Android aplikacija za praćenje zdravstvenog stanja

Zovko, Matea

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:465760

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-10**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Preddiplomski sveučilišni studij računarstva

MOBILNA APLIKACIJA ZA PRAĆENJE ZDRAVSTVENOG STANJA OSOBE

Završni rad

Matea Zovko

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 12.09.2023.

Ime i prezime studenta:	Matea Zovko
Studij:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. studenta, godina upisa:	R 4445, 22.07.2019.
Turnitin podudaranje [%]:	14

Ovom izjavom izjavljujem da je rad pod nazivom**: Android aplikacija za praćenje zdravstvenog stanja**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Rudec

i sumentora izv. prof. dr. sc. Tomislav Keser

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:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

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

Osijek, 28.08.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Matea Zovko
Studij, smjer:	Sveučilišni prijediplomski studij Računarstvo
Mat. br. Pristupnika, godina	R 4445, 22.07.2019.
OIB Pristupnika:	38307538021
Mentor:	doc. dr. sc. Tomislav Rudec
Sumentor:	izv. prof. dr. sc. Tomislav Keser
Sumentor iz tvrtke:	
Naslov završnog rada:	Android aplikacija za praćenje zdravstvenog stanja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Student će izraditi android aplikaciju za praćenje zdravstvenog stanja Tema zauzeta: Matea Zovko
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	28.08.2023.
Datum potvrde ocjene od strane Odbora:	08.09.2023.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

SADRŽAJ

1. UVOD

U svakodnevnom životu, većini ljudi barem se jednom dogodilo da osjeti neobjašnjiv umor, da se generalno loše osjeća, a ne prepoznaje odmah iz kojeg razloga. Razlog tome, uz naravno teže bolesti s kojima se osoba mora posavjetovati s liječnikom može biti nedovoljan unos tekućine, većinom sjedilački način života bez fizičke aktivnosti, visok ili nizak krvni tlak koji se ne kontrolira redovito ili zaboravljanje uzimanja redovite terapije prepisane od liječnika. U vrlo dinamičnom načinu života s kojim se većina svakodnevno bori, teško je svakodnevno pratiti koliko smo, na primjer, popili vode ili hodali. Tema ovog završnog rada je izrada aplikacije za praćenje zdravstvenog stanja osobe. Cilj ove mobilne aplikacije je olakšati praćenje navedenih parametara osobama svih uzrasta i time općenito popraviti kvalitetu života.

U drugom poglavlju bit će opisane slične aplikacije koje postoje na tržištu i bile su ključne za razvoj ove, koja sadrži, po subjektivnom mišljenju, najvažnije karakteristike obje najprodavanije aplikacije po pitanju cjelokupnog zdravlja, ali i nudi dodatne koje još nisu uključene u već postojeće radi boljeg korisničkog iskustva. Detaljan opis korištenih alata i tehnologija nalazi se u trećem poglavlju rada, dok četvrto poglavlje sadrži korake izrade dizajna aplikacije u Figmi, te same aplikacije u okruženju Android Studio koristeći programski jezik Kotlin. Peto poglavlje daje prikaz samog toka aplikacije i svih mogućih ekrana i mogućnosti unutar aplikacije.

1.1Zadatak završnog rada

Zadatak ovog završnoga rada jest objasniti postupak izrade aplikacije u Android Studiu, te izraditi aplikaciju za olakšano praćenje zdravstvenog stanja osobe. Aplikacija omogućava stvaranje korisničkog računa, unošenje specifičnih podataka poput dnevnog unosa vode, količinu sna, broj koraka i sl. Aplikacija također omogućava korisniku pregled prethodnih unosa te kroz jednostavno sučelje daje korisniku do znanja ako neki od unosa nije ispunio preporučenu dnevnu količinu. Korisnik također unosi osobne podatke o sebi poput visine, težine, starosti i sl. Te podatke korisnik onda može naknadno mijenjati.

2. SLIČNE APLIKACIJE NA TRŽIŠTU

Danas na tržištu možemo pronaći mnoge aplikacije koje olakšavaju praćenje i unaprjeđenje zdravstvenog stanja općenito. Neke od popularnijih su one velikih tvrtki poput Samsunga i njihove aplikacije Samsung Health, te Huawei-ev Health. U ovom poglavlju bit će prikazane navedene aplikacije sa svojim prednostima i nedostatcima.

2.1 Samsung Health App

Samsung Health je besplatna aplikacija koju je razvio Samsung 2012. godine i služi za praćenje različitih aspekata svakodnevnog života koji doprinose dobrobiti poput fizičke aktivnosti, prehrane i sna. Ova aplikacija nije izrazito složena i zahtjevna, ali nudi mnogo alata koji mogu pomoći u upravljanju i praćenju kvalitete života općenito. Također, nudi bogate resurse, fitness trenere, vježbe te zajednicu istomišljenika [1].

Glavne značajke aplikacije:

- Korištenje ciljeva koje je aplikacija predložila za poboljšanje rezultata
- Pedometar
- Tjedni sažeci glavnih značajki
- Praćenje aktivnosti
- Praćenje prehrane (apsorbirane kalorije i hranjive tvari)
- Praćenje težine
- Praćenje spavanja

Slika 2.1 Korisničko sučelje aplikacije Samsung Health [2]

2.2 HUAWEI Health

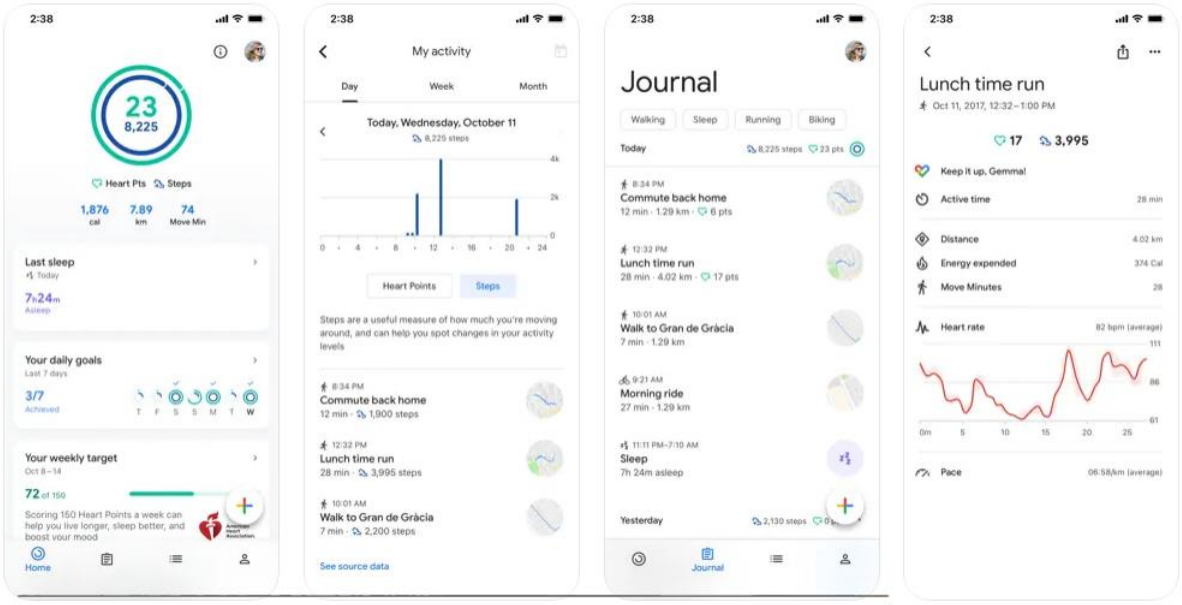
Huawei Health je službena Huawei aplikacija za praćenje zdravlja i tjelesne aktivnosti. Aplikacija omogućuje vođenje detaljnih podataka o navikama spavanja, povijesti tjelesne težine, dnevnim sagorjelim kalorijama ili pulsu [3].

Kartica vježbe omogućava unos dnevne fizičke aktivnosti. Kartice spavanja i otkucaja srca omogućuju kontrolu trenutnog zdravstvenog stanja. Također, prikazuje koliko je sati laganog i dubokog sna svake noći, te pokazuje podatke o otkucajima srca.

Slika 2.2 Korisničko sučelje aplikacije Huawei Health [4]

2.3 Google Fit

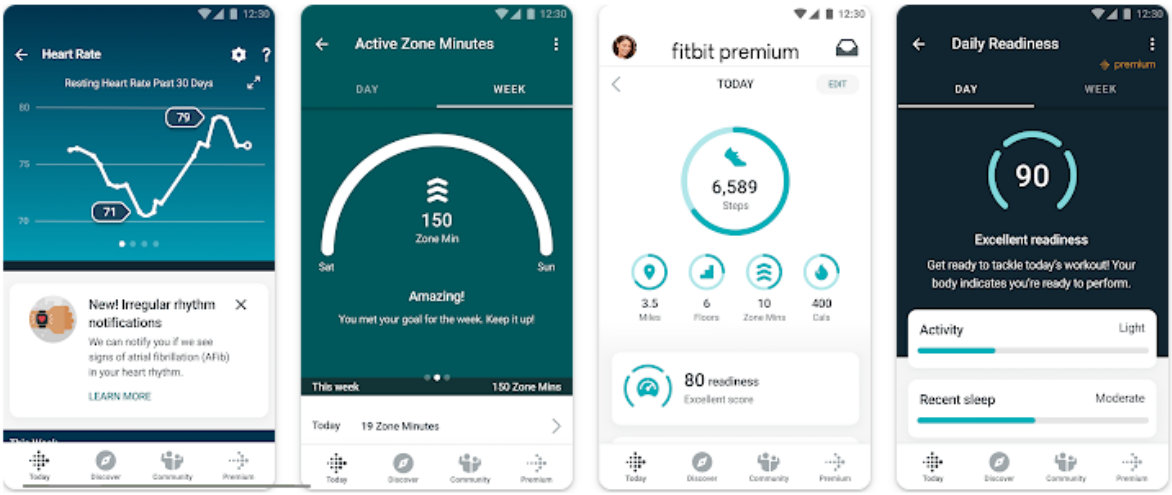
Ova aplikacija je razvijena 2014. godine od strane Google-a. Ova aplikacija je razvijena za mobilne i Wear OS uređaja. Glava odlika joj je to što automatski čita senzore s uređaja i tako pruža korisniku razne podatke o zdravlju i trenutnoj aktivnosti. Ova aplikacija također omogućava integraciju i povezivanje sa drugim sličnim aplikacijama poput Strava i Aqualert. Glavne prednosti aplikacije su dodatna integracija sa drugim aplikacijama, detaljan pregled podataka i efikasno praćenje koraka i otkucaja. Nedostatak je nekada prekomplicirano korisničko sučelje i nedostatak praćenja stvarnih vježbi.



Slika 2.3 Korisničko sučelje aplikacije Google Fit [5]

2.4 Fitbit

Fitbit je još jedna aplikacija koja je dio Google-a. Omogućava praćenje podataka poput koraka, aktivnosti te nudi korisniku savjete u vidu motivacije i dodatnih opcija po pitanju potrebnih aktivnosti. Omogućava i praćenje sna. Sve ovo je omogućeno uz pomoć posebnog uređaja koji korisnik nosi na sebi i koji se povezuje sa aplikacijom. To je ujedno i jedan od nedostataka jer bez uređaja aplikacija je beskorisna. Točnost je naravno uvijek upitna sa ovim uređajima također. Prednosti su jako širok izbor funkcionalnosti te detaljan pregled svih podataka i mogućnost pregleda dodatnih aktivnosti koje pomažu korisniku da napreduje.



Slika 2.4 Korisničko sučelje aplikacije Fitbit [6]

3. KORIŠTENI ALATI I TEHNOLOGIJE

3.1 Android

Android je operacijski sustav otvorenog programskog koda za mobilne uređaje, kao što su pametni telefoni, pametni satovi i tablet računala, američke tvrtke Google Inc. Android OS temeljen je na jezgri Linux i drugom softveru otvorenog kôda. Uz mobilne uređaje android se još koristi i u pametnim televizorima, te u pametnim satovima. Android platforma prilagođena je za uporabu na uređajima s većim zaslonima, poput pametnih telefona koji rabe 2D grafičku knjižnicu ili 3D grafičku knjižnicu temeljenu na OpenGL ES 2.0 specifikacijama. Za pohranu podataka upotrebljava se SQLite relacijska baza podataka, napisana u programskom jeziku C. Android je isprva razvila tvrtka Android Inc. Prva namjera tvrtke je bila razviti operacijski sustav za digitalne kamere, no to je tržište bilo premalo pa su promijenili cilj i okrenuli se operacijskom sustavu za mobilne uređaje kao izravnu konkurenciju Symbian-u i Windows mobileu. 2005. godine Google kupuje tvrtku Android Inc., a 2007. godine osnovan je Open Handset Alliance (OHA) s ciljem stvaranja javnog standarda za mobilne uređaje te je odmah iste godine bilo prvo javno predstavljanje Androida kao operacijskog sustava baziranog na Linux jezgri [7].

Prvi mobilni uređaj sa android sustavom je bio HTC Dream prikazan na slici 3.1, dok je na slici 3.2 prikazana najnovija inačica androida na pametnom telefonu. Dosadašnje inačice Androida kroz povijest prikazane su na slici 3.3.

Slika 3.1 HTC Dream(Android 1.1) [7]

Slika 3.2 Prikaz sučelja Android 13 [7]

Name	Internal codename[9]	Version number(s)	API level	Initial stable release date
Android 1.0	N/A	1.0	1	September 23, 2008
Android 1.1	Petit Four	1.1	2	February 9, 2009
Android Cupcake	Cupcake	1.5	3	April 27, 2009
Android Donut	Donut	1.6	4	September 15, 2009
Android Eclair	Eclair	2.0	5	October 27, 2009
		2.0.1	6	December 3, 2009
		2.1	7	January 11, 2010[16]
Android Froyo	Froyo	2.2 – 2.2.3	8	May 20, 2010
Android Gingerbread	Gingerbread	2.3 – 2.3.2	9	December 6, 2010
		2.3.3 – 2.3.7	10	February 9, 2011
Android Honeycomb	Honeycomb	3.0	11	February 22, 2011
		3.1	12	May 10, 2011
		3.2 – 3.2.6	13	July 15, 2011
Android Ice Cream Sandwich	Ice Cream Sandwich	4.0 – 4.0.2	14	October 18, 2011
		4.0.3 – 4.0.4	15	December 16, 2011
Android Jelly Bean	Jelly Bean	4.1 – 4.1.2	16	July 9, 2012
		4.2 – 4.2.2	17	November 13, 2012
		4.3 – 4.3.1	18	July 24, 2013
Android KitKat	Key Lime Pie	4.4 – 4.4.4	19	October 31, 2013
		4.4W – 4.4W.2	20	June 25, 2014
Android Lollipop	Lemon Meringue Pie	5.0 – 5.0.2	21	November 4, 2014[17]
		5.1 – 5.1.1	22	March 2, 2015[18]
Android Marshmallow	Macadamia Nut Cookie	6.0 – 6.0.1	23	October 2, 2015[19]
Android Nougat	New York Cheesecake	7.0	24	August 22, 2016
		7.1 – 7.1.2	25	October 4, 2016
Android Oreo	Oatmeal Cookie	8.0	26	August 21, 2017
		8.1	27	December 5, 2017
Android Pie	Pistachio Ice Cream[20]	9	28	August 6, 2018
Android 10	Quince Tart[21]	10	29	September 3, 2019
Android 11	Red Velvet Cake[21]	11	30	September 8, 2020
Android 12	Snow Cone	12	31	October 4, 2021
Android 12L	Snow Cone v2	**12.1**[a][b]	32	March 7, 2022
Android 13	Tiramisu[23]	13[c]	33	Q3 2022

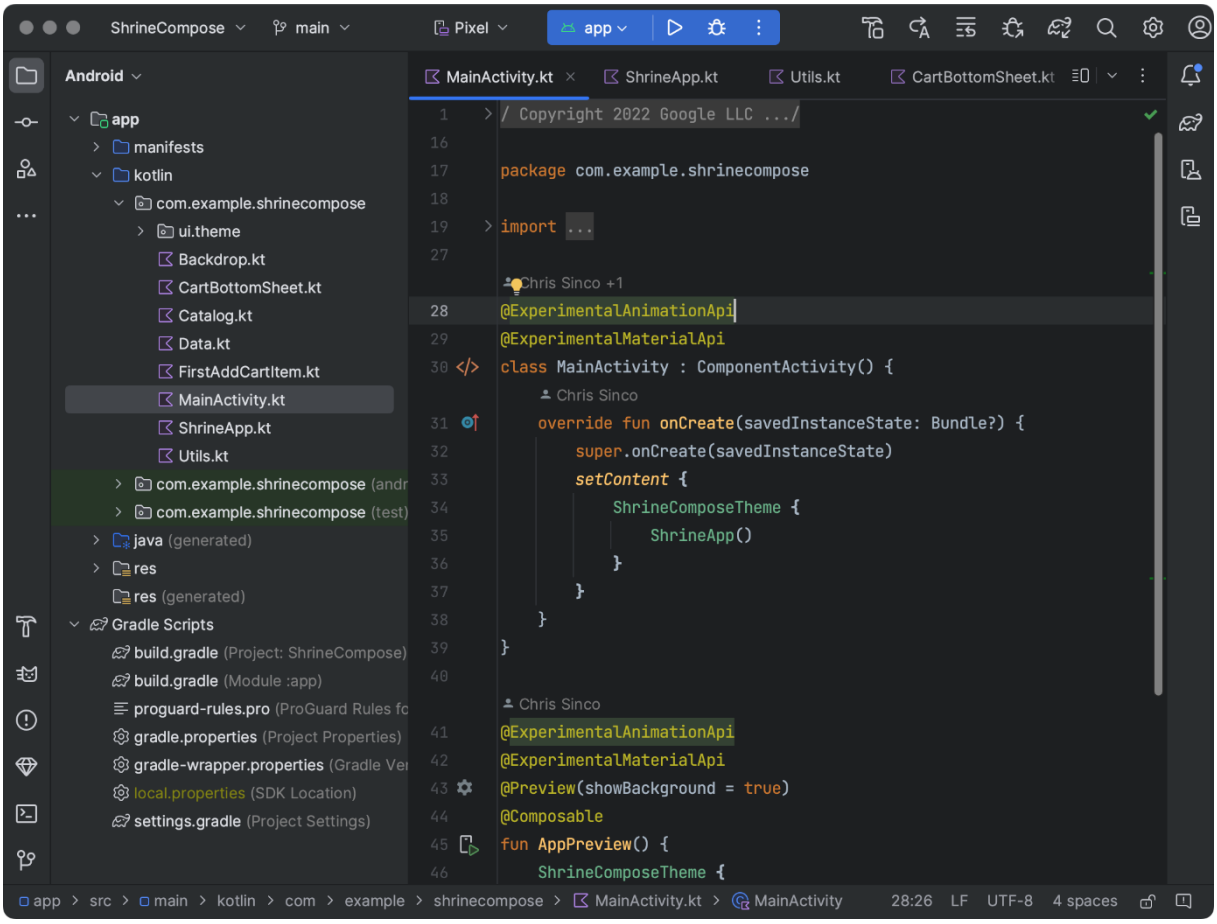
Slika 3.3 Inačice Androida kroz povijest [7]

3.2 Android Studio

Android Studio službeno je integrirano razvojno okruženje za razvoj Android aplikacija, temeljeno na IntelliJ IDEA. Osim IntelliJ-ovog moćnog uređivača koda i alata za programere, Android Studio nudi još značajki koje poboljšavaju produktivnost pri izradi Android aplikacija, kao što su [8]:

- Fleksibilan sustav gradnje zasnovan na Gradleu
- Brz i bogat emulator
- Jedinstveno okruženje u kojem možete razvijati za sve Android uređaje
- Opsežni alati i okviri za testiranje
- Podrška za Kotlin, Javu, C++ i NDK

Android studio dostupan je za računala s MacOS, Linux te Windows operacijskim sustavima



Slika 3.3 Korisničko sučelje Android Studia [9]

3.3 Programski jezik Kotlin

Kotlin je programski jezik otvorenog koda koji podržava objektno orijentirano, ali i funkcijsko programiranje. Ima sintaksu sličnu drugim često korištenim programskim jezicima poput C# ili Jave. Kotlin je službeno podržan od strane Google-a za razvoj Android aplikacija tako što su sva dokumentacija i alati prilagođeni za Kotlin. Upravo zbog toga se najčešće koristi pri izradi Android aplikacija. Preko 50% Android developera koristi Kotlin kao svoj primarni jezik, dok je njih svega 30% za svoj primarni jezik odabralo Javu [10].

3.4 Jetpack Compose

Jetpack Compose predstavlja moderni način izrade korisničkog sučelja i upravljanjem svih podataka koji se prikazuju. To je posve novi skup alata razvijen direktno od strane *Google-a* i konstantno se poboljšava te se već uveliko koristi za izradu profesionalih produkcijskih aplikacija. Najnoviji primjer aplikacije koja koristi Jetpack Compose je *Threads* aplikacija [11].

Jetpack Compose se razlikuje znatno od prijašnje metode izrade sučelja koja je bila zasnovana na XML datotekama koje pomoću osnovnih gradivnih elmenata opisuju izgled sučelja. U većim aplikacijama ovo je često uzrokovalo dosta nejasnoća po pitanju toka podataka i dosta *biolerplate* koda. Compose je u potpunosti zasnovan na Kotlin programskom jeziku i svaki element sučelja se može predstaviti funkcijom koja je naznačena kao *Composable* funkcija. Ove funkcije imaju pristup gradivnim elementima sučelja a najosnovniji od njih su *Column, Row, Box, Surface* koji omogućuju slaganje elemenata u određenom redoslijedu ili rasporedu. Ovi gradivni elementi se razlikuju od onih u *XML-u* po tome što je svaki element također *Composable* funkcija. Ovo znači da gniježdenje ovih elementa ne uzrokuje smanjenje performansi korisničkog sučelja što je od iznimne važnosti. Developer ima slobodu stvaranja raznoraznih tipova sučelja bez brige da će to utjecati na performanse, čak i na starijim uređajima [12].

Composable funkcije su zasnovane na principu rekompozicije. Podaci koji se prikazuju na sučelju su parametri *Composable* funkcija. Ovi podaci su često dinamični i mijenjaju se kako korisnik koristi aplikaciju te ih treba konstantno osvježavati. *Composable* funkcije zbog toga imaju stanje odnosno, *state,* i ovaj poseban tip podatka omogućuje *Composable* funkcijama da automatski osvježe podatke koji se mijenjaju dokle god su ti podaci tipa *state* [12].

3.5 Flow

Flow je poseban tip podatka koji je specifičan za Kotlin programski jezik. Ovaj tip podatka je specifično stvoren za potrebe Android aplikacija i omogućava automatsko slanje novih podataka kada god se oni promjene. *Flow* predstavlja kolekciju određenih podataka. To mogu biti primitivni tipovi podataka kao *Int, Boolean, String* i sl. ili može biti i neki kompliciraniji podatak poput klase [13].

Svaki *flow* automatski odašilje promjene kada god novi podaci stignu u njega a da bi se ti podaci prikupili i iskoristili potrebno je „pretplatiti“ se na njih. Dohvaćanje tih podataka je asinkrona operacija i mora se izvršiti unutak korutine. Korutine su pojednostavljene niti procesa koje omogućavaju asinkrono izvršavanje operacija kako se glavna nit ne bi opteretila i ostala zadužena samo za prikazivanje sučelja [13].

Pored *Flow* podatka postoje i specifične implementacije a to su *StateFlow* i *SharedFlow*. Ovo su optimizirani tipovi *flow* podataka koji su korisniki za Android aplikacije jer se direktno nadovezuju na *state* koji je potreban za *composable* funkcije. *StateFlow* zahtjeva inicijalnu vrijednost dok *SharedFlow* ne zahtjeva. *SharedFlow* također omogućava dublju kontrolu nad samim ponašanjem slanja i primanja podataka i ne koristi se često osim u specifičnim slučajevima gdje je ta kontrola potrebna [13].

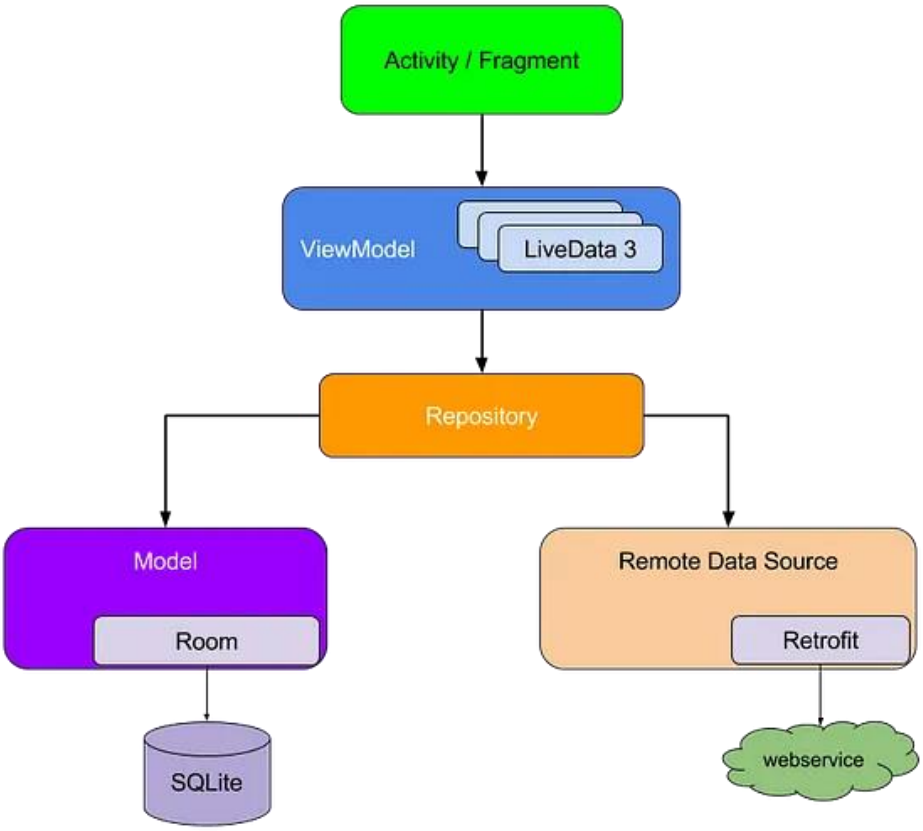
3.6 MVVM

MVVM(*Model-View-ViewModel*) je jako popularan i priznat oblikovni obrazac razvoja Android aplikacija koji je zasnovan na razdvajanju korisničkog sučelja i podataka. Sastoji se od 3 sloja.

Prvi sloj, *model* drži podatke i sve dijelove aplikacije koji su zaslužni za manipuliranje i dohvaćanje podataka(API klase, baza podataka i sl.).

Drugi sloj, *View*, predstavlja korisničko sučelje, u ovom slučaju *composable* funkcije koje predstavljaju ekrane.

Treći i zadnji sloj, *View-Model*, je posrednik između prva dva sloja. Ovaj sloj dostavlja *view* sloju podatke iz *model* sloja te također reagira na akcije iz *view* sloja, odnosno korisničke akcije te ovisno o akciji poduzima određene akcije koje se onda reflektiraju u *model* sloju [14].



Slika 3.4 Grafički prikaz MVVM arhitekture [15]

3.7 Room

Jedan od glavnih načina spremanja podataka lokalno kada su u pitanju Android aplikacije je Room. Room je službeni API zaslužen za jednostavno spremanje podataka u lokalnu bazu podataka. Android operativni sustav omogućava svojim aplikacijama da lokalno spremaju podatke u bazu podataka uz pomoć SQLite biblioteke. Room predstavlja apstrakciju nad SQLite bazom podataka i tako omogućava jednostavno spremanje podataka u vidu Kotlin ili Java klasa [16].

Room pruža nekoliko jednostavnih funkcija koje predstavljaju SQL operacije poput *Insert* i *Delete* ali također omogućava kreiranje vlastitih SQL naredbi i također vrši verifikaciju istih u sklopu *compiler-a* [16].

Svaka klasa predstavlja jednu tablicu unutar baze podataka i posjeduje sve funkcionalnosti klasičnih baza podataka poput primarnih i stranih ključeva, migracije, verzije i

sl. Sve operacije se definiraju unutar *DAO* sučelja. To je sučelje koje je zaslužena za direktnu komunikaciju sa bazom podataka i interpretiranje svih funkcija u SQL upite [16].

```
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

Slika 3.5 Klasa koja predstavlja jednu tablicu i podatke u njoj [16]

```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

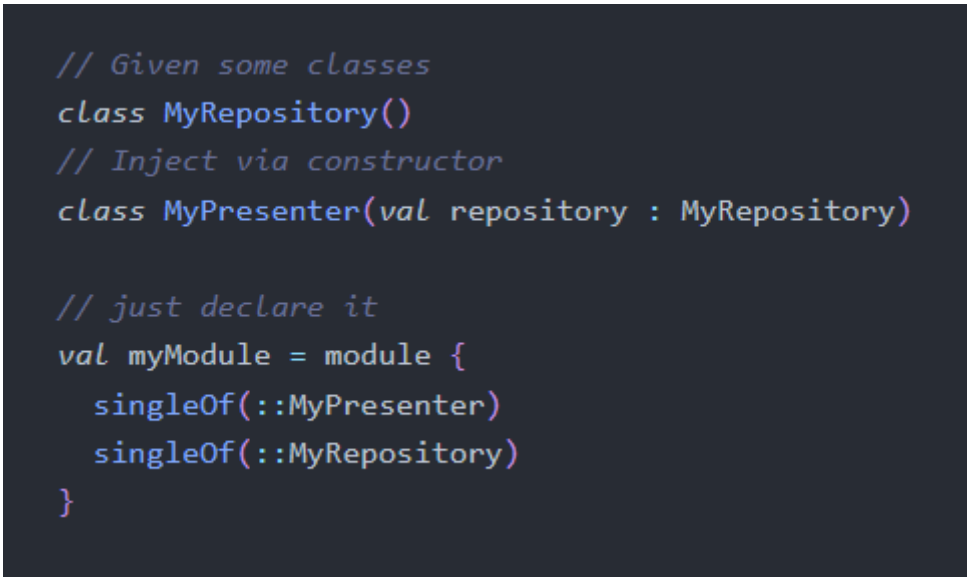
Slika 3.6 DAO sučelje u kojemu su definirane razne funkcije [16]

3.8 Dependency Injection(Koin)

Dependency injection [17] je jako važan pojam kada je u pitanju razvoj Android aplikacija. To je proces umetanja instanci objekata u konstruktor klasa koje ovise o tim objektima. Ako je potrebno stvoriti objekt *Računalo* onda je također potrebno imati komponente računala od kojih je ono sastavljeno, kao npr. *Procesor*. Dakle prilikom stvaranja objekta *Računalo* potreban nam je prvo objekt *Procesor*. Ovo se možda čini jednostavno, samo je potrebno napraviti taj objekt i ubaciti ga u konstruktor objekta *Računalo*. No to nije jedini objekt o kojem ovisi računalo. Vrlo brzo se klase mogu zakomplicirati i povećati i potrebno je puno ponavljanja i stvaranja novih objekata koji ovise o nekim drugim objektima kada god želimo sastaviti novo računalo. Ovdje nastupa *dependency injection*. *Dependency injection* omogućava definiranje potrebnih objekata na jednom mjestu u vidu singletona i onda se

instance tih objekata jednostavno ubacuju gdje god su oni potrebni. Kod razvoja Android aplikacija ova pojava je česta. *ViewModel* ovisi o *Repository* objektu a *Repository* objekt ovisi o *DAO* sučelju. Vrlo često se pojavljuje lančana ovisnost raznih objekata i ručno stvaranje tih objekata znatno komplicira sam kod aplikacije i troši bespotrebne resurse i performanse [17].

Koin je biblioteka koja omogućava stvaranje instanci raznih objekata i sadržava posebne alate koji su specifični za Android, te je tako moguće stvarati optimizirane instance *ViewModel* klasa, *Factory* klasa i *Singleton* klasa. Koin također podržava dohvaćanje istih na optimiziran i siguran način za *Jetpack Compose* sučelja, pogotovo kada su u pitanju ViewModel klase [18].



Slika 3.7 Princip rada Koin biblioteke [18]

4. IMPLEMENTACIJA APLIKACIJE

Aplikacija je zasnovana na ranije spomenutoj MVVM arhitekturi te u skladu s tim implementacija aplikacije se sastojala od rada na 3 zasebna dijela koja su međusobno povezana.

4.1 Model

Model sloj se sastoji od baze podataka, repozitorija te *dependency injection* modula. Ovo je sloj zaslužan za držanje i obradu svih podataka koji se koriste u aplikaciji te je to i prvi dio koji je bitno definirati i implementirati.

Za potrebe ove aplikacije baza podataka se sastoji od dvije tablice. Prva tablica sadržava podatke o svim korisnicima. Ova tablica sadrži podatke poput korisničkog imena, šifre te osnovnim podacima o korisniku poput njegove visine, težine, starosti i sl. Ove su informacije poprilično bitne s obzirom na tip aplikacije te je vrlo korisno imati ih trajno spremljeno. Druga tablica sadrži dnevne podatke koje korisnik sam unosi. Korsnik unosi dnevne podatke o količini tekućine, krvnom pritisku, broju koraka, količini sna te bilo koje vježbe koje je možda danas odradio.

```
6  @Entity(tableName = "user")
7  data class User(
8      @PrimaryKey
9      val username: String,
10     val password: String,
11     val height: Int,
12     val weight: Int,
13     val age: Int,
14     val gender: String,
15     val activity: Int,
16     val medications: List<String>
17 )
```

Slika 4.1 Tablica koja sadrži podatke o korisniku

```
@Entity(
    tableName = "user_daily_data",
    foreignKeys = [ForeignKey(
        entity = User::class,
        parentColumns = ["username"],
        childColumns = ["user"],
        onDelete = CASCADE
    )]
)
data class UserDailyData(
    @PrimaryKey
    val date: LocalDate,
    @ColumnInfo(index = true)
    val user: String,
    val water: Int,
    val bloodPressure: String,
    val steps: Int,
    val sleep: Int,
    val workouts: List<String>
)
```

Slika 4.2 Tablica koja sadrži dnevne podatke koje unosi korisnik

Slika 4.2 prikazuje tablicu o dnevnim podacima koje unosi korisnik. Ovdje je bitno naglasiti pojavu stranog ključa. Taj strani ključ drži referencu na korisnika koji je unijeo podatke. Tako je moguće lako se pozvati na podatke koji se odnose na tog korisnika.

Također je bitno spomenuti *DAO* sučelje koje sadrži sve potrebne funkcije i *SQL* upite. Oni su predstavljeni jednostavnim funkcijama koje su naznačene sa posebnim *SQL* upitima koji su pisani u *SQL* jeziku.

```kotlin
@Dao
interface HealthDao {
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertUser(user: User): Long

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertDailyUserData(userDailyData: UserDailyData): Long

    @Query("SELECT * FROM user WHERE username LIKE :username AND password LIKE :password")
    suspend fun validateUser(username: String, password: String): User?

    @Query("SELECT * FROM user_daily_data WHERE user LIKE :username AND date LIKE :date")
    fun getDailyData(username: String, date: String): Flow<UserDailyData?>

    @Query("SELECT * FROM user WHERE username LIKE :username")
    fun getUserData(username: String): Flow<User>

    @Query("SELECT * FROM user_daily_data WHERE user LIKE :username")
    fun getAllDailyEntries(username: String): Flow<List<UserDailyData>>
}
```

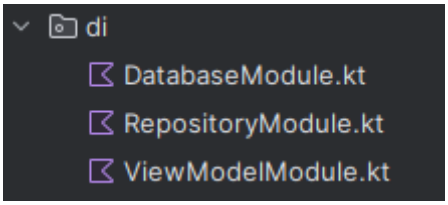
Slika 4.3 DAO sučelje

Repozitorij je jednostavna klasa koja je zadužena za pozivanje *DAO* funkcija i prosljeđivanje rezultata u *ViewModel* sloj. Ovaj sloj je zadužen i za manipulaciju podacima po potrebi i jako je bitan jer omogućava lakše testiranje i razumijevanje toka podataka.

```
class Repository(
    private val healthDao: HealthDao
) {
    suspend fun validateUser(username: String, password: String): User? {
        return healthDao.validateUser(username, password)
    }

    suspend fun insertUser(user: User) {
        healthDao.insertUser(user)
    }

    fun getUserDailyData(username: String, date: String): Flow<UserDailyData?> {
        return healthDao.getDailyData(username, date)
    }

    suspend fun insertDailyUserData(userDailyData: UserDailyData){
        healthDao.insertDailyUserData(userDailyData)
    }

    fun getAllDailyEntries(username: String): Flow<List<UserDailyData>> {
        return healthDao.getAllDailyEntries(username)
    }
}
```

Slika 4.4 Repozitorij

Zadnji dio *model* sloja je *dependency injection*. Koin radi na principu modula. Moguće je definirati više modula od kojih je svaki zadužen za stvaranje specifičnih objekata, poput recimo *ViewModel-a*. Tako u aplikaciji postoje 3 modula, *ViewModel*, *Repository* i *Database* modul.



Slika 4.5 DI moduli

Database modul je zaslužen za stvaranje *Room* baze podataka. S obzirom da nije baš poželjno stvarati objekte za pristup bazi podataka i DAO sučelju svaki put kada su oni potrebni, DI nam omogućava da definiramo singleton za oba tipa i njega referenciramo po potrebi.

Repository modul je zadužen za inicijalizaciju *Repository* klase. Ovaj objekt je potreban svakom *ViewModel-u* te zbog toga je također prigladno definirati ga pomoću *dependency injection-a*.

ViewModel modul je zaslužan za definiranje *ViewModel-a* za svaki ekran. *ViewModel-i* su posebni jer oni „žive"(ne čisti ih *garbage collector*) dokle god je ekran uz koji su vezani još uvijek na *back stack-u*(može se vratiti na taj ekran pritiskom na *back* dugme.). Tako da je potrebno biti posebno oprezan prilikom stvaranja objekata i tu *Koin* pomaže iznimno jer pruža posebne funkcije koje to omogućavaju na vrlo jednostavan i siguran način.

```
val databaseModule = module { this: Module
    single { this: Scope  it: ParametersHolder
        Room.databaseBuilder(
            androidContext(),
            HealthDatabase::class.java,
            name: "health_database"
        )
            .fallbackToDestructiveMigration()
            .build()
    }

    single { this: Scope  it: ParametersHolder
        get<HealthDatabase>().getHealthDao()
    }
}
```

Slika 4.6 Database modul

```
val viewModelModule = module { this: Module
    viewModelOf(::MainViewModel)
    viewModelOf(::LoginViewModel)
    viewModelOf(::SignupViewModel)
    viewModelOf(::HomeScreenViewModel)
    viewModelOf(::InputDataViewModel)
    viewModelOf(::DailyEntriesViewModel)
    viewModelOf(::UserViewModel)
}
```

Slika 4.7 ViewModel modul

4.2 ViewModel

ViewModel sloj je srednji sloj, posrednik između sučelja i podataka koji se nalaze u *model* sloju. Ovaj sloj se veže direktno za sučelje, odnosno svaki bitniji ekran aplikacije ima ViewModel klasu koja se veže za njega. Cilj *ViewModel* klase je da drži podatke koji će preživjeti kada korisnik navigira na neki drugi ekran ili zatvori aplikaciju privremeno. Također priprema te podatke za prikaz na sučelju i drži *state* za svoj ekran. *ViewModel* klasa neće biti uklonjena od strane *garbage collector-a* dokle god je ekran za koji je vezana na *back stack-u*, odnosno može se na njega vratiti pritiskom na *back* button.

```
class LoginViewModel(
    private val repository: Repository
) : ViewModel() {

    val username = mutableStateOf(TextFieldValue( text: ""))
    val password = mutableStateOf(TextFieldValue( text: ""))
    val isUsernameError = mutableStateOf( value: false)
    val isPasswordError = mutableStateOf( value: false)

    private val _user = MutableStateFlow<User?>( value: null)
    val user = _user.asStateFlow()

    fun validateUser() {
        viewModelScope.launch { this: CoroutineScope
            _user.value = repository.validateUser(username.value.text, password.value.text)
            if (user.value == null) {
                isUsernameError.value = true
                isPasswordError.value = true
            }
        }
    }
}
```

Slika 4.8 Login ViewModel

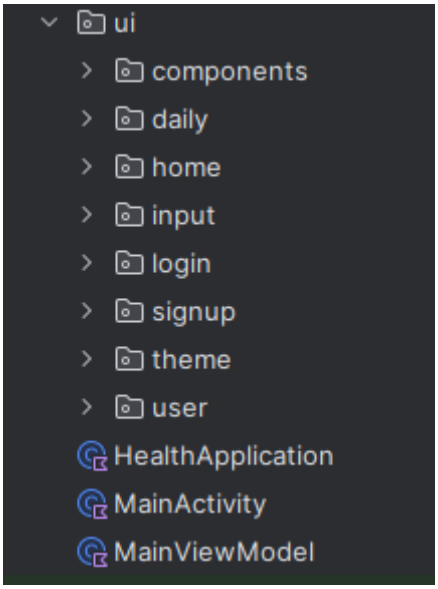
Slika 4.8 prikazuje *ViewModel* koji je vezan za Login ekran i drži stanje o unesenim podacima i poziva funkcije iz repozitorija i *DAO* sučelja za validaciju korisnika.

```
@RootNavGraph(start = true)
@Destination
@Composable
fun LoginScreen(
    navigator: DestinationsNavigator,
    mainViewModel: MainViewModel
) {
    val loginViewModel = koinViewModel<LoginViewModel>()
```

Slika 4.9 Dohvaćanje view modela

4.3 View

View sloj sadrži korisničko sučelje, u ovom slučaju to su *Composable* funkcije koje predstavljaju ekrane i pojedine komponente svakog od ekrana. Pošto su svi ekrani i komponenta u biti samo funkcije, svaki od njih prima određene parametre a najbitniji parametri su *state* i *callback* parametri. *State* parametri sadržavaju trenutno stanje podataka koji se prikazuju a *callback* parametri su funkcije koje se pozivaju kada se dogodi određeni događaj, npr. korisnik klikne na neki *button*.



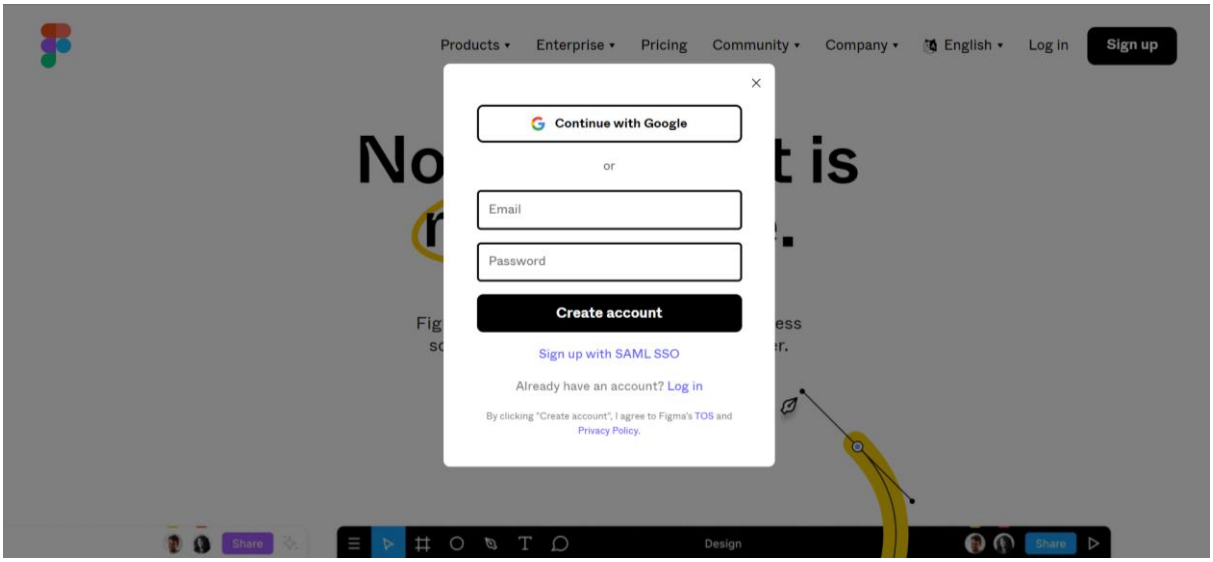
Slika 4.10 View sloj aplikacije

5. DIZAJN I FUNKCIONALNOST APLIKACIJE

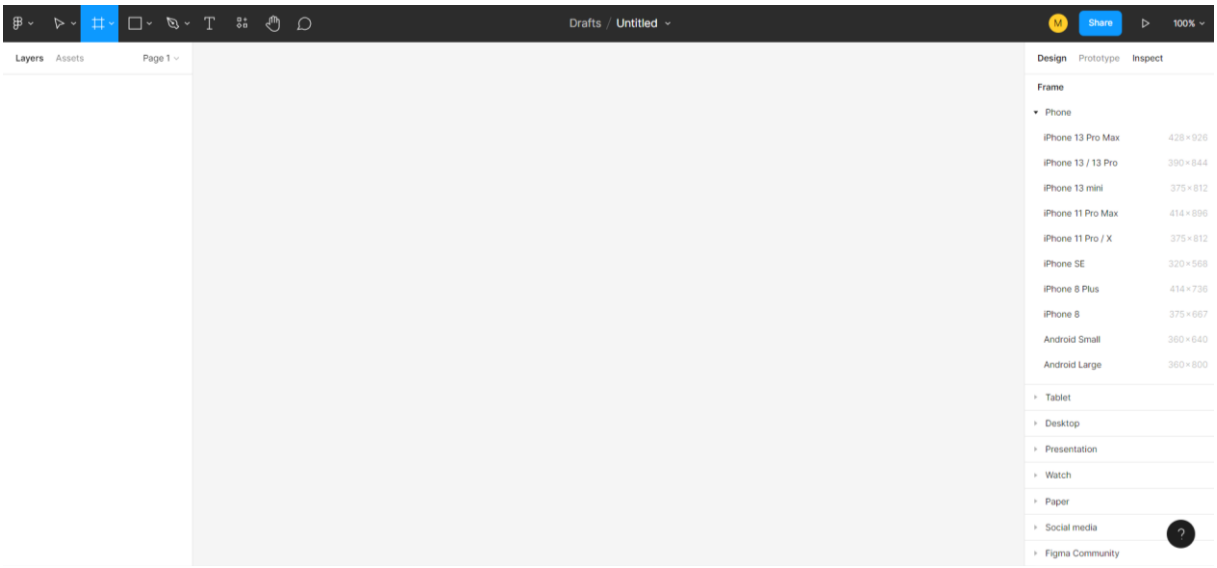
Osim izgleda, kod dizajna aplikacija unaprijed treba obratiti pažnju i na funkcionalnost, navigaciju, kao i prikazu na uređajima svih veličina, kako bi svi aplikaciju vidjeli upravo onako kako je zamišljena i dizajnirana. U nastavku će biti opisana *Figma,* kao jedan od alata koji se koristi pri izradi dizajna.

5.1 Figma

Figma je web aplikacija za uređivanje grafike i dizajna korisničkog sučelja. Koristi se za obavljanje svih vrsta grafičkog dizajna kao što su dizajn sučelja mobilnih aplikacija, dizajn prototipa, izrada postova za društvene mreže i slično. Razlikuje se od ostalih alata za uređivanje grafike jer radi izravno na internetskom pregledniku. To znači da se prethodnim projektima može pristupiti i započeti rad s bilo kojeg računala bez kupnje više licenci ili instaliranja softvera [19].



Slika 5.1 Prijava u Figmu

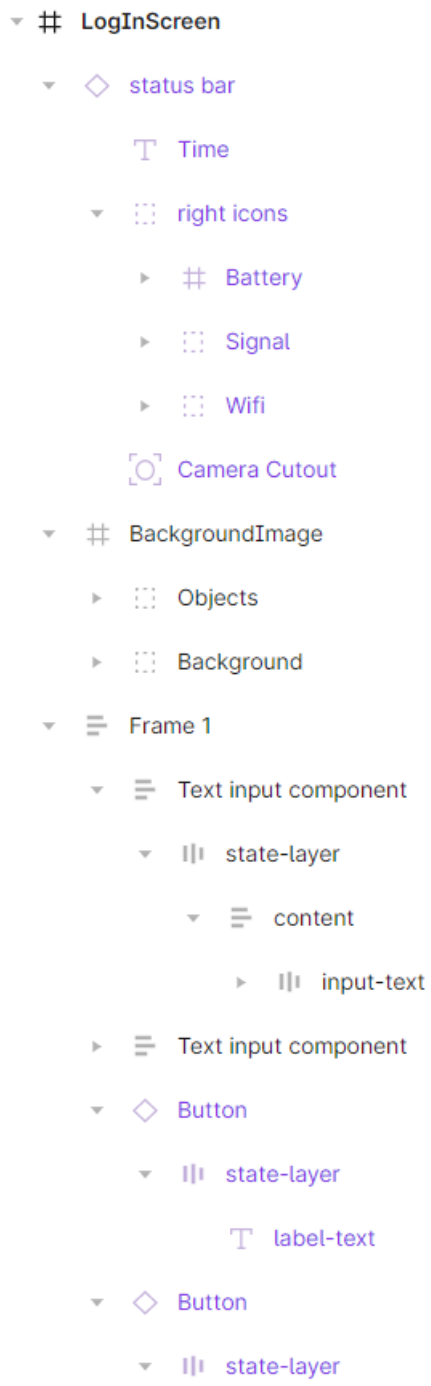


Slika 5.2 Korisničko sučelje Figme

5.2 Dizajn aplikacije za praćenje zdravstvenog stanja osobe

Prvi korak u izradi aplikacije jest kreiranje dizajna same aplikacije. Za izradu dizajna korišten je prethodno opisan alat, Figma. Prvi korak pri izradi dizajna aplikacije bio je osmisliti što želimo prikazati korisniku, i to izvesti na što pristupačniji način. Izgled aplikacije je vrlo važan, šarene boje privlače pozornost korisniku, jednostavno korisničko sučelje također će navesti korisnika da odluči isprobati i u krajnjem slučaju nastaviti koristiti aplikaciju. Aplikacija se sastoji od 6 različitih zaslona od kojih neki prikazuju različit sadržaj, ovisno o podacima koji se prikazuju. Za izradu svakog od ekrana korišten je Material Design 3 Kit [20] iz figme koji posjeduje sve moguće Material Design 3 komponente.

Najprije kreiramo početni zaslon u kojemu su od komponenti korišteni: Frame, Status Bar, TextInput i Button. Također, dodana je i fotografija zbog virtualno ljepšeg početnog zaslona.

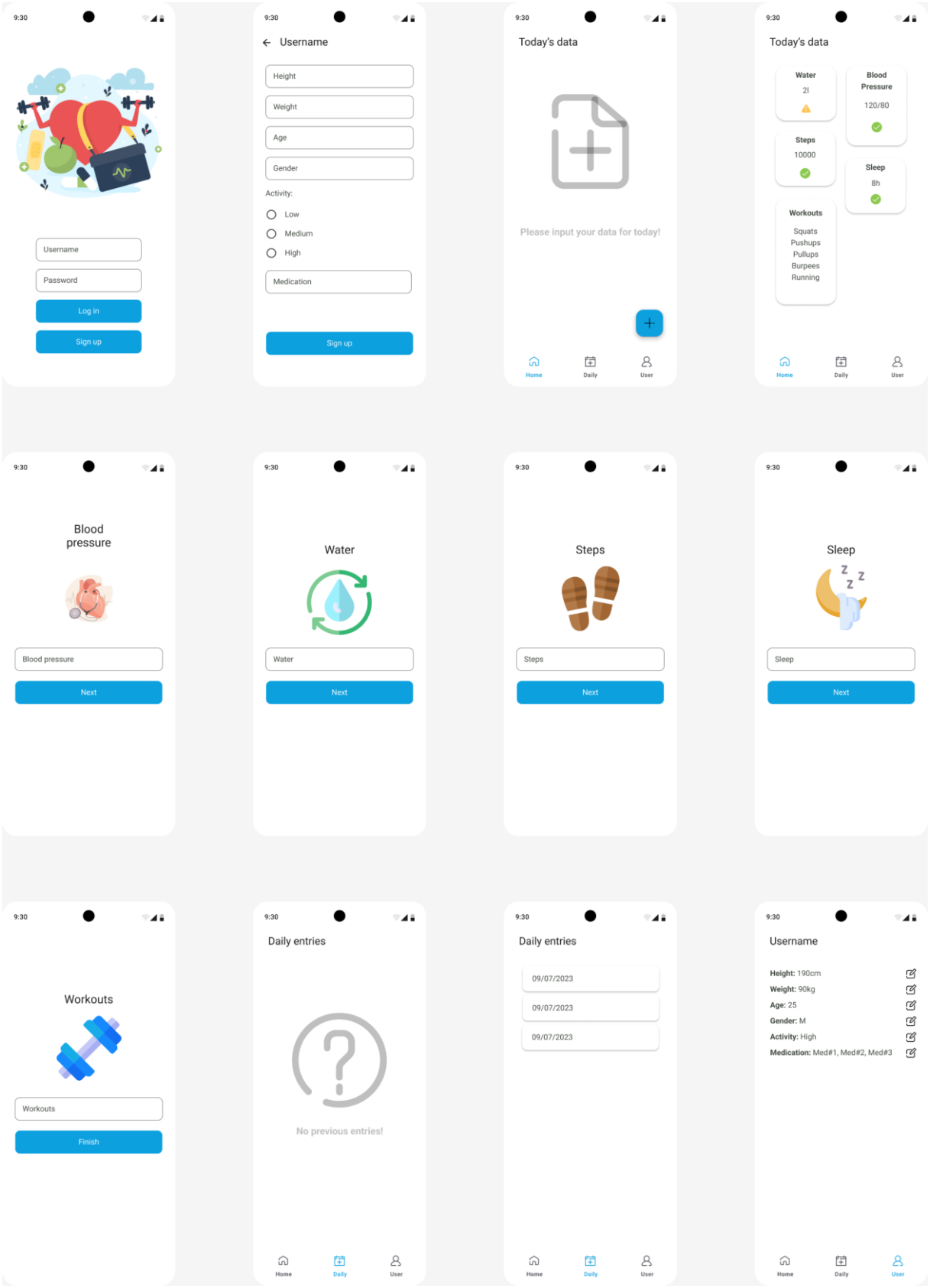


Slika 5.3 Korištene komponente na početnom zaslonu

Izrada profila korisnika koji se može prilagođavati nalazi se na drugom zaslonu, koriste se također slične komponente poput TextInput i Button.

Na trećem zaslonu otvara se mogućnost dodavanja podataka pritiskom gumba. Automatski nakon toga slijedi zaslon za ručni unos odabranog podatka.

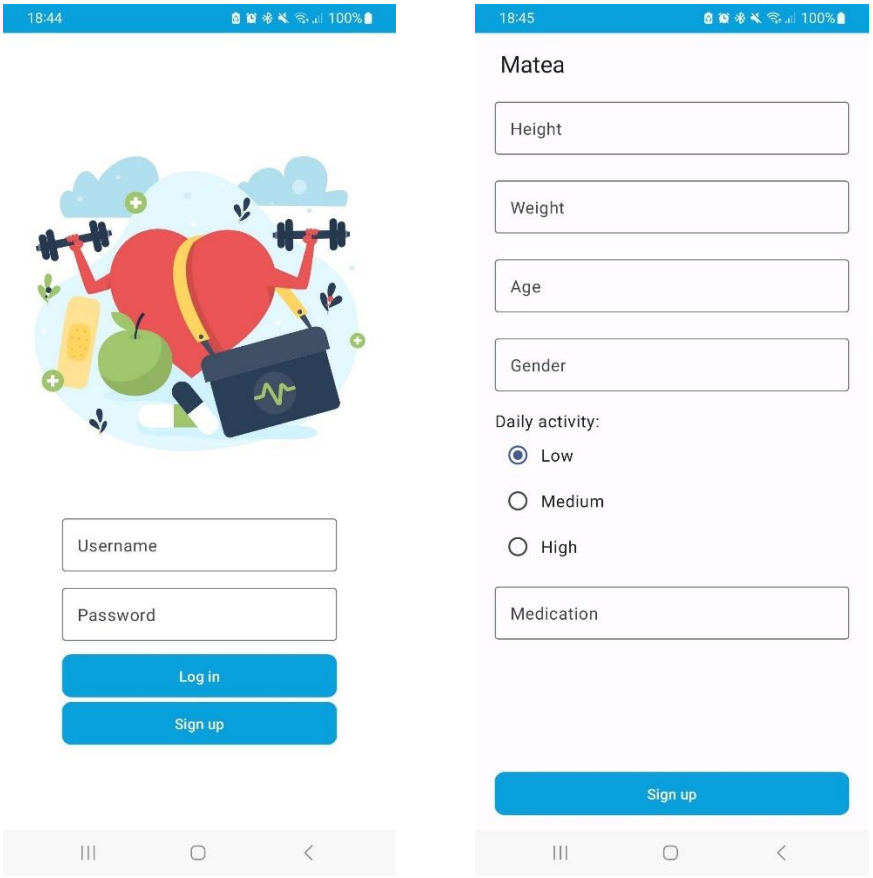
Nakon svih unesenih podataka, klikom na „Finish“ uneseni će se podaci spremiti u bazu podataka. Pritiskom na „Home“ gumb, bit će prikazani podaci uneseni toga dana. Također, moguće je pogledati i podatke unesene prethodnih dana pritiskom na „Daily“ gumb na traci za navigaciju. Detalji o korisniku mogu se vidjeti pritiskom na „User“ gumb.



Slika 5.4 Dizajn za aplikaciju izrađen u Figmi

5.3 Prijava korisnika

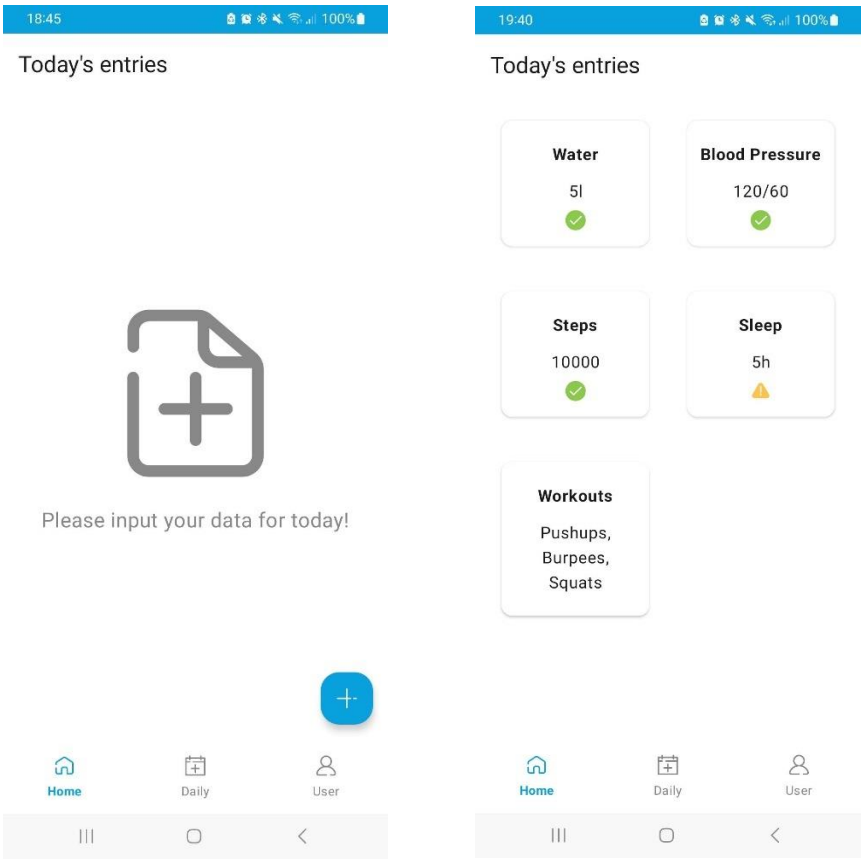
Prvi ekran aplikacije omogućava prijavu korisnika. Ako korisnik već ima postojeći račun može se prijaviti no ako nema onda mora napraviti račun. U tom slučaju klikom na „Sign up“ gumb korisniku se prikazuje ekran na kojem unosi dodatne podatke o sebi. Sustav je relativno jednostavan i nema dodatnih provjera osim da korisnik nije ostavio polja prazna.



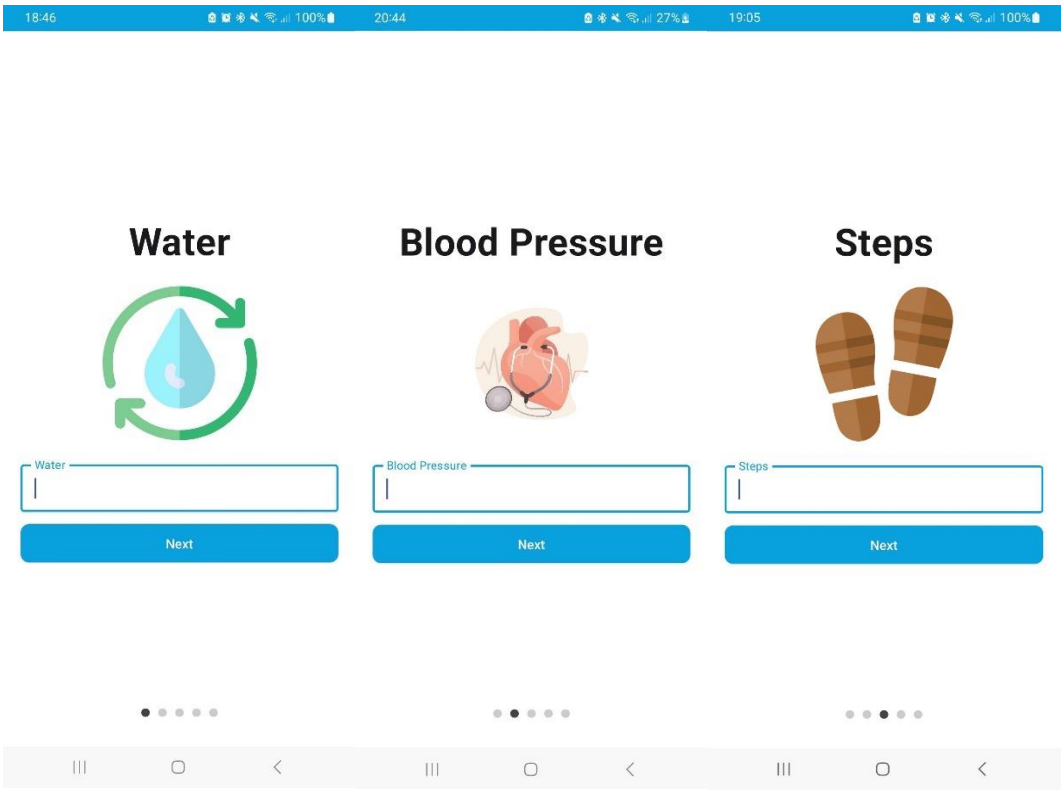
Slika 5.5 Prijava korisnika

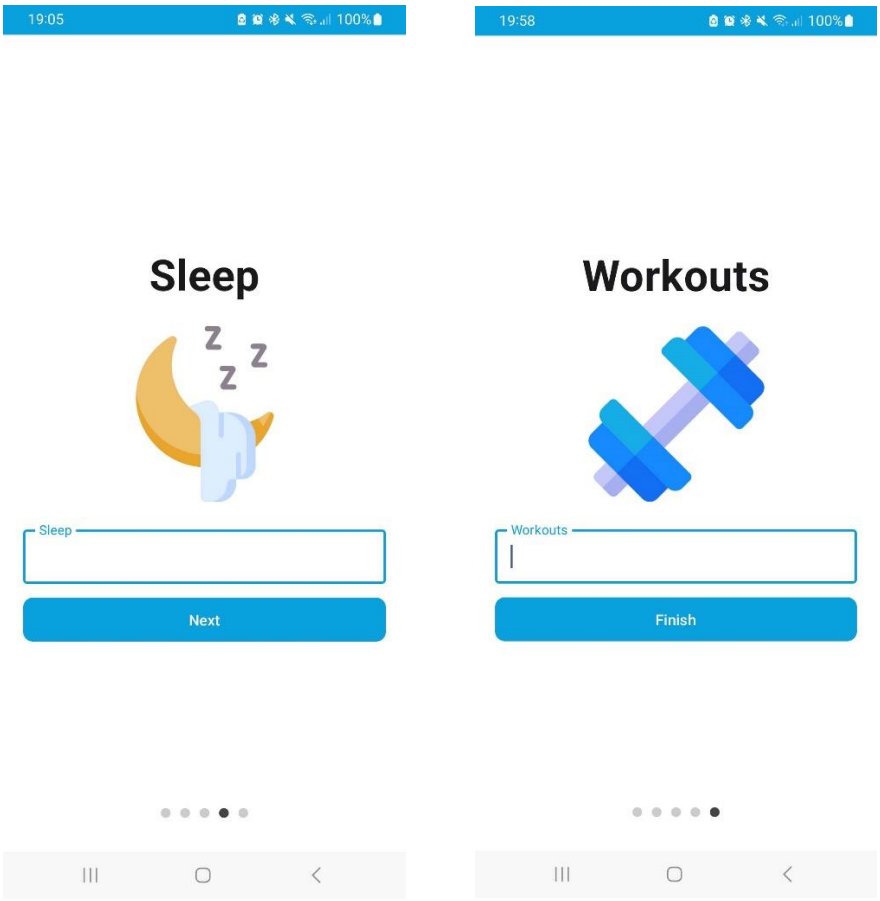
5.4 Unos dnevnih podataka

Nakon što se korisnik uspješno prijavi prikaziti će mu se ekran koji sadrži dnevne podatke. S obzirom da ove podatke unosi sam korisnik, ovaj će ekran biti prazan te će prikazati prikladnu poruku kako bi korisnik znao što se od njega očekuje da učini. Klikom na Floating Action Button pri dnu ekrana korisniku se prikazuje novi ekran koji mu nudi unos potrebnih podataka. Nakon unosa podataka korisnik se vraća na ekran koji prikazuje dnevne podatke i sada na njemu on vidi unesene podatke.



Slika 5.6 Prikaz dnevnih podataka

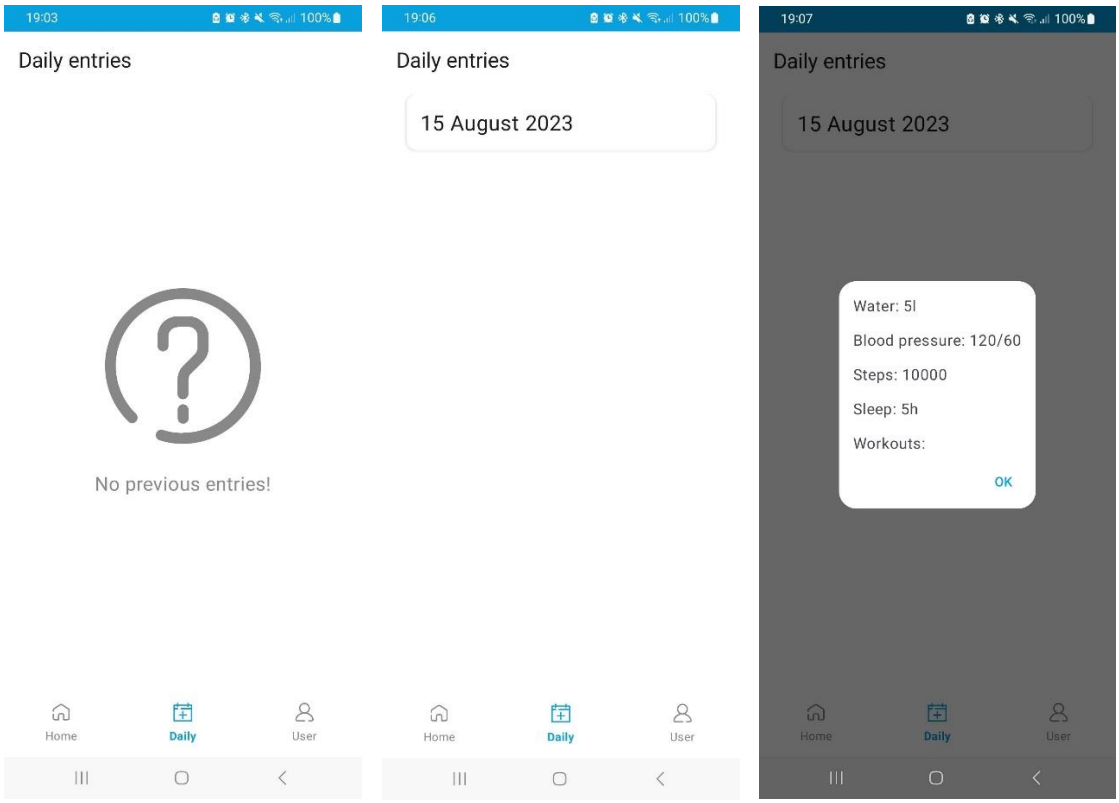




Slika 5.7 Unos dnevnih podataka

5.5 Pregled dnevnih unosa podataka

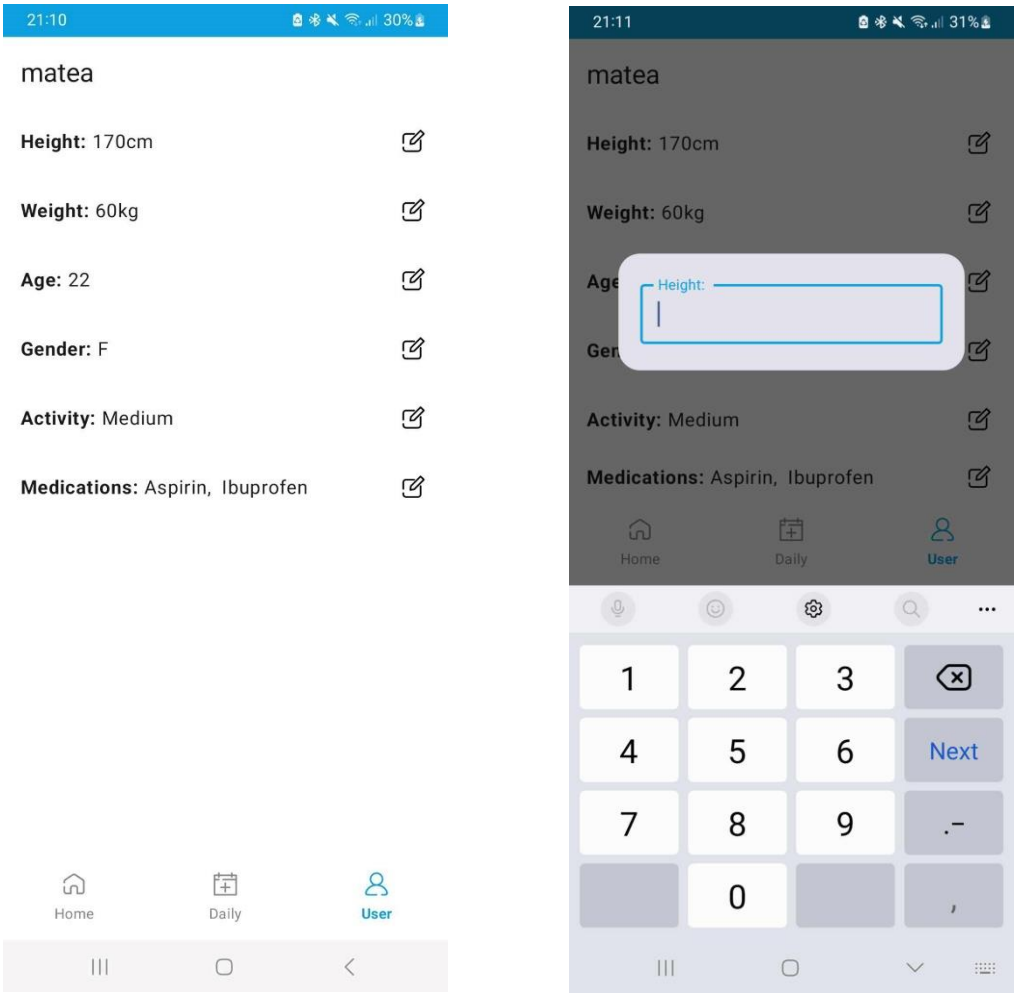
Aplikacija posjeduje navigacijsku traku pri dnu ekrana. Klikom na „Daily“ gumb korisnik navigira na ekran gdje može pregledati sve prijašnje unose sortirane po datumu unosa. Klikom na pojedini element prikazuje se *AlertDialog* prozor sa podacima koji su vezani za taj unos.



Slika 5.8 Pregled dnevnih unosa

5.6 Pregled korisnika

Klikom na treći gumb, „User“, korisniku se prikazuju osnovne informacije o njemu te mu se omogućava da ih izmjeni po volji. Izmjenu je moguće izvršiti pritiskom na ikonicu pored svakog podatka nakon čega se otvara *Dialog* prozor.



Slika 5.9 Prikaz korisnika

6. ZAKLJUČAK

Obzirom da se sve aplikacije na tržištu neprestano mijenjaju, nadograđuju i teže biti što boljima, u budućnosti bi se u konkretno ovoj aplikaciji moglo uvesti automatsko učitavanje podataka s uređaja za mjerenje poput pametnog sata ili narukvice. U ovu svrhu bi vrlo zgodno bilo iskoristiti Wear OS koji se vrlo jednostavno povezuje sa Android aplikacijama na mobilnom uređaju. To bi dovelo do još jednostavnijeg praćenja zdravstvenog stanja i zasigurno dodatno olakšalo život.

U svrhu testiranja, aplikacija je instalirana na Samsung Galaxy A32 koji koristi Android operacijski sustav i sve glavne značajke rade bez problema. Za izradu aplikacije korištene su preporučene tehnologije od strane Google-a te je implementirana moderna arhitektura i obrasci za izradu Android aplikacija poput MVVM arhitekture, repository-a, Jetpack Compose i sl. Aplikacija podržava samo svijetlu temu, omogućava autentikaciju korisnika i dinamično spremanje svih relevantnih podataka.

Osim Wear OS proširenja može se još dodati lokalizacija, tamna tema, integracija sa nekim API sustavom za detaljniju obradu podataka i sl.

LITERATURA

[1] K. Cimino, »What is Samsung Health? Features, compatibility, and more,« 19 Svibanj 2023. Dostupno na: https://www.androidauthority.com/samsung-health-3037491/.

[2] A. Lopez, »Huawei Health,« 2019.

Dostupno na: https://huawei-health.en.uptodown.com/android.

[3] Material Design, »Material 3 Design Kit,« 15 Kolovoz 2023.

Dostupno na: https://www.figma.com/community/file/1035203688168086460.

[4] »What is Figma? (And How to Use Figma for Beginners),« 26 Travanj 2023.

Dostupno na: https://www.theme-junkie.com/what-is-figma/.

[5] »The pragmatic Kotlin & Kotlin Multiplatform Dependency Injection framework,« 2023. Dostupno na: https://insert-koin.io/.

[6] K. Bhavya, »A quick intro to Dependency Injection: what it is, and when to use it,« 18 Listopad 2018.

Dostupno na: https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/.

[7] »Save data in a local database using Room,« 23 Kolovoz 2023.

Dostupno na: https://developer.android.com/training/data-storage/room.

[8] E. Fox, »Android Architecture starring Kotlin Coroutines, Jetpack (MVVM, Room, Paging), Retrofit and Dagger 2,« 6 Rujan 2019.

Dostupno na: https://proandroiddev.com/android-architecture-starring-kotlin-coroutines-jetpack-mvvm-room-paging-retrofit-and-dagger-7749b2bae5f7.

[9] RISHU_MISHRA, »MVVM (Model View ViewModel) Architecture Pattern in Android,« 18 Listopad 2022.

Dostupno na: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-in-android/.

[10] »Asynchronous Flow,« 25 Kolovoz 2023.

Dostupno na: https://kotlinlang.org/docs/flow.html.

[11] »Build better apps faster with Jetpack Compose,«

Dostupno na: https://developer.android.com/jetpack/compose.

[12] R. Z, »Threads post,« Srpanj 2023.

Dostupno na: https://www.threads.net/@richz/post/CuW_fXZOgPc?igshid=NTc4MTIwNjQ2YQ%3D%3D.

[13] »Kotlin for Android,« 11 Svibanj 2023.

Dostupno na: https://kotlinlang.org/docs/android-overview.html.

[14] »Android Studio Giraffe | 2022.3.1,« 25 Kolovoz 2023.

Dostupno na: https://developer.android.com/studio/releases.

[15] »Meet Android Studio,« 16 Kolovoz 2023.

Dostupno na: https://developer.android.com/studio/intro.

[16] »Android (operating system),« 26 Kolovoz 2023.

Dostupno na: https://en.wikipedia.org/wiki/Android_(operating_system).

[17] »Google Play,«

Dostupno na: https://play.google.com/store/apps/details?id=com.fitbit.FitbitMobile&hl=en&gl=US&pli=1.

[18] »App Store Preview,«

Dostupno na: https://apps.apple.com/app/id1433864494.

[19] »Light/Dark Huawei Health Dashboard Concept,«

Dostupno na: https://dribbble.com/shots/14484139/attachments/6168468?mode=media.

[20] »New S Health Lets Users Team Up to Tone Up,« news.samsung, 22 Kolovoz 2016.

Dostupno na: https://news.samsung.com/global/new-s-health-lets-users-team-up-to-tone-up.

SAŽETAK

Tema ovog rada bila je osmisliti i uspješno implementirati aplikaciju koja bi pomogla ljudima koji se bore s ubrzanim načinom života i manje pažnje posvećuju svom zdravlju pratiti one najvažnije zdravstvene stavke. U radu su objašnjene sve tehnologije i alati koji su se koristili pri izradi poput Jetpack Compose-a, Flow-a, MVVM-a, Room-a i drugih. Dizajn aplikacije je napravljen u Figmi, a kompletna aplikacija napisana je programskim jezikom Kotlin u Android Studiu. Pri kreiranju bilo je važno učiniti aplikaciju vrlo jednostavnom za svakodnevno korištenje.

Ključne riječi: Android, Flow, Jetpack Compose, Kotlin, zdravlje

ABSTRACT

The topic of this work was to design and successfully implement an application that would help people, who struggle with a fast-paced lifestyle and pay less attention to their health, to monitor the most important health parameters. All the technologies and tools that were used during creation, such as Jetpack Compose, Flow, MVVM, Room and others were explained in detail. The application design was made in Figma, and the complete application was written in the Kotlin programming language in Android Studio. While creating the aplication, it was important to make it very simple for everyday use.

Keywords: Android, Flow, Jetpack Compose, Kotlin, health