprijed popunjenim primateljem i predmetom. Skroz desni element prikazan ikonom za izlaz odjavljuje korisnika iz aplikacije i prikazuje se zaslon za prijavu. U sredini trake nalazi se gumb za odabir između privatnih (engl. *personal*) ili poslovnih (engl. *office*) zadataka, o tome nešto više kod kalendarskog prikaza zadataka.

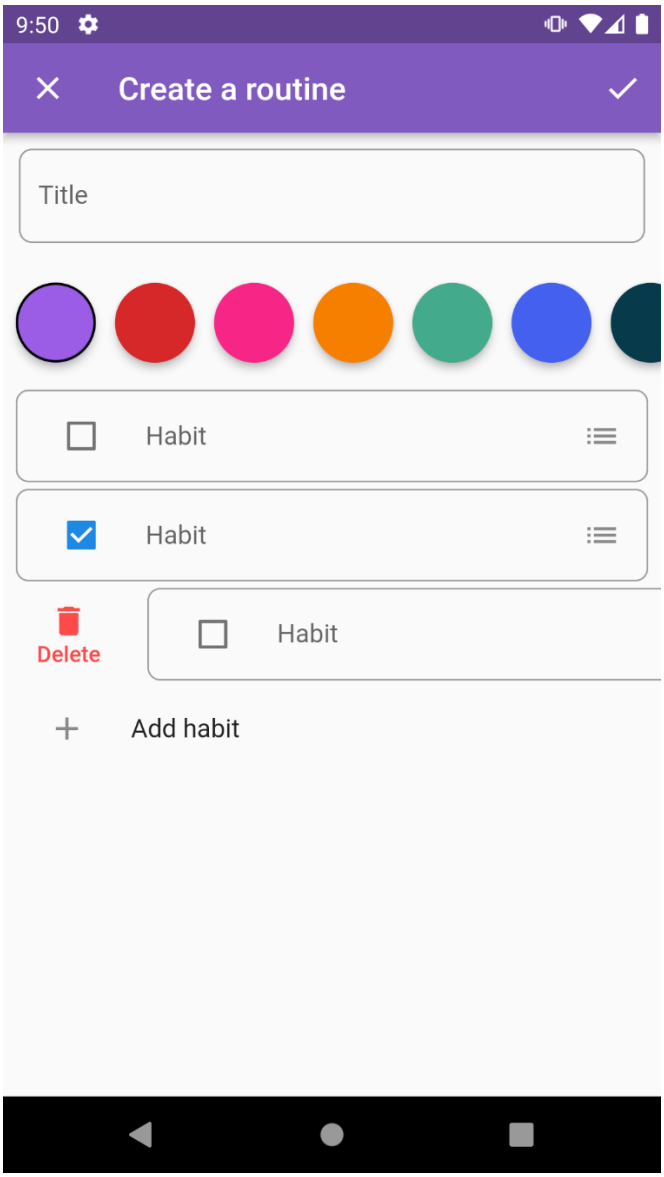
Navigacijska traka koja se nalazi na dnu zaslona ima tri gumba početna zaslon (engl. *home*), zaslon sa zadacima (engl. *assignment*) i statistika (engl. *statistics*). Klikom na gumb prikazuje se sadržaj.



Slika 12. Početni zazlon, zaslon rutina

Rutine (engl. *routine*) predstavljaju kategoriju u kojima se nalaze navike (engl. *habit*). Klikom na ikonu sa listom otvara se forma za dodavanje rutine. Slika 13 prikazuje formu za dodavanje rutine. U gornjoj traci nalaze se dva gumba: odustajanje od dodavanja rutine (✕) i dodavanje rutine (✓)

Forma se sastoji od naslova (engl. *title*), boje koja označava prioritet rutine i liste navika. Sva polja moraju biti popunjena u protivnom ispisat će se greška ispod polja koja nisu ispunjena ili su krivo popunjena. Polje naslov mora biti ispunjeno i ne smije sadržavati više od 60 znakova. Boju je moguće odabrati u protivnom je odabrana prva boja. Lista navika mora sadržavati barem jednu naviku i maksimalan broj navika je 15. Navika mora imati naziv. Naviku možemo označiti kao izvršenu, moguće je izbrisati iz liste povlačenjem u desnu stranu i sortirati.



Slika 13. Forma za kreiranje rutine i navika

Klikom na gumb (✓) poziva se sučelje u domenskom sloju za spremanje rutina. Sučelje je apstraktna klasa koja u sebi sadrži nazive metoda te njihovi ulazni i izlazni tipovi. Kako bi

znali koja je implementacija za koje sučelje koristi se inverzija ovisnosti. Isječak programskog koda 3 prikazuje sučelje koje se poziva prilikom dohvaćanja, kreiranja, ažuriranja i brisanja rutina.

Isječak programskog koda 3: Sučelje za rutine

```
1.  abstract class IRoutineRepository {
2.    Stream<Either<RoutineFailure, KtList<Routine>>> watchAll();
3.    Future<Either<RoutineFailure, Unit>> create(Routine routine);
4.    Future<Either<RoutineFailure, Unit>> update(Routine routine);
5.    Future<Either<RoutineFailure, Unit>> delete(Routine routine);
6.  }
```

Isječak programskog koda 4 prikazuje implementiranu metodu *create* iz sučelja za rutine prikazanog u isječku programskog koda 3. Implementacija metode nalazi se u sloju infrastrukture.

Namjena je spremanje rutina. Metoda prima validirani objekt tipa *Routine*, izlazni tip je greška ili Unit. Validirani objekt Routine kao takav ne može se spremiti, jer sadrži složene tipove i mora se pretvoriti u objekt s osnovnim tipovima podataka, jer Firestore poznaje samo osnovne tipove, npr. *string*, *number*, *boolean*, *array*, *timestamp*. Pretvaranje se obavlja u domenskom sloju pozivanjem funkcije „*fromDomain*“ nad objektom „*RoutineDto*“. Zatim pretvoreni objekt „*RoutineDto*“ koji sadrži osnovne tipove podataka pretvara se u JSON. Firestore se injektira u implementaciju sučelja pomoću injekcija ovisnosti (engl. *dependency injection*). Injekcija ovisnosti je prosljeđivanje objekata drugim objektima kojima su potrebni umjesto da ih oni sami kreiraju. Daje nam prednost u performansama, jer samo jednom kreiramo instancu Firestore objekta. Firestore sprema dokument s automatski generiranim identifikatorom u kolekciju „*routines*“. Ukoliko je uspješno spremljena rutina, funkcija vraća Unit u protivnom vraća grešku. Postoje dvije greške, korisnik nema prava na zapis u bazu i da se nešto neočekivano dogodilo.

Isječak programskog koda 4: Implemenatacije metod create

```
1.  Future<Either<RoutineFailure, Unit>> create(Routine routine) async {
2.      try {
3.        final userId = await getUserIdString();
4.
5.        final userRoutines = _firestore.collection('routines');
6.
7.        final routineDto = RoutineDto.fromDomain(routine, userId);
8.
9.        final data = routineDto.toJson();
10.
11.       await userRoutines.doc(routineDto.id).set(data);
12.       return right(unit);
13.     } on FirebaseException catch (e) {
14.       if (e.message!.contains('PERMISSION_DENIED')) {
15.         return left(const RoutineFailure.insufficientPermission());
16.       } else {
17.         return left(const RoutineFailure.unexpected());
18.       }
19.     }
20.   }
```

Nakon uspješnog spremanja korisniku će se prikazati zaslon s rutinama. Ukoliko se dogodi greška rutina se neće spremiti i greška će se prikazati korisniku. Klikom na rutinu otvara se ista forma kao i kod dodavanja samo sa popunjenim podacima. Možemo označiti koje smo navike obavili u toj rutini te možemo dodati još neku naviku, promijeniti naziv ili odabrati drugu boju odnosno prioritet. Navike u rutinama se svakim danom u ponoć ponovo kopiraju. Tako da se može pratiti povijest izvršenih ili ne izvršenih navika. Kopiranje izvršava funkcija napisana u Firebase Functions koja se okida u ponoć. Radi na način da se u navikama provjeri datum zadnje i uspoređuje sa trenutnim datumom. Ukoliko se datumi razlikuju kopira se navika i ona postaje aktivna.

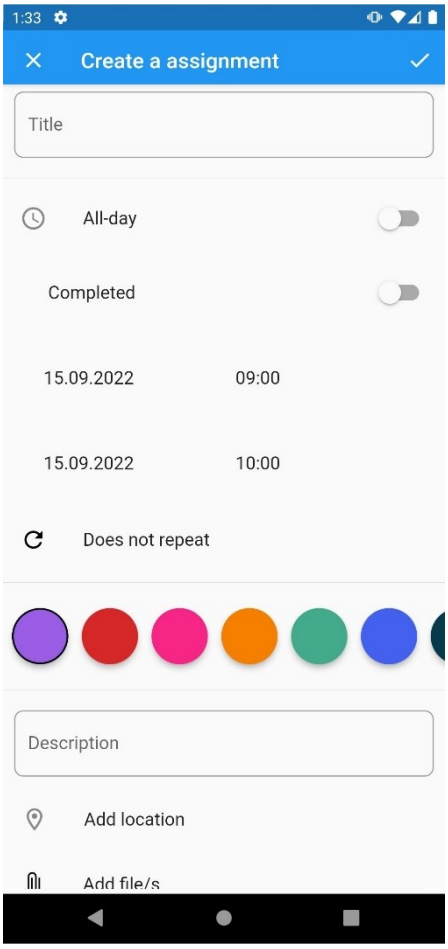
5.6. Zaslon sa zadacima

Klikom na zadatke u navigacijskoj traci otvara se zaslon sa zadacima. Zaslon omogućuje kalendarski pregled, uređivanje i brisanje zadataka. Osobni i poslovni zadaci mogu se mijenjati pomoću gumba na gornjoj traci. Zadaci u kalendaru se mogu prikazati na nekoliko načina, po danu, tjednu, radnom tjednu, mjesecu i godini ili po rasporedu što daje bolji uvid u zadatke. Moguće je odabrati prikaz zadataka za bilo koji mjesec i godinu.

Kada se prikaže stranica poziva se metoda koja radi stalnu poveznicu iz Firestore na aplikaciju i prati sve promijene koje se događaju. Ukoliko korisnik izgubi internetsku konekciju zadaci će i dalje moći pregledavati zbog izvanmrežnog načina rada Firestorea

Dodavanje zadataka je jednostavno upotrebom klika na željeni dan u kalendarskom prikazu. Otvara se prikaz po danu, odabere se vrijeme početak zadatka i otvara se forma za

dodavanje zadataka – slika 14. Kod dodavanja zadataka potrebno je popuniti samo naslov. Vrijeme kraja zadatka mora biti veće od vremena početka zadatka. Opcija cijeli dan (engl. *all-day*) označava zadatak za cijeli dan i onemogućuje odabir datuma i vremena. Završeno (engl. *completed*) određuje je li zadatak izvršen.

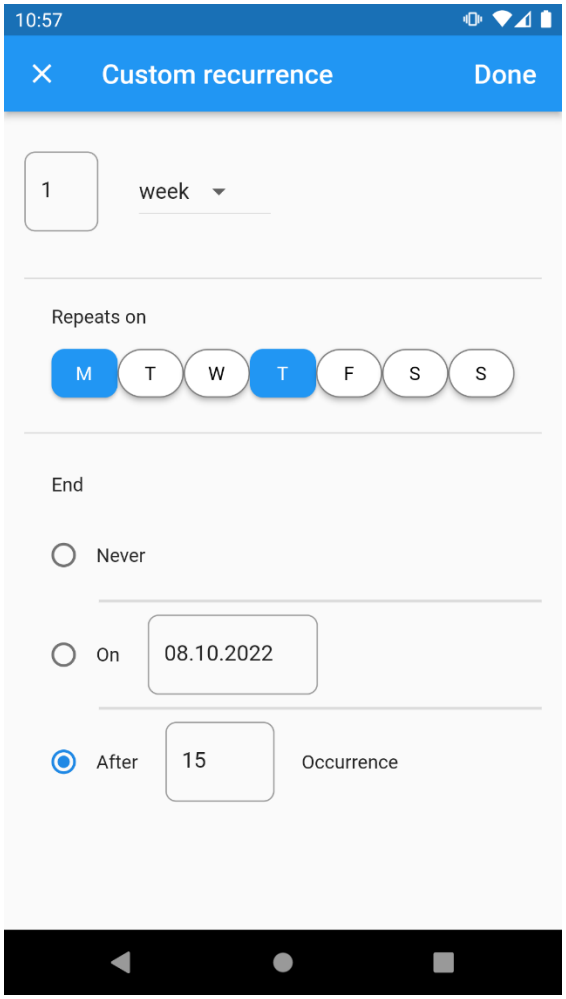


Slika 14. Forma za dodavanje zadatka

Ispod dodavanja vremena nalazi se opcija za ponavljanje zadataka. Klikom na opciju otvara se dijalog i prikazuju se opcije za ponavljanje. Slika 15 prikazuje opcije: bez ponavljanja (engl. *does not repeat*), svaki dan (engl. *every day*), svaki tjedan (engl. *every week*), svaki mjesec (engl. *every month*), svaku godinu (engl. *every year*) i prilagođeno (engl. *custom*). Odabir opcije prilagođeno ponavljanje, otvara novi zaslon koji omogućava odabir dana, jednog ili više te periodičnost ponavljanja zadatka. Periodičnost se može definirati kao dan, tjedan, mjesec ili godina, a završetak periodičnost se realizira odabirom opcije nikad, na koji dan završava ponavljanje ili nakon koliko ponavljanja.

Boje označavaju prioritet što daje fleksibilnost svakom korisniku da si sam određuje prioritete. Korisnik može odabrati 7 različitih boja. Polje opis (engl. *description*) zadatka poželjan je, ali ne mora biti nužan.

Zadatak može sadržavati lokaciju, klikom na dodaj lokaciju (engl. *add location*) otvara se dijalog i prikazuju se Google karte i pita korisnika da li aplikacije može koristiti lokaciju. Korisnik može, ali ne mora prihvatiti uvjet, ova značajka služi da se postavi lokacija korisnikovog mobitela. Lokacija se može odrediti pomicanjem karte ili upisivanjem u polje gdje korisniku ponudi moguće lokacije na temelju unesenog naziva. Osim lokacije korisniku je omogućeno dodavanje datoteka na zadatak.



Slika 15. Prilagođeno ponavljanje zadatka

6. Zaključak

Izrada mobilnih aplikacija ovisi o platformi na kojoj se izvode. Flutter uz programski jezik Dart omogućava da se jedna aplikacija izvodi na više platformi. Flutter se redovito nadograđuje kako bi bio u skladu s novim mogućnostima mobilnih uređaja, web-a i računala. Otvorenost koda i velika zajednica pospješuje nadograđivanje Fluttera. Flutter sadrži gradivne elemente widgete za izradu korisničkog sučelja aplikacije. Kako bi proširili gradivne elemente uključuju se dodatni paketi u aplikaciju. Uz Flutter čija je svrha izgradnja korisničkog sučelja koristi se Firebase. Firebase sadrži servise smještene u oblaku koji su neovisni, ali imaju mogućnost zajedničkog rada. Primjeri nekih funkcionalnosti koje pružaju Firebase servisi su jednostavna prijava putem Google računa, sinkronizacija baze podataka i analitika.

Svrha izrađene mobilne aplikacije u ovom radu je organiziranje osobnih i poslovnih zadataka i rutina. Rutine u aplikaciji predstavljaju jednostavne zadatke koji se obavljaju svaki dan. Osim toga kalendarski prikaz zadataka omogućuje bolje organiziranje i efikasnije izvođenje zadataka uz grafički prikaz uspješnosti. Sama aplikacija napravljena je pomoću Fluttera i Firebasea, strukturirana je prema principima oblikovanja vođenog domenom koja omogućuje laku nadogradnju i održavanje.

Kroz zadatak završnog rada proširio sam znanje u području programiranja mobilnih aplikacija i izradi arhitekture na temelju principa oblikovanja vođenog domenom.

6. Popis literature

[1] Statcounter. Pristupljeno 1.Rujan. 20222 [Online]. Dostupno na: https://gs.statcounter.com/os-market-share/mobile/worldwide
[2] Androidauthority, Pristupljeno 1. rujna 2022 [Online]. Dostupno na: https://www.androidauthority.com/what-is-android-328076/
[3] Tutorialwing, Pristupljeno 3. rujna 2022 [Online]. Dostupno na: https://tutorialwing.com/an-overview-of-android-architecture/
[4] Educationecosystem. Pristupljeno 3. rujna 2022 [Online]. Dostupno na: https://educationecosystem.com/guides/programming/ios/history
[5] Apple. Pristupljeno 4. rujna 2022 [Online]. Dostupno na: https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/Mach/Mach.html
[6] Flutter. Pristupljeno: 2. kolovoza 2022. [Online]. Dostupno na: https://flutter.dev/
[7] AWH, Hackernoon (2021). Pristupljeno 5. rujna 2022 [Online]. Dostupno na: https://hackernoon.com/what-is-flutter-and-how-does-it-work-lh49370l
[8] Dart. Pristupljeno: 2. kolovoza 2022. [Online]. Dostupno na: https://dart.dev/
[9] Firebase. Pristupljeno: 3. kolovoza 2022. [Online]. Dostupno na: https://firebase.google.com/
[10] Google Cloud . Pristupljeno: 3. kolovoza 2022. [Online]. Dostupno na: https://cloud.google.com/firestore/docs
[11] Sara Miteva, Medium(2022). Pristupljeno 4. rujna 2022 [Online]. Dostupno na: https://medium.com/microtica/the-concept-of-domain-driven-design-explained-3184c0fd7c3f
[12] Matt Rešetár, ResoCoder (2020). Pristupljeno 5. rujna 2022 [Online]. Dostupno na: https://resocoder.com/2020/03/09/flutter-firebase-ddd-course-1-domain-driven-design-principles/
[13] Android. Pristupljeno 1.rujna 2022 [Online]. Dostupno na: https://developer.android.com/guide/platform
[14] Lucideus, Medium (2019). Pristupljeno 1. rujna 2022 [Online]. Dostupno na: https://medium.com/@lucideus/understanding-the-structure-of-an-ios-application-a3144f1140d4

7. Popis slika

Slika 1. Android arhitektura [13] 4
Slika 2. iOS arhitektura [14] 6
Slika 3. Centrirani gumb 8
Slika 4. Prikaz widget stabla u aplikaciji 8
Slika 5. Flutter arhitektura [6] 10
Slika 6. Firebase konzola 11
Slika 7. Komunikacija između tadicionalnog razvoja i korišenje firebasea 12
Slika 8. Arhitektura aplikacije. 15
Slika 9. Shema baze podataka korištena u aplikaciji. 16
Slika 10. Prijava u aplikaciju 17
Slika 11. Registracija korisnika 19
Slika 12. Početni zazlon, zaslon rutina 20
Slika 13. Forma za kreiranje rutine i navika 21
Slika 14. Forma za dodavanje zadatka 24
Slika 15. Prilagođeno ponavljanje zadatka 25

Veleučilište u Virovitici

OBRAZAC 5

IZJAVA O AUTORSTVU

Ja, Ivan Horvat

izjavljujem da sam autor/ica završnog/diplomskog rada pod nazivom

Mobilna aplikacija za organizaciju osobnih i poslovnih zadataka i rutina

Svojim vlastoručnim potpisom jamčim sljedeće:

- da je predani završni/diplomski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija,
- da su radovi i mišljenja drugih autora/ica, koje sam u svom radu koristio/la, jasno navedeni i označeni u tekstu te u popisu literature,
- da sam u radu poštivao/la pravila znanstvenog i akademskog rada.

Potpis studenta/ice

Ivan Horvat

Veleučilište u Virovitici

OBRAZAC 6

ODOBRENJE ZA POHRANU I OBJAVU ZAVRŠNOG/DIPLOMSKOG RADA

Ja Ivan Horvat

dajem odobrenje za objavljivanje mog autorskog završnog/diplomskog rada u javno dostupnom digitalnom repozitoriju Veleučilišta u Virovitici te u javnoj internetskoj bazi završnih radova Nacionalne i sveučilišne knjižnice bez vremenskog ograničenja i novčane nadoknade, a u skladu s odredbama 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, 131/17).

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

(a) široj javnosti
b) studentima i djelatnicima ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

Potpis studenta/ice

Ivan Horvat

U Virovitici, 9.9.2022

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem završnom/diplomskom radu, podnosi se pisani obrazloženi zahtjev.*