UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN INFORMACIJSKE TEHNOLOGIJE



Tehnična dokumentacija

**Projektni Seminar – Aplikacija za analizo treninga**

Avtorji: Andraž Anderle, Jan Jovan
Mentor: dr. Branko Kavšek

Koper, Junij 2024

Kazalo

[1. Uvod 3](#_Toc175549623)

[1.1. Opis problema 3](#_Toc175549624)

[2. Funkcijska specifikacija 3](#_Toc175549625)

[3. Uporabljena orodja in arhitektura 3](#_Toc175549626)

[1.2. Upravljanje stanja 4](#_Toc175549627)

[1.3. Vzorec in knjižnica BLoC 4](#_Toc175549628)

[4. Zaledni del 5](#_Toc175549629)

[1.4. Model 5](#_Toc175549630)

[1.5. Datoteka FIT 6](#_Toc175549631)

[1.6. Analiza treninga 6](#_Toc175549632)

[1.7. Flutter Isolates 6](#_Toc175549633)

Kazalo slik

[Slika 1:Arhitektura Bloc 5](#_Toc175549663)

# Uvod

Poročilo je namenjeno opisu lastnosti in funkciji aplikacije za analizo treninga. Sestavljeno je iz treh poglavij:

* Specifikacija zahtev sistema: opisane bodo vse zahteve sistema v podrobnosti
* Funkcijska specifikacija: namenjena pregledu vseh funkcionalnosti aplikacije
* Tehnična specifikacija: predstavljena bo dejanska tehnična izvedba in tehnologije, ki sestavljajo končani produkt

## Opis problema

Šport je v današnjem svetu zelo pomemben in napredek tehnologije ima velik vpliv tudi na športne aktivnosti in dosežke tekmovalcev. V zadnjih letih je opaziti velik porast uporabe tehnologije v različnih športnih disciplinah, kar omogoča bolj natančno spremljanje treningov, analiziranje zmogljivosti in na koncu tudi izboljšanje rezultatov. Ena izmed športnih disciplin, kjer je tehnologija še posebej pripomogla k napredku, je kolesarstvo.

Trening kolesarstva vključuje številne vidike, ki jih je mogoče natančno spremljati in optimizirati s pomočjo tehnologije. Kolesarji morajo biti pozorni na različne dejavnike, kot so vzdržljivost, moč, hitrost in tehnika vožnje. Z analizo teh dejavnikov lahko športniki prilagodijo svoje treninge, da dosežejo optimalne rezultate.

Napredne tehnologije omogočajo kolesarjem sledenje poti, merjenje srčnega utripa, spremljanje hitrosti in razdalje ter spremljanje drugih pomembnih parametrov, kot so kadenca in moč. S pomočjo teh podatkov lahko kolesarji natančno določijo svoje močne in šibke točke ter ustrezno prilagodijo svoj trening. Poleg tega pa tehnologija omogoča tudi primerjavo rezultatov skozi čas, kar kolesarjem pomaga spremljati njihov napredek in učinkovito načrtovati prihodnje treninge.

# Funkcijska specifikacija

Glavni namen aplikacije je omogočiti uporabniku vnesti svoje osebne podatke kot so teža, višina, najvišji srčni utrip ter območja srčnega utripa ter nalaganje njegovih aktivnosti, ki jih aplikacija sprocesira in mu poda uporabne nasvete o njegovi fizični pripravljenosti in napredku treninga.

Ponuja možnost nalaganja, vizualizacije ter analize podatkov iz `.fit` datotek, ki jih beležijo kolesarski računalniki. Ključni nameni aplikacije so:

* Nalaganje podatkov: Uporabniki lahko naložijo `.fit` datoteke iz svojih naprav za sledenje vadbi.
* Vizualizacija podatkov: Aplikacija prikaže osnovne in napredne kolesarske metrike, kot so hitrost, razdalja in višinska razlika, ter generira grafe in zemljevide poti.
* Analiza treninga: Uporabnikom omogoča izračun ključnih kazalnikov, kot sta Training Stress Score (TSS) in Form, za spremljanje intenzivnosti treningov in oceno trenutne telesne pripravljenosti.
* Napredne funkcije: Omogoča tudi primerjavo preteklih treningov

# Uporabljena orodja in arhitektura

Za razvoj uporabniškega vmesnika mobilne aplikacije smo izbrali ogrodje Flutter. Programski jezik Dart, ki je temelj za razvoj aplikacij v Flutterju, nam je omogočil hitro programiranje in prilagoditev vmesnika. Kljub temu, da je lahko aplikacija, razvita s Flutterjem, nekoliko počasnejša v primerjavi z namensko (ang. native) aplikacijo, je za naše zahteve povsem ustrezna. Kodo za čelnega dela smo razdelili na tri glavne sklope:

* Predstavitveni sklop: Ta del vsebuje kodo, ki je odgovorna za prikaz različnih zaslonov, ločenih glede na funkcije in vrsto uporabnika.
* Logični sklop: Ta komponenta skrbi za izvajanje glavne logike aplikacije, vključno z obdelavo uporabniških interakcij, manipulacijo s podatki in upravljanjem stanja aplikacije. Prav tako omogoča komunikacijo z različnimi storitvami.
* Storitveni sklop: Tukaj so implementirane knjižnice, ki omogočajo uporabo različnih funkcij mobilnih naprav, kot so shranjevanje podatkov v napravi, uporaba kamere in dostop do senzorjev.

S tem smo dosegli dobro organizirano strukturo aplikacije, ki omogoča enostavno vzdrževanje, razširjanje in prilagajanje v prihodnosti.

Aplikacijo smo razvili v razvojem okolju Android Studio, ki je eno izmed uradno podprtih okolji za razvoj Flutter aplikacij.

## Upravljanje stanja

Upravljanje stanja je vedno aktualna tema med razvijalci čelnih delov aplikacij. V številnih ogrodjih so na voljo različne metode za upravljanje stanja, kar pogosto vodi do različnih mnenj med razvijalci, ki se odločajo glede na svoje preference. Podobno je tudi pri Flutterju, saj ta sam po sebi ne določa specifičnega načina za upravljanje stanja, kot to počne na primer Jetpack Compose na Android platformi. Najosnovnejša metoda v Flutterju vključuje uporabo gradnikov *Stateful*, ki omogočajo shranjevanje stanja. Ta pristop je primeren le za preproste aplikacije ali funkcionalnosti, kjer ni potrebna kompleksna komunikacija in posodabljanje med različnimi deli aplikacije. To je privedlo do razvoja številnih orodij za upravljanje stanja, pri čemer so se kot najbolj priljubljene izoblikovale tri knjižnice: BLoC, Riverpod in GetIt. Za upravljanje stanja smo izbrali knjižnico BLoC, medtem ko smo za enostavnejše funkcionalnosti z manj uporabniške interakcije uporabili *Stateful* gradnike, saj ni bilo potrebe po bolj zapletenem pristopu.

## Vzorec in knjižnica BLoC

Vzorec BLoC omogoča ločevanje med uporabniškim vmesnikom in logiko ter podatki aplikacije, kot je prikazano na sliki 1. V predstavitvenem delu aplikacije je zgolj koda, ki skrbi za uporabniški vmesnik in pošilja dogodke logičnemu delu. Primer dogodka je lahko klik na gumb ali sprememba besedila v polju. Logični del nato obdela te dogodke in vrne novo stanje. Če pride do spremembe stanja, se posodobi samo tisti del uporabniškega vmesnika, ki je povezan s spremenjenim stanjem, kar omogoča boljšo zmogljivost in omogoča ponovno uporabo komponent v različnih delih aplikacije.

V vzorcu BLoC je logični del razdeljen na tri glavne komponente: dogodke, stanja in logiko. Dogodki so definirani kot razredi, ki predstavljajo različne vrste dogodkov, ki se lahko zgodijo. Ti dogodki lahko vsebujejo različne argumente, ki jih definiramo znotraj razreda dogodka. Na primer, dogodek za prijavo uporabnika bi vključeval dva parametra – e-poštni naslov in geslo. V razredu bi tako ustvarili dve spremenljivki tipa *String*.

Logični del nato prestreza te dogodke, ki jih sproži uporabniški vmesnik. Vsak dogodek sproži določeno funkcijo v logičnem delu. Če vzamemo primer prijave uporabnika, logični del prejme dogodek za prijavo z e-poštnim naslovom in geslom. Na podlagi teh podatkov se izvede poskus prijave uporabnika v sistem, v primeru napake ali uspeha pa se stanje ustrezno posodobi. Če so podatki napačni, se uporabniku prikaže obvestilo o neuspešni prijavi in poziv, naj poskusi znova. Če je prijava uspešna, se shranijo potrebni podatki in uporabnik se preusmeri na glavno stran aplikacije. Posodabljanje uporabniškega vmesnika se izvede le, če se stanje spremeni, kar zmanjša porabo procesorske moči v primerjavi z uporabo *Stateful* gradnikov.



Slika 1:Arhitektura Bloc

# Zaledni del

Za podatkovno bazo smo izbrali Firestore, ki je NoSQL baza, kar pomeni, da se njen podatkovni model razlikuje od tradicionalnih SQL modelov. Namesto klasičnih tabel in vrstic Firestore uporablja dokumente in zbirke. Dokument lahko primerjamo s tabelo, vendar brez vnaprej določenih struktur, kot jih poznamo v tradicionalnih SQL bazah. To pomeni, da pri ustvarjanju dokumenta ni potrebno vnaprej določiti vseh polj in njihovih podatkovnih tipov. Glavna prednost uporabe nerelacijske baze, kot je Firestore, je v hitrosti razvoja. Med razvojem aplikacije so se naši podatkovni modeli večkrat spremenili, kar bi pri uporabi tradicionalne relacijske baze zahtevalo brisanje vseh podatkov, prilagoditev definicije tabel in ponovno nalaganje podatkov. V primeru Firestore pa smo lahko preprosto ročno dodajali ali spreminjali atribute znotraj posameznih dokumentov v zbirkah, kar je znatno olajšalo in pohitrilo razvoj.

## Model

Za shranjevanje podatkov v podatkovno bazo smo uporabili 2 modela oziroma zbirki: uporabniki ter aktivnosti. V zbirko uporabnikov smo shranili njihove osnovne podatke ter vnešene podrobnosti o njihovi fizični pripravljenosti (cone srčnega utripa), ki so se kasneje uporabljali za računanje podrobnosti vnešenih aktivnosti. Glavna funkcionalnost aplikacije pa je seveda analiza treningov, ki smo jih shranjevali v ločeno zbirko.

## Datoteka FIT

FIT (Flexible and Interoperable Data Transfer) file je vrsta datotečne oblike, ki se pogosto uporablja za shranjevanje podatkov o aktivnostih v športu in fitnesu. Razvil jo je Garmin, da bi omogočil standardiziran način shranjevanja in izmenjave podatkov, povezanih z vadbo. FIT datoteke vsebujejo različne informacije, kot so:

1. GPS podatki: Podatki o lokaciji, kot so koordinate, višina, hitrost in prevožena razdalja, ki jih beležijo naprave, opremljene z GPS, kot so športne ure, kolesarski računalniki in pametni telefoni.
2. Fiziološki podatki: Podatki, ki se nanašajo na telesno zmogljivost, kot so srčni utrip, kadenca (frekvenca vrtenja pedal pri kolesarstvu), moč (v vatih) in drugi biometriki podatki.
3. Podatki o času: Podatki o trajanju aktivnosti, vključno s časom začetka in konca, skupaj s podrobnostmi o tem, kdaj so bili posamezni podatki zabeleženi med aktivnostjo.
4. Podatki o uporabniku: Informacije, kot so teža, starost, spol, ki jih naprava uporablja za izračunavanje določenih metrik, kot so porabljene kalorije.
5. Podatki o dejavnostih: Informacije o tipu dejavnosti (npr. tek, kolesarjenje, plavanje), intenzivnosti in drugih specifičnih lastnostih posamezne vadbe.

FIT datoteke so zasnovane tako, da so učinkovite in stisnjene, kar omogoča, da se velike količine podatkov shranijo in prenesejo hitro in z minimalnim porabo prostora. Zaradi svoje standardizirane narave so združljive z različnimi napravami in programsko opremo, kar olajša izmenjavo podatkov med različnimi sistemi.

## Analiza treninga

Najprej smo razvili algoritem za procesiranje vnešene aktivnosti. Uporabili smo programske jezik Python, saj ima zelo dobro podporo za delo z FIT datotekami. Osnovna ideja je bila, da bi ta algoritem tekel na nekem zalednem sistemu, kamor bi uporabnik naložil aktivnost in ta bi asinhrono analiziral trening in ob končani analizi v bazo podatkov zapisal celotno analizo. S tem uporabniku ne bi bilo potrebno čakati na končano analizo ter bi lahko v vmesnem času normalno uporabljal aplikacijo. Želeli smo uporabiti funkcije Firebase Cloud, ki bi omogočajo prav to. Po večih poskusih odpravljanja napak in iskanju vzroka, smo idejo opustili in se za to verzijo aplikacije odločilo, da bomo procesirali podatke kar na napravi

## Flutter Isolates

Za procesiranje podatkov kar na mobilni napravi smo nato uporabili Flutter isolates, ki so lahki, neodvisni izvajalni tokovi (execution threads) v jeziku Dart. Omogočili so nam paralelno izvajanje kode, ne da bi pri tem blokirali glavni uporabniški vmesnik (UI) niti.

**Glavne značilnosti:**

1. **Neodvisnost**: Vsak isolate ima svojo lastno pomnilniško površino. To pomeni, da ne deli stanja z drugimi isolates ali glavnim isolate, kjer se izvaja UI. Zaradi tega se izogibajo problemom, povezanim z varnostjo niti (thread safety), saj izolati ne morejo neposredno dostopati do pomnilnika drugih izolacij.
2. **Komunikacija**: Izolati med seboj komunicirajo prek sporočil. To se doseže z uporabo SendPort in ReceivePort. Ti objekti omogočajo, da en isolate pošlje podatke drugemu, vendar se podatki kopirajo, namesto da bi se delili, kar zagotavlja varnost in neodvisnost podatkov.
3. **Ustvarjanje in uporaba**: V Dart (in posledično v Flutter) je lahko isolate ustvarjen z uporabo funkcije *Isolate.spawn().* Ko se ustvarijo, se izolati lahko uporabljajo za izvajanje nalog, ki so preveč obsežne ali dolgotrajne, da bi jih izvedli v glavnem niti brez vpliva na zmogljivost aplikacije.
4. **Uporaba v Flutterju**: Flutter isolates so še posebej uporabni pri aplikacijah, ki zahtevajo težke izračune ali obdelavo velikih količin podatkov. Na primer, obdelava slik, analiza podatkov ali nalaganje velikih datotek v ozadju lahko potekajo v ločenih isolates, da ohranijo odzivnost UI.

Algoritem za procesiranje podatkov je bilo potrebno prepisati iz programskega jezika Python v jezik Dart, ki je na srečo imel dovolj dobro knjižnico za procesiranje FIT datotek, da nam to ni povzročilo preveč težav. Vsakič ko uporabnik naloži trening, aplikacija ustvari isolate in neodvisno od glavne niti na kateri teče uporabniški vmesnik, procesira podatke in ob zaključku analizo pošlje na podatkovno bazo, od koder lahko nato uporabnik dostopa do podatkov. Na povprečnem telefonu to procesiranje traja nekaj 5 sekund.

