Mobilna aplikacija za timske rekreativne sportove

Pavičić, Matija

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1582

MOBILNA APLIKACIJA ZA TIMSKE REKREATIVNE SPORTOVE

Matija Pavičić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1582

MOBILNA APLIKACIJA ZA TIMSKE REKREATIVNE SPORTOVE

Matija Pavičić

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1582

Pristupnik:	**Matija Pavičić (0036535308)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	izv. prof. dr. sc. Vinko Lešić

Zadatak: **Mobilna aplikacija za timske rekreativne sportove**

Opis zadatka:

Potrebno je izraditi aplikaciju pogodnu za iOS operativne sustave koja omogućuje okupljanje igrača za timske rekreativne sportove pristupom osobnih prijava u otvorene grupe. Pri tome su grupe definirane bilo od korisnika kao već postojećih okupljenih timova ili slobodnih termina za dostupne sportske terene. Korisničke postavke uključuju parametre kao što su vrsta sporta, vremenskog raspona termina i prostorne udaljenosti na temelju kojih se odabire prikaz dostupnih grupa. Prikaz dostupnih grupa treba omogućavati i dodatne značajke kao što su fotografije i recenzije korisnika. Aplikacija također treba uključivati i modul za pripremu i provođenje natjecanja uz praćenje uspjeha pojedinačnih timova. Aplikaciju je potrebno testirati na primjeru rekreativne košarke uz proizvoljne, modelske podatke.

Rok za predaju rada: 14. lipnja 2024.

Sadržaj

1. Uvod

U današnjem užurbanom svijetu, pronalazak vremena za rekreativne aktivnosti predstavlja izazov za mnoge, posebno kada je riječ o timskim sportovima. Ova mobilna iOS aplikacija pruža rješenje za sve one koji žele igrati timske sportove, ali često imaju poteškoća s okupljanjem dovoljno igrača. Omogućava korisnicima da objave kada i gdje igraju te da naznače koliko im je još igrača potrebno. Glavna ideja je da drugi korisnici mogu lako pronaći i pridružiti se igri koja odgovara njihovim preferencijama i slobodnom vremenu.

Kroz intuitivno sučelje, korisnici mogu pregledavati dostupne događaje i potvrditi svoje sudjelovanje ili stvoriti nove događaje kako bi pronašli nedostajuće igraće. Aplikacija također pruža mogućnost filtriranja prema sportu, lokaciji i vremenu, što dodatno olakšava pronalazak savršene prilike za igru.

U nastavku rada, detaljno se istražuju funkcionalnosti aplikacije, tehnički aspekti njezinog razvoja te njezine prednosti i izazovi. U prva dva poglavlja opisano je i objašnjeno razvojno okruženje i tehnologije koje su korištene za izradu aplikacije. U trećem poglavlju slijedi programsko rješenje najznačajnijih funkcionalnosti gdje je glavni naglasak na samom kodu te se četvrto poglavlje bavi izgledom i objašnjenjem aplikacijskog sučelja. Na kraju, dan je zaključak rada i pregled korištene literature.

2. Razvojno okruženje

Za potrebe ovog rada korišteno je razvojno okruženje Xcode, dostupno samo na Appleovim Mac računalima. Aplikacija je napisana u programskom jeziku Swift, uz SwiftUI koji je alat za izradu korisničkog sučelja.

2.1. Xcode

Xcode, dostupno u [1], je integrirano razvojno okruženje (IDE) koje je Apple razvio kako bi olakšao proces razvoja aplikacija za svoje platforme poput iOSa, macOSa, watchOSa i tvOSa. To je sveobuhvatan alat koji programerima pruža sve potrebne resurse za razvoj, testiranje i distribuciju njihovih aplikacija. Xcode dolazi s različitim komponentama, uključujući uređivač koda s podrškom za različite programerske jezike poput Swifta i Objective-Ca, grafičke alate za izradu korisničkih sučelja, integrirane programe za pronalaženje i ispravljanje grešaka u kodu (debugger), simulatore za testiranje aplikacija na različitim uređajima i mnoge druge korisne resurse.

Osim toga, Xcode pruža bogat ekosustav dodataka (plugins) i ekstenzija koji dodatno proširuju njegove mogućnosti. Programeri mogu koristiti Xcode za izradu različitih vrsta aplikacija, uključujući igre, poslovne aplikacije, alate za produktivnost i još mnogo toga. Integracija s Apple-ovim servisima kao što su App Store Connect omogućuje programerima jednostavno upravljanje distribucijom svojih aplikacija i praćenje performansi na tržištu.

Xcode se redovito ažurira kako bi se osigurala kompatibilnost s najnovijim verzijama Apple-ovih operativnih sustava i tehnologija te kako bi se uvodile nove funkcionalnosti i poboljšanja koja olakšavaju proces razvoja aplikacija. Zahvaljujući sveobuhvatnosti i moćnim alatima koje pruža, Xcode ostaje ključni alat za svakog programera koji želi stvarati aplikacije za Apple-ove platforme.

2.2. Swift

Swift, dostupno u [2], je moderan i sve popularniji programski jezik koji je Apple predstavio 2014. godine kao zamjenu za tradicionalni Objective-C.

S obzirom na svoju čistu sintaksu, sigurnost tipova i brzinu izvođenja, Swift je postao preferirani jezik za razvoj aplikacija za Apple-ove platforme poput iOSa, macOSa, watchOSa i tvOSa.

Jedna od glavnih karakteristika Swifta je njegova jednostavnost i lakoća učenja, što ga čini pristupačnim čak i za početnike u programiranju. Sintaksa Swifta je intuitivna i čitljiva, što olakšava pisanje i održavanje koda. Osim toga, Swift nudi brojne moderne funkcionalnosti kao što su *generics*, *closures*, *type inference* i *pattern matching*, što programerima omogućuje pisanje efikasnog i čistog koda.

Swift također nudi napredne značajke poput opcionalnih vrijednosti (*optional*) i upravljanja memorijom, što pomaže u sprječavanju uobičajenih grešaka u programiranju kao što su *null* reference ili curenje memorije. Osim toga, Swift je interoperabilan s Objective-Com, što znači da programeri mogu koristiti obje tehnologije unutar istog projekta, olakšavajući postupni prijelaz na novi jezik.

2.3. SwiftUI

SwiftUI, dostupno u [3], revolucionarni je radni okvir (eng. framework) za izradu korisničkih sučelja (eng. user interface, UI) kojeg je Apple predstavio 2019. godine kao modernu alternativu tradicionalnom UIKitu. SwiftUI omogućuje programerima da brzo i lako grade korisnička sučelja za svoje aplikacije koristeći deklarativni pristup. Umjesto pisanja imperativnog koda koji opisuje kako se korisničko sučelje treba konstruirati, programeri koriste SwiftUI za opisivanje željenog stanja sučelja, a zatim se iOS operativni sustav brine o tome kako to implementirati i održavati.

Jedna od glavnih prednosti SwiftUI-a je brzina razvoja aplikacija. Zahvaljujući jednostavnoj sintaksi i mogućnosti vizualnog pregleda promjena u realnom vremenu, programeri mogu brzo iterirati kroz različite dizajnerske opcije i brzo prilagoditi svoje sučelje prema potrebama korisnika. SwiftUI također podržava dinamičke promjene, što omogućuje aplikacijama da automatski reagiraju na promjene u stanju ili podacima bez potrebe za ručnim ažuriranjem korisničkog sučelja.

Osim toga, SwiftUI pruža ugrađenu podršku za animacije, geste i ostale interaktivne elemente, što olakšava stvaranje bogatih i privlačnih korisničkih iskustava.

Budući da je SwiftUI potpuno integriran s ostatkom Swift ekosustava, programeri mogu koristiti sve prednosti Swifta poput *generics*, *closures* i *type safety* u izgradnji svojih korisničkih sučelja.

3. Korištene tehnologije

Uz već navedeno razvojno okruženje i programski jezik, za potrebe aplikacije korištene su i tehnologije Swift Package Manager, Firebase i MapKit.

3.1. Swift Package Manager

Swift Package Manager (SwiftPM), dostupno u [4], je alat za upravljanje paketima i *build* sustav za Swift, kojeg je razvila Apple zajednica. SPM olakšava integraciju i upravljanje vanjskim bibliotekama unutar Swift projekata, ubrzavajući razvoj aplikacija. Koristi jednostavnu deklarativnu sintaksu za definiranje paketa putem datoteke `Package.swift`, koja opisuje ovisnosti, verzije i ciljeve.

Integracija SPM-a s Xcodeom, dostupna od verzije Xcode 11, omogućuje lako dodavanje, uklanjanje i ažuriranje paketa unutar projekata koristeći grafičko korisničko sučelje. To automatski rješava verzijske sukobe i upravljanje ovisnostima, smanjujući rizik od problema.

SPM podržava različite platforme, uključujući macOS, iOS, watchOS, tvOS i Linux, omogućujući višeplatformski razvoj s istim bibliotekama. Također podržava stvaranje privatnih i javnih repozitorija, olakšavajući dijeljenje koda unutar timova ili zajednica. Njegov fokus na brzinu i učinkovitost rezultira bržim build procesima.

Swift Package Manager je ključni alat za Swift programere, omogućujući jednostavno upravljanje ovisnostima, poboljšanu suradnju i brži razvoj aplikacija. Integriranost s Xcodeom i podrška za različite platforme čine ga neizostavnim dijelom Swift ekosustava.

3.2. Firebase

Firebase, dostupno u [5], je platforma za razvoj aplikacija koju je Google kupio i razvija je kao *Backend-as-a-Service* (BaaS). Namijenjena je olakšavanju razvoja mobilnih i web aplikacija pružajući niz alata i usluga koji omogućuju programerima da se fokusiraju na razvoj funkcionalnosti bez potrebe za upravljanjem serverima i infrastrukturom.

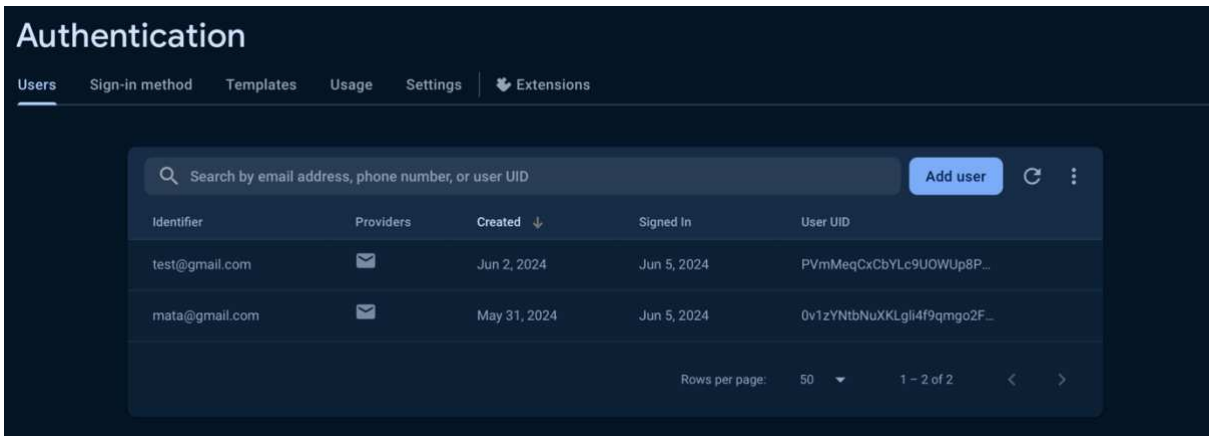
Firebase nudi različite module za autentifikaciju korisnika, baze podataka u realnom vremenu, skladištenje datoteka, analitiku i druge usluge koje pomažu u bržem i učinkovitijem razvoju aplikacija.

Jedna od ključnih prednosti Firebasea je njegova integracija s drugim Googleovim alatima i uslugama, što omogućuje jednostavno povezivanje s Google Cloud platformom, Google Analyticsom i drugim resursima. Firebase također pruža skup alata za razvoj softvera (eng. software development kit, SDK) za različite platforme uključujući iOS, Android i web, što omogućuje programerima da jednostavno integriraju njegove funkcionalnosti u svoje aplikacije.

3.2.1. Firebase Authentication

Firebase Authentication čiji je isječak prikazan na slici 3.1., je usluga unutar Firebase platforme koja omogućuje jednostavnu autentifikaciju korisnika. Ova usluga podržava različite metode prijave, uključujući prijavu putem emaila i lozinke, telefonskog broja, kao i prijavu putem trećih strana kao što su Google, Facebook, Twitter i GitHub. Firebase Authentication nudi siguran i pouzdan način za upravljanje korisničkim prijavama, što omogućuje programerima da se fokusiraju na razvoj aplikacija bez brige o složenim sigurnosnim pitanjima.

Integracija Firebase Authenticationa s drugim Firebase uslugama, poput Firestorea i Firebase Storagea, omogućuje jednostavno upravljanje korisničkim podacima i autentifikacijom unutar aplikacija. Programeri mogu koristiti jednostavne API-je za implementaciju autentifikacije, upravljanje korisničkim sesijama i pristup kontrolama. Firebase također nudi gotove UI komponente za autentifikaciju koje se mogu lako prilagoditi i integrirati u aplikacije, čime se dodatno ubrzava razvojni proces.



Sl. 3.1. Primjer registriranih korisnika

3.2.2. Firebase Firestore Database

Firebase Firestore Database je moderni, skalabilni NoSQL cloud bazni sustav koji je dio Firebase platforme. Dizajniran je za jednostavno pohranjivanje, sinkronizaciju i upite podataka u realnom vremenu za mobilne i web aplikacije. Firestore podržava strukture podataka temeljene na dokumentima i kolekcijama, omogućujući programerima da pohranjuju podatke u fleksibilnom i hijerarhijskom obliku.

Jedna od glavnih prednosti Firestorea je njegova sposobnost sinkronizacije podataka u realnom vremenu među svim povezanim klijentima. Ovo omogućuje izradu interaktivnih aplikacija koje trenutno ažuriraju podatke na svim uređajima korisnika. Firestore također podržava izvanmrežni način rada, što znači da aplikacije mogu nastaviti raditi čak i kada korisnici izgube internetsku vezu. Kada se veza ponovno uspostavi, Firestore automatski sinkronizira promjene.

Firestore nudi snažne mogućnosti upita i indeksa, omogućujući programerima da lako dohvaćaju i manipuliraju podacima. Integracija s Firebase Authenticationom osigurava siguran pristup podacima temeljen na korisničkim privilegijama. Osim toga, Firestore je skalabilan i može rukovati velikim količinama podataka i velikim brojem korisnika, što ga čini idealnim izborom za aplikacije koje zahtijevaju visoke performanse i pouzdanost.

Firebase, zajedno sa svojim uslugama Authentication i Firestore Database, pruža sveobuhvatan skup alata za brzi razvoj, implementaciju i upravljanje modernim mobilnim i web aplikacijama, olakšavajući programerima fokus na kreiranje izvrsnih korisničkih iskustava.

3.3. MapKit

MapKit, dostupno u [6], je framework razvijen od strane Applea koji omogućava integraciju mapa i geolokacijskih funkcionalnosti u iOS i macOS aplikacijama. Koristeći MapKit, programeri mogu dodavati ugrađene mape u svoje aplikacije, omogućavati korisnicima navigaciju, prikazivati oznake te obavljati složene geolokacijske zadatke.

MapKit također podržava napredne geolokacijske funkcionalnosti kao što su geokodiranje i pretraživanje lokacija. Geokodiranje omogućuje pretvaranje adresa u geografske koordinate, dok obrnuto geokodiranje pretvara koordinate u čitljive adrese. Ove funkcionalnosti su ključne za aplikacije koje trebaju precizno locirati korisnike ili specifične adrese na mapi.

Jedna od značajnih mogućnosti MapKita je podrška za dodavanje pretraživanja i ruta unutar mape. Ovo omogućuje korisnicima aplikacija da pretražuju lokacije, dobivaju upute za vožnju, hodanje ili javni prijevoz te vide prikaz ruta na mapi. Ove funkcionalnosti su ključne za navigacijske aplikacije ili bilo koje aplikacije koje trebaju pružiti korisnicima informacije o tome kako doći do određene lokacije.

Integracija MapKita u Swift aplikacije je jednostavna zahvaljujući dobro dokumentiranom API-ju i podršci unutar Xcodea. Korištenjem MapKit frameworka, programeri mogu obogatiti svoje aplikacije naprednim geolokacijskim funkcionalnostima i pružiti korisnicima interaktivna i korisna kartografska rješenja. MapKit je ključni alat za razvoj moderne iOS i macOS aplikacije koje zahtijevaju rad s mapama i geolokacijom.

4. Programsko rješenje modela u aplikaciji

U svrhu izrade ovog rada korišteno je samostalno stečeno znanje o potrebnim tehnologijama, paradigme objektno orijentiranog programiranja stečene na fakultetu te mnogi drugi izvori navedeni u literaturi.

4.1. Korisnik

Korisnik ima svoju kolekciju unutar Firebase Firestorea, definiranu na sljedeći način `Firestore.firestore().collection("users")`. Firebase sam stvara tu kolekciju ili radi promjene na postojećoj s danim nazivom. Podaci koje svaki korisnik pohranjuje prikazani su na slici 4.1., varijabla `initials` je takozvana computed varijabla, odnosno poprima vrijednost na temelju jedne ili više varijabli koje korisnik pohranjuje, u ovom slučaju stvara inicijale iz varijable `username`.

```
struct User: Identifiable, Codable {
    let id: String
    let username: String
    let email: String
    var eventIds: [String]?

    var initials: String {
        let formatter = PersonNameComponentsFormatter()
        if let components = formatter.personNameComponents(from: username) {
            formatter.style = .abbreviated
            return formatter.string(from: components)
        }
        return ""
    }
}
```

Sl. 4.1. Model korisnika

4.1.1. Registracija korisnika

Funkcija `createUser`, prikazana na slici 4.2., koristi Firebase za registraciju novih korisnika u aplikaciji. Prvo, funkcija pokušava kreirati korisnika s danim emailom i lozinkom koristeći `Auth.auth().createUser`.

Nakon uspješne registracije, korisnički ID (uid), korisničko ime i email pohranjuju se u bazu podataka. Podaci korisnika se enkodiraju i pohranjuju u kolekciju korisnika pomoću `setData` metode.

Na kraju, funkcija poziva `fetchUser`, prikazana na slici 4.4., kako bi dohvatila najnovije podatke o korisniku. Ako dođe do pogreške tijekom bilo kojeg koraka, funkcija ispisuje poruku o grešci.

```
func createUser(withEmail email: String, password: String, username: String) async throws {
    do {
        let result = try await Auth.auth().createUser(withEmail: email, password: password)
        self.userSession = result.user
        let user = User(id: result.user.uid, username: username, email: email)
        let encodedUser = try Firestore.Encoder().encode(user)
        try await usersCollection.document(user.id).setData(encodedUser)
        await fetchUser()
    } catch {
        print("DEBUG: Failed to create user with error \(error.localizedDescription)")
    }
}
```

Sl. 4.2. Funkcija za registraciju korisnika

4.1.2. Prijava korisnika

Za prijavu korisnika služi funkcija `signIn`, prikazana na slici 4.3., koja također koristi Firebase. Pokušava prijaviti korisnika s danim emailom i lozinkom koristeći `Auth.auth().signIn`. Nakon uspješne prijave, rezultat prijave, koji uključuje korisničke podatke, pohranjuje se u `userSession`. Zatim funkcija poziva `fetchUser`, prikazanu na slici 4.4., kako bi dohvatila najnovije podatke o korisniku. Ako dođe do pogreške tijekom prijave, funkcija ispisuje poruku o grešci.

```
func signIn(withEmail email: String, password: String) async throws {
    do {
        let result = try await Auth.auth().signIn(withEmail: email, password: password)
        self.userSession = result.user
        await fetchUser()
    } catch {
        print("DEBUG: Failed to login with error \(error.localizedDescription)")
    }
}
```

Sl. 4.3. Funkcija za prijavu korisnika

```swift
func fetchUser() async {
    guard let uid = Auth.auth().currentUser?.uid else { return }
    guard let snapshot = try? await usersCollection.document(uid).getDocument() else { return }
    self.currentUser = try? snapshot.data(as: User.self)
}
```

Sl. 4.4. Funkcija za dohvat trenutno prijavljenog korisnika

4.1.3. Brisanje korisnika

Za potrebe brisanja korisnika napisana je `deleteAccount` funkcija, prikazana na slici 4.5. Prvo provjerava postoji li trenutno prijavljeni korisnik koristeći `Auth.auth().currentUser`. Ako korisnik nije prijavljen, funkcija vraća grešku `URLError(.badURL)`. Prvo se poziva funkcija `removeDeletedUserFromAllEvents`, vidljiva na slici 4.6., koja briše sve događaje koje je korisnik stvorio i kojima se pridružio. Ako je korisnik prijavljen, funkcija briše korisnika iz kolekcije koristeći `usersCollection.document(user.uid).delete()`. Nakon toga, funkcija briše korisnički račun iz Firebase Authenticationa pozivom `user.delete()`. Na kraju, funkcija postavlja `userSession` i `currentUser` na `nil`, čime se uklanjaju podaci o trenutno prijavljenom korisniku iz lokalne sesije.

```swift
func deleteAccount() async throws {
    guard let user = Auth.auth().currentUser else {
        throw URLError(.badURL)
    }
    try await removeDeletedUserFromAllEvents(eventIds: currentUser?.eventIds ?? [])

    try await usersCollection.document(user.uid).delete()
    try await user.delete()

    self.userSession = nil
    self.currentUser = nil
}
```

Sl. 4.5. Funkcija za brisanje korisnika

```
func removeDeletedUserFromAllEvents(eventIds: [String]) async throws {
    guard let currentUserId = currentUser?.id else { return }
    let userCreatedEvents = try await eventsCollection.whereField("eventCreatorUserId", isEqualTo: currentUserId).getDocuments(as:
        Event.self)

    for eventId in eventIds {
        let data: [String:Any] = [
            "userIds" : FieldValue.arrayRemove([currentUserId])
        ]
        try await eventsCollection.document(eventId).updateData(data)
    }
    for event in userCreatedEvents {
        let joinedUserIds = event.userIds ?? []
        try await deleteEvent(eventId: event.id)

        for userId in joinedUserIds {
            try await removeEventFromUser(eventId: event.id, userId: userId)
        }
    }
}
```

Sl. 4.6. Funkcija za brisanje svih događaja povezanih s korisnikom

4.2. Događaj

Kako za korisnika, tako i za događaj postoji kolekcija u Firebase Firestoreu definirana s `Firestore.firestore().collection("events")`. Svaki događaj ima definirane podatke prikazane na slici modela `Event` koji je prikazan na slici 4.7. Model ima definirane dvije computed varijable, objašnjene u poglavlju o korisniku, koje predstavljaju geografske koordinate i format događaja.

```
struct Event: Identifiable, Codable {
    let id: String
    let sport: String
    let maxPlayers: Int
    let missingPlayers: Int
    let matchDate: String
    let matchStart: String
    let matchEnd: String
    let latitude: Double
    let longitude: Double
    let eventCreatorUserId: String
    var userIds: [String]?

    var coordinate: CLLocationCoordinate2D {
        CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
    }

    var format: String {
        let playersPerTeam = maxPlayers / 2
        return "\(playersPerTeam)v\(playersPerTeam)"
    }
}
```

Sl. 4.7. Model događaja

4.2.1. Stvaranje događaja

Funkcija `createEvent`, prikazana na slici 4.8., koristi Firebase za stvaranje novog događaja u aplikaciji. Primarno, funkcija prima različite parametre događaja kao što su sport, maksimalni broj igrača, broj igrača koji nedostaju, datum i vrijeme početka i završetka utakmice, te geografske koordinate događaja. Također prima ID korisnika koji kreira događaj.

Prvo, funkcija kreira novi `Event` objekt s jedinstvenim identifikatorom (`id`) generiranim s pomoću `NSUUID().uuidString`.

Zatim, provjerava postojanje trenutnog korisnika (`currentUser`) i, ako je korisnik prijavljen, dodaje njegov ID u popis korisničkih ID-ova povezanih s događajem i dodaje ID događaja u popis događaja trenutnog korisnika.

Nakon toga, funkcija enkodira `Event` objekt u format prikladan za Firestore s pomoću `Firestore.Encoder().encode(event)`. Enkodirani događaj se zatim pohranjuje u bazu podataka u kolekciji `eventsCollection` koristeći `setData` metodu s dokumentom označenim jedinstvenim ID-om događaja.

Ako dođe do greške tijekom stvaranja događaja ili pohranjivanja podataka, funkcija ispisuje poruku o grešci s detaljima o nastaloj grešci.

```
func createEvent(sport: String, maxPlayers: Int, missingPlayers: Int, matchDate: String, matchStart: String, matchEnd: String,
    latitude: Double, longitude: Double, eventCreatorUserId: String) async throws {
    do {
        let event = Event(id: NSUUID().uuidString,
                          sport: sport,
                          maxPlayers: maxPlayers,
                          missingPlayers: missingPlayers,
                          matchDate: matchDate,
                          matchStart: matchStart,
                          matchEnd: matchEnd,
                          latitude: latitude,
                          longitude: longitude,
                          eventCreatorUserId: eventCreatorUserId,
                          userIds: [currentUser?.id ?? ""]
        )
        try await addEventForUser(eventId: event.id)
        let encodedEvent = try Firestore.Encoder().encode(event)
        try await eventsCollection.document(event.id).setData(encodedEvent)
    } catch {
        print("DEBUG: Failed to create event with error \(error.localizedDescription)")
    }
}
```

Sl. 4.8. Funkcija za stvaranje događaja

4.2.2. Brisanje događaja

Funkcija `deleteEvent`, prikazana na slici 4.9., služi za brisanje događaja iz aplikacije. Proces započinje dohvaćanjem svih korisničkih dokumenata iz kolekcije `usersCollection` pomoću `getDocuments()`. Ako dohvaćanje korisničkih dokumenata nije uspješno, funkcija se prekida. Za svaki korisnički dokument, funkcija dohvaća podatke dokumenta i provjerava postoji li `eventId` u polju `eventIds`.

Ako korisnik sadrži `eventId` u svom popisu događaja, pokreće se `removeEventFromUser` funkcija prikazana na slici 4.10., kao asinkroni zadatak (*Task*). Ova funkcija uklanja određeni `eventId` iz popisa `eventIds` trenutnog korisnika.

Kreira se rječnik `data` koji sadrži ključ `"eventIds"` s vrijednošću `FieldValue.arrayRemove([eventId])`, što označava da se `eventId` treba ukloniti iz polja `eventIds`. Zatim se koristi `updateData(data)` metoda za ažuriranje korisničkog dokumenta u `usersCollection`, uklanjajući `eventId` iz popisa događaja.

Nakon što se događaj ukloni iz svih korisničkih podataka, događaj se briše iz kolekcije `eventsCollection` s pomoću `delete()`. Ovaj proces omogućuje sigurno brisanje događaja i ažuriranje svih relevantnih korisničkih podataka unutar aplikacije, osiguravajući da se događaj ukloni iz svih povezanih kolekcija u bazi podataka.

```
func deleteEvent(eventId: String) async throws {
    let event = try await eventsCollection.document(eventId).getDocument(as: Event.self)
    let allUsers = event.userIds ?? []
    for user in allUsers {
        let data: [String:Any] = [
            "eventIds" : FieldValue.arrayRemove([eventId])
        ]
        try await usersCollection.document(user).updateData(data)
    }
    try await eventsCollection.document(eventId).delete()
}
```

Sl. 4.9. Funkcija za brisanje korisnika

```
func removeEventFromUser(eventId: String) async throws {
    let data: [String:Any] = [
        "eventIds" : FieldValue.arrayRemove([eventId])
    ]
    try await usersCollection.document(currentUser?.id ?? "").updateData(data)
}
```

Sl. 4.10. Funkcija za brisanje događaja povezanog s korisnikom

4.2.3. Dohvaćanje spremljenih događaja

Za razne potrebe dohvaćanja događaja napisano je nekoliko različitih funkcija. Jedna od njih je `getAllEventsFilteredBySport` vidljiva na slici 4.11., koja vraća sve događaje iz kolekcije `eventsCollection` filtrirane po sportu koji korisnik odabere i sortirane po datumu i vremenu početka događaja. Ima nekoliko varijanti ove funkcije ovisno o upitima za dohvat podataka koji su traženi.

```
func getAllEventsFilteredBySport(sport: String) async throws {
    events = try await eventsCollection
        .whereField("sport", isEqualTo: sport)
        .order(by: "matchDate")
        .order(by: "matchStart")
        .getDocuments(as: Event.self)
}
```

Sl. 4.11. Dohvaćanje svih događaja

Nadalje, funkcija `getCreatedEventsForCurrentUserBySport` prikazana na slici 4.12., vraća sve događaje koje je stvorio trenutno prijavljeni korisnik, također filtrirane i sortirane na isti način kao i u prethodnoj funkciji.

```
func getCreatedEventsForCurrentUserBySport(sport: String) async throws {
    guard let currentUserId = currentUser?.id else { return }
    events = try await eventsCollection
        .whereField("eventCreatorUserId", isEqualTo: currentUserId)
        .whereField("sport", isEqualTo: sport)
        .order(by: "matchDate")
        .order(by: "matchStart")
        .getDocuments(as: Event.self)
}
```

Sl. 4.12. Dohvaćanje događaja stvorenih od strane prijavljenog korisnika

Posljednja potrebna funkcija je `getJoinedEventsForCurrentUserBySport`, prikazana slikom 4.13., ona vraća sve događaje kojima se trenutno prijavljeni korisnik pridružio, odnosno prijavio za sudjelovanje.

```
func getJoinedEventsForCurrentUserBySport(sport: String) async throws {
    guard let currentUserId = currentUser?.id else { return }
    events = try await eventsCollection
        .whereField("userIds", arrayContains: currentUserId)
        .whereField("sport", isEqualTo: sport)
        .whereField("eventCreatorUserId", isNotEqualTo: currentUserId)
        .order(by: "matchDate")
        .order(by: "matchStart")
        .getDocuments(as: Event.self)
}
```

Sl. 4.13. Dohvaćanje događaja kojima se korisnik pridružio

Sve navedene funkcije za rad s Firebase Firestore bazom podataka i Firestore Authentication alatom, asinkrone su funkcije definirane unutar `AuthViewModel` klase koja se onda koristi kroz zaslone napisane u SwiftUI-u.

Asinkronost funkcija osigurava da UX nema poteškoća s prikazivanjem sadržaja na zaslonu dok se čeka odgovor željene funkcije.

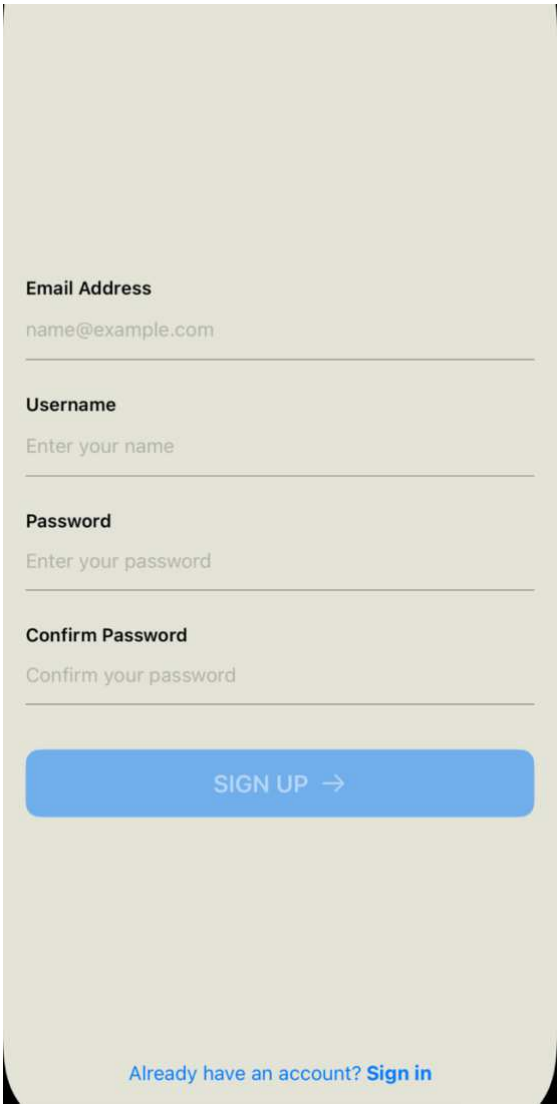
Svakom zaslonu kojem je potreban pristup tim funkcijama definira se `@EnvironmentObject var viewModel: AuthViewModel` i jednostavno se koriste kroz cijelu aplikaciju.

5. Izgled aplikacije

Mobilna aplikacija sastoji se od šest tipova zaslona korištenih kroz aplikaciju objašnjenih detaljnije u narednim potpoglavljima.

5.1. Zaslon za registraciju

Pri prvom ulasku u aplikaciju, korisniku se prikazuje zaslon za prijavu, odnosno registraciju ako nema postojeći račun. Zaslon za registraciju, prikazan slikom 5.1., sastoji se od četiri komponente za upis teksta, `TextField` za e-adresu i korisničko ime, `SecureField`. za zaporku i potvrdu upisa zaporke. Na slici 5.2. prikazani su uvjeti koji moraju biti ispunjeni kako bi se omogućio gumb „*Sign up*“.



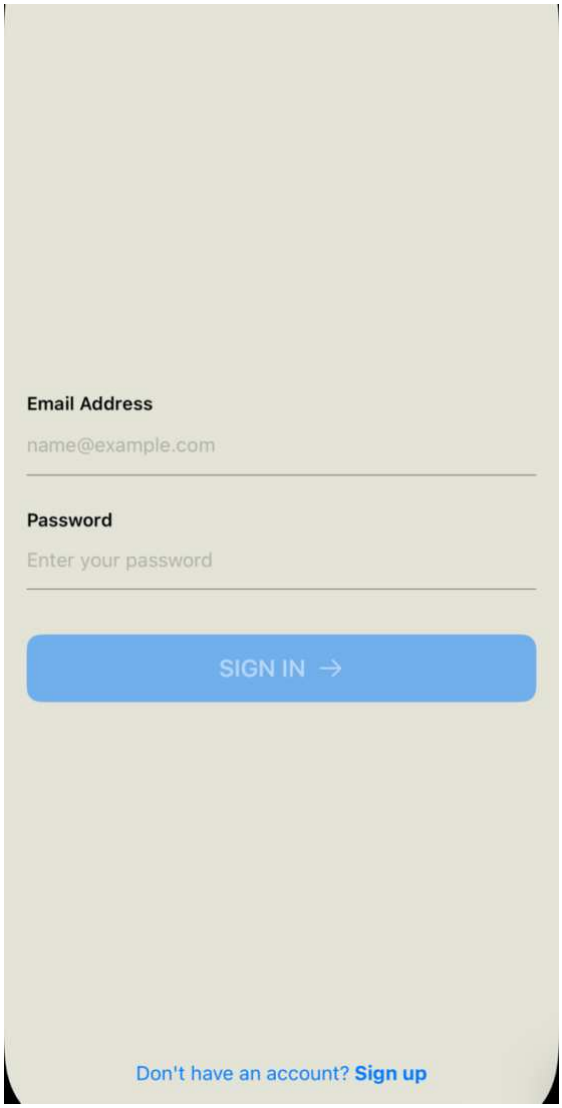
Sl. 5.1. Zaslon za registraciju

```
extension LogInView: AuthenticationFormProtocol {
    var formIsValid: Bool {
        return !email.isEmpty
        && email.contains("@")
        && !password.isEmpty
        && password.count > 5
    }
}
```

Sl. 5.2. Protokol za provjeru unesenih podataka pri registraciji

5.2. Zaslon za prijavu

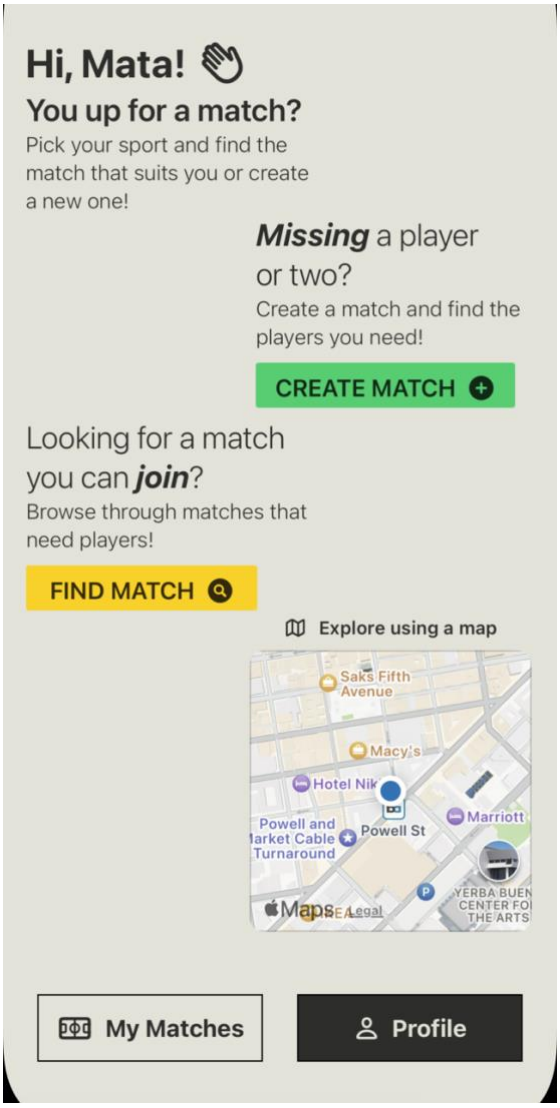
Zaslon za prijavu, vidljiv na slici 5.3., sastoji se od dvije forme za unos teksta, jedna `TextField` i druga `SecureField`. Jednako kao zaslon za registraciju, zaslon za prijavu ima opciju prijelaza na drugi od ta dva. Također, gumb „*Sign in*“ onemogućen je dok korisnik ne unese pravilnu formu e-adrese i zaporku odgovarajuće duljine.



Sl. 5.3. Zaslon za prijavu

5.3. Početni zaslon

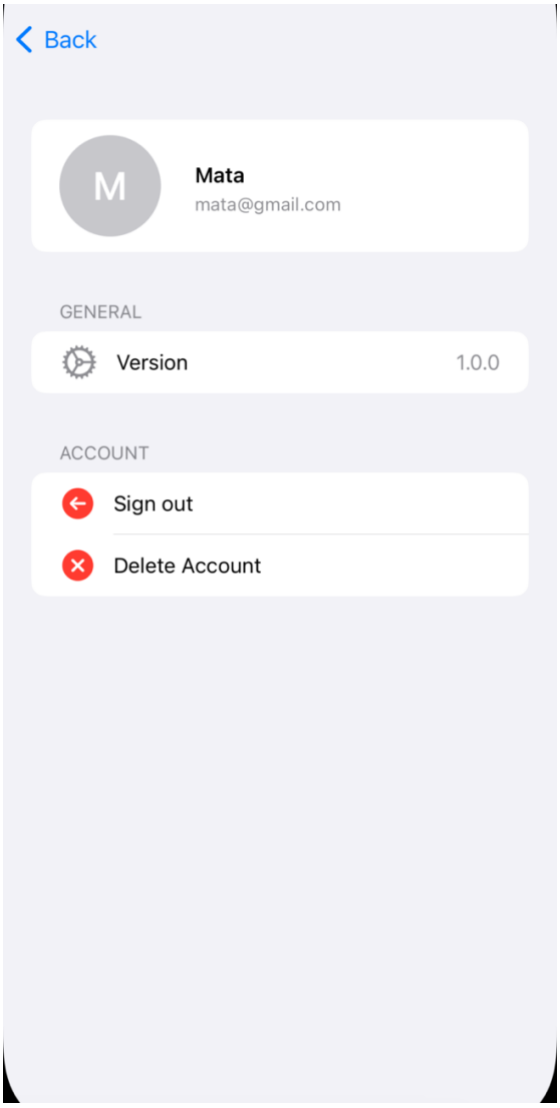
Nakon prijave ili registracije otvara se početni zaslon, prikazan slikom 5.4. Na zaslonu nalaze se dva gumba uz svoje opisne tekstove, jedan služi za otvaranje zaslona za stvaranje novog događaja, a drugi otvara zaslon s prikazanim dostupnim događajima koje su stvorili drugi korisnici. Nakon njih prikazana je mapa s lokacijom korisnika, pritiskom na nju otvara se zaslon mape s označenim dostupnim događajima. Na dnu se nalaze dva gumba, lijevi otvara zaslon sa svim korisnikovim događajima, a desni otvara zaslon korisničkog profila.



Sl. 5.4. Početni zaslon

5.4. Zaslon korisničkog profila

Zaslon korisničkog profila, prikazan na slici 5.5., sastoji se od osnovnih podataka o korisniku, verziji aplikacije te gumba za odjavu iz aplikacije i brisanje korisnika.



Sl. 5.5. Zaslon korisničkog profila

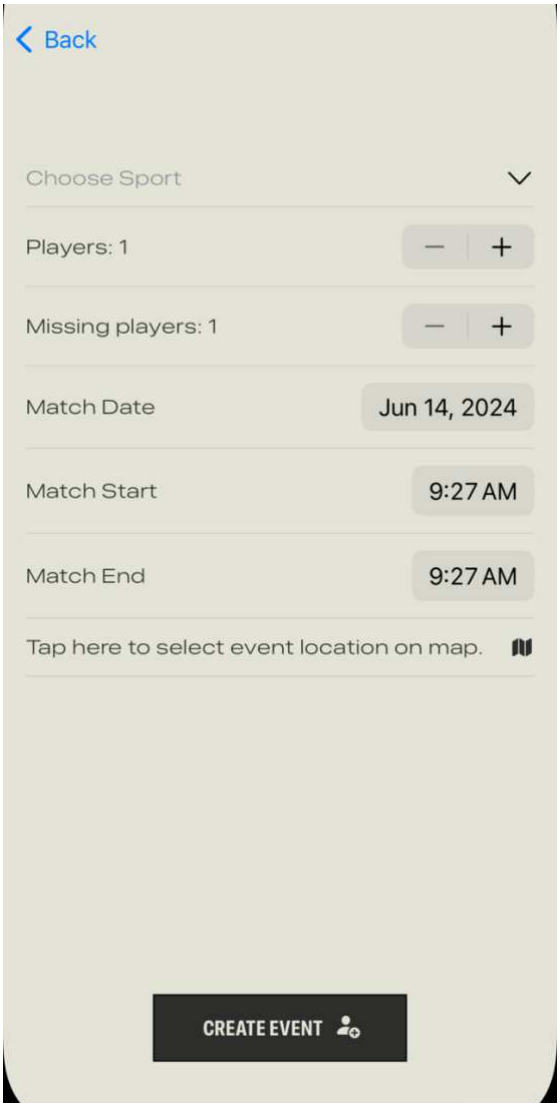
5.5. Zasloni događaja

Za pregledavanje postojećih i stvaranje novih događaja napravljeno je nekoliko različitih zaslona koji su objašnjeni i prikazani u sljedećim potpoglavljima.

5.5.1. Zaslon za stvaranje događaja

Zaslon za stvaranje novog događaja, prikazan slikom 5.6., sastoji se od raznih polja za unos podataka potrebnih za stvaranje novog događaja u bazi podataka. Prvi se nalazi padajući izbornik sportova, nakon njega dva polja zaslužna za unos maksimalnog broja igrača te broja igrača koji nedostaju.

Zatim, dolaze tri forme za unos datuma, odnosno `DatePicker` komponente, u prvom se bira datum održavanja događaja, u drugom satnica početka događaja te u trećem satnica završetka događaja. Na kraju se nalazi tekst s ikonicom, koja na pritisak otvara mapu s inicijalnom pozicijom mape na trenutnoj lokaciji korisnika. Korisnik na prikazanoj mapi stavlja oznaku gdje će se događaj održati.

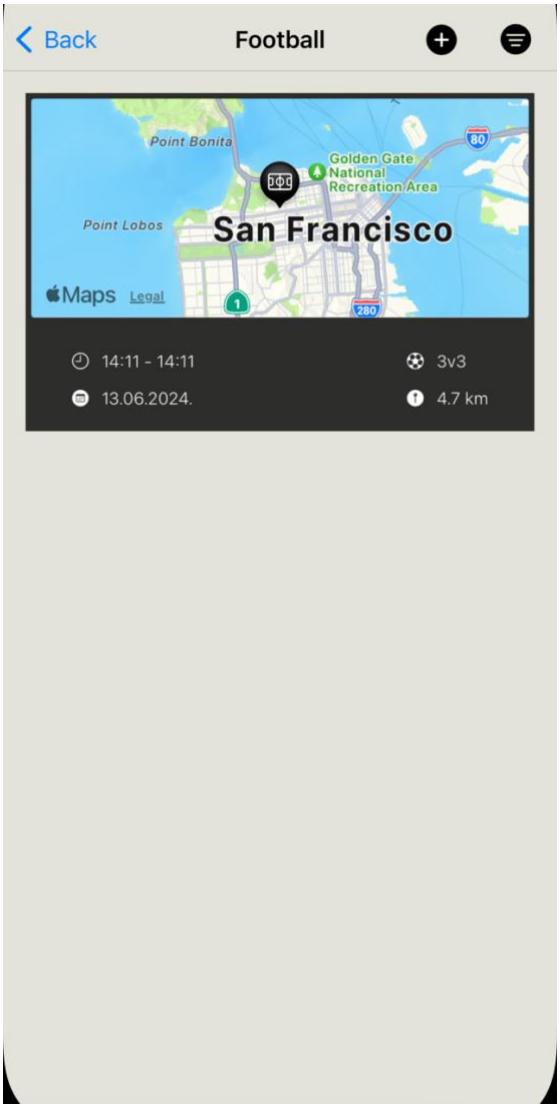


Sl. 5.6. Zaslon za stvaranje događaja

5.5.2. Zaslon za pregled dostupnih događaja

Za pregled dostupnih događaja služi zaslon prikazan na slici 5.7. Na vrhu zaslona naznačeno je za koji sport su trenutno prikazani dostupni događaji. U desnom gornjem kutu nalaze se dva gumba, lijevi za otvaranje zaslona za stvaranje novog događaja, desni za otvaranje dostupnih filtera za prikazivanje događaja.

Događaji se mogu filtrirati po sportu i po datumu održavanja događaja. Svaki događaj prikazan je u svom `EventBoxView`, u prvom planu je mapa s oznakom gdje se održava događaj, ispod mape datum i vrijeme trajanja događaja, format u kojem se igra i korisnikova udaljenost od lokacije događaja. Klikom na `EventBoxView` otvara se zaslon za pregled detalja događaja.

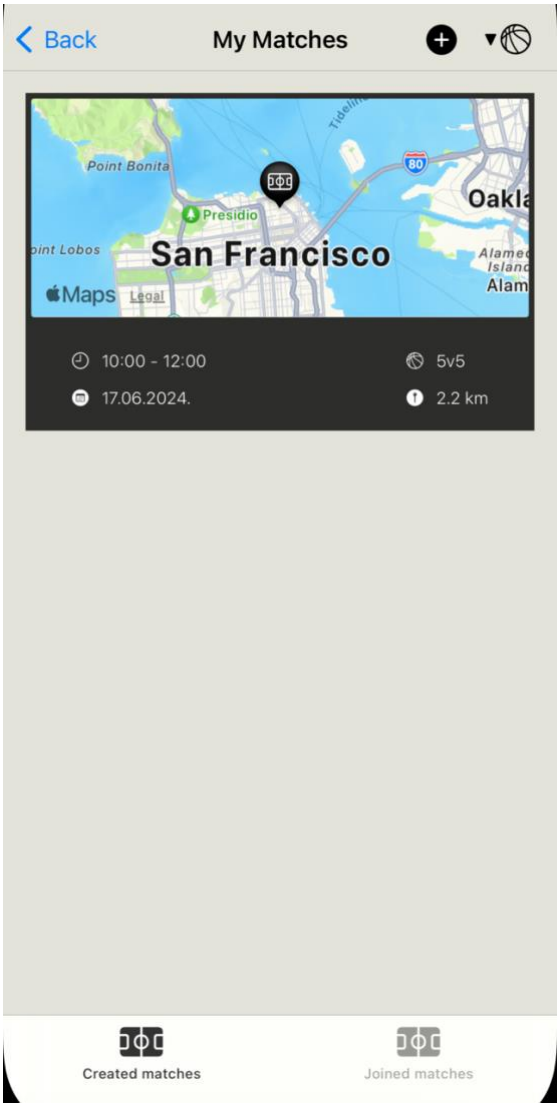


Sl. 5.7. Zaslon za pregled dostupnih događaja

5.5.3. Zaslon za pregled vlastitih događaja

Zaslon za pregled vlastitih događaja prikazuje dva izgledom ista zaslona, prikazana na slici 5.8., jedan za događaje koje je korisnik stvorio, a drugi za događaje kojima se korisnik pridružio.

Na vrhu zaslona nalaze se gumbi za stvaranje novog događaja i filtriranje prikazanih po sportu, za prikaz događaja korištene su `EventBoxView` komponente, klikom na svaku od njih otvara se zaslon s detaljima događaja.



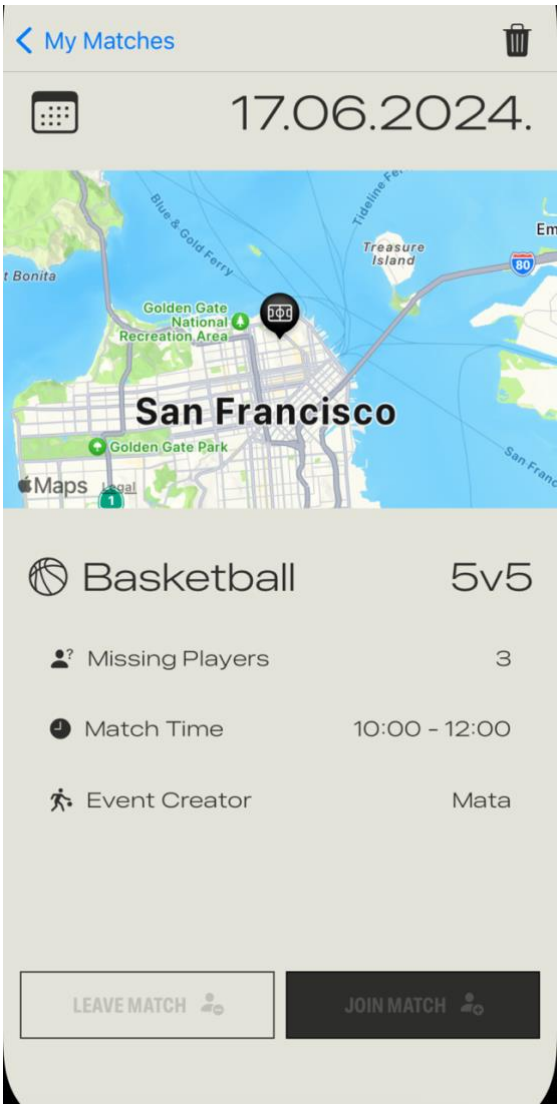
Sl. 5.8. Zaslon za pregled vlastitih događaja

5.5.4. Zaslon za pregled detalja događaja

Na vrhu zaslona za pregled detalja događaja, prikazan na slici 5.9., je datum održavanja događaja, nakon njega okvir mape koja u prvom planu ima lokaciju održavanja događaja, klikom na nju otvara se zaslon s mapom, istog izgleda kao zaslon na slici 5.10. samo s oznakom događaja za koji se pregledavaju detalji. Na tom zaslonu se mogu dobiti upute za dolazak do lokacije događaja od trenutne korisnikove lokacije, vrijeme potrebno do lokacije, te udaljenost korisnika.

Ispod mape nalaze se još detalji o sportu, formatu igre, početku i završetku događaja i korisničko ime stvaratelja događaja. Na dnu zaslona su dva gumba za pridruživanje, odnosno napuštanje otvorenog događaja, gumbi se ovisno o statusu korisnika unutar tog događaja mogu pritisnuti ili ne.

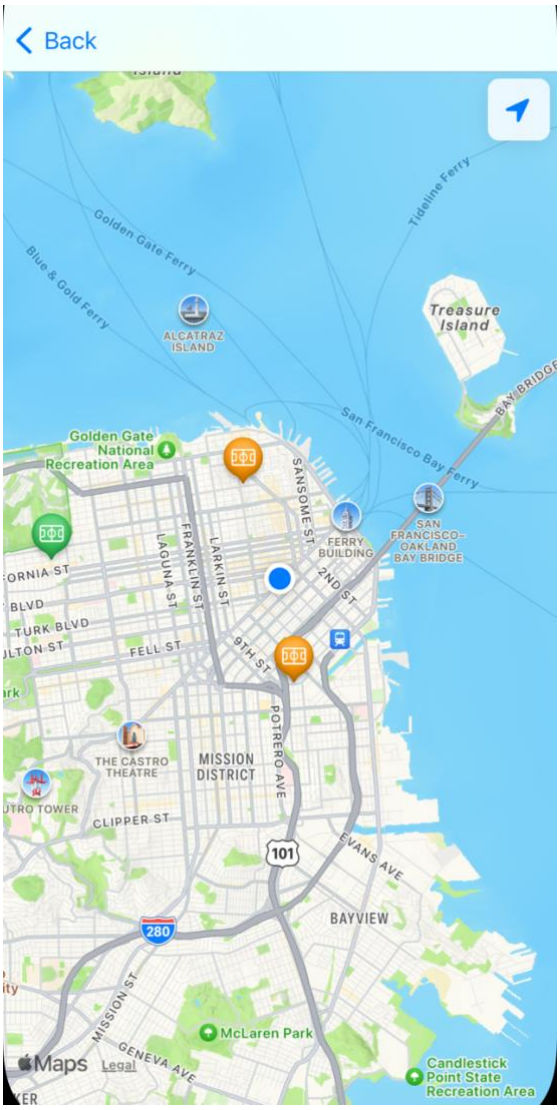
Ako je korisnik stvaratelj trenutno otvorenog događaja u gornjem desnom kutu prikazana je ikonica koja klikom na nju briše otvoreni događaj.



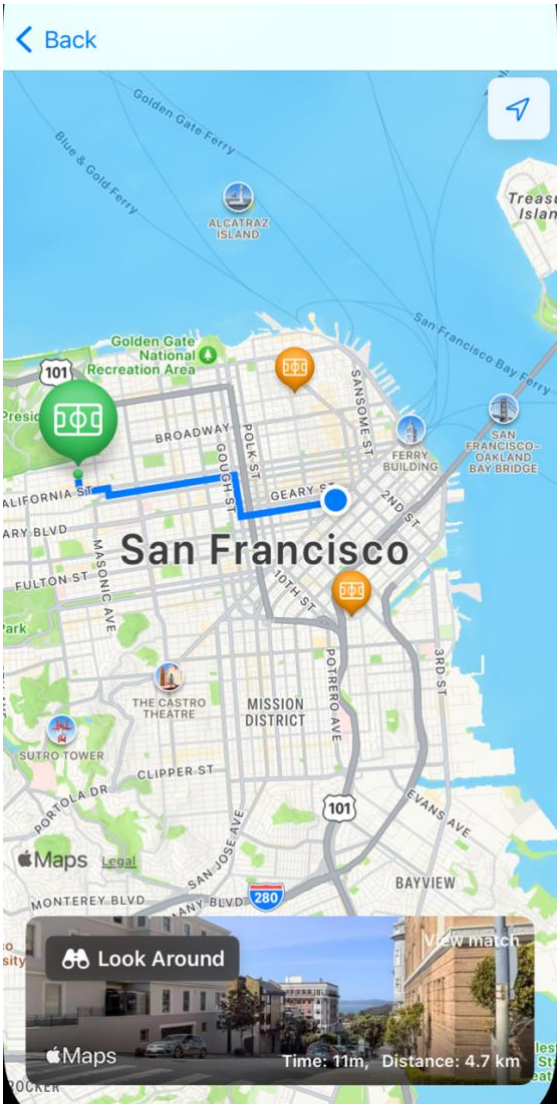
Sl. 5.9. Zaslon za pregled detalja događaja

5.5.5. Zaslon za pregled dostupnih događaja na mapi

Otvaranjem mape na početnom zaslonu otvara se zaslon za pregled dostupnih događaja na mapi, prikazan na slici 5.10. Mapa je centrirana na korisnikovu trenutnu lokaciju te su označeni svi dostupni događaji. Klikom na pojedinu oznaku događaja otvaraju se detalji događaja i iscrtavaju se upute dolaska od korisnika do odabrane lokacije, vidljivo na slici 5.11, klikom na prikaz koji se otvorio korisnik se može kretati u prostoru s početkom u označenoj odabranoj lokaciji. Kako bi korisnik mogao koristiti mapu, pri prvom otvaranju aplikacije traži se dozvola za korištenje korisnikove lokacije.



Sl. 5.10. Zaslon za pregled dostupnih događaja na mapi



Sl. 5.11. Zaslon detalja odabranog događaja na mapi

6. Zaključak

U ovom završnom radu razvijena je mobilna aplikacija za rekreativne sportove korištenjem Xcodea i programskog jezika Swift.

Korisnicima je omogućena registracija i prijava, stvaranje događaja s odabranim sportom, datumom, satnicom, brojem igrača i brojem igrača koji nedostaju ili pridruživanje događajima koje su stvorili drugi korisnici. Izgradnja aplikacije znatno je olakšana koristeći Firebase Firestore bazu podataka.

Opisana je registracija i prijava korisnika kroz Firebase Authentication te osnovne funkcije za rad aplikacije s podacima. Izgled i funkcionalnost svakog zaslona predstavljen je kroz poglavlja slikama zaslona i odgovarajućim opisima.

Aplikacija je testirana u simulatoru dostupnom unutar Xcodea i provjereno obavlja sve opisane funkcionalnosti. MapKit koji je zaslužan za najkompleksnije značajke aplikacije, pokazao se kao moćan alat za jednostavan i učinkovit rad s mapama unutar Swifta.

Ovaj završni rad pruža osnovu za daljnje razvijanje i proširenje mobilne aplikacije za okupljanje ljudi diljem svijeta na rekreativnim sportovima.

Literatura

[1] https://developer.apple.com/documentation/xcode; pristupljeno 3. lipnja 2024.

[2] https://www.swift.org/about/; pristupljeno 3. lipnja 2024.

[3] https://developer.apple.com/xcode/swiftui/; pristupljeno 3. lipnja 2024.

[4] https://www.swift.org/documentation/package-manager/; pristupljeno 4. lipnja 2024.

[5] https://www.geeksforgeeks.org/firebase-introduction/; pristupljeno 6. lipnja 2024.

[6] https://developer.apple.com/documentation/mapkit/mapkit_for_swiftui; pristupljeno 6. lipnja 2024.

Sažetak

Tema ovog završnog rada mobilna je aplikacija za timske rekreativne sportove za iOS operativni sustav. Cilj je omogućavanje korisniku da na jednostavan i brz način pronađe ljude za neki od odabranih rekreativnih sportova ili se pridruži već postojećem događaju stvorenom od strane drugog korisnika. Za potrebe pohrane podataka, manipuliranja podacima i autentifikacije korisnika korišten je Firebase. Klijentski dio aplikacije napravljen je u Xcode razvojnom okruženju koristeći programski jezik Swift uz SwiftUI radni okvir za izradu korisničkih sučelja.

Ključne riječi: Sport, iOS, Firebase, Xcode, Swift, SwiftUI

Summary

The topic of this final thesis is a mobile application for team recreational sports for the iOS operating system. Its goal is to enable the user to easily and quickly find people for one of the selected recreational sports or to join an existing event created by another user. For data storage, data manipulation, and user authentication it uses Firebase. The client side of the application was developed in the Xcode development environment using the Swift programming language along with the SwiftUI framework for building user interfaces.

Keywords: Sport, iOS, Firebase, Xcode, Swift, SwiftUI