Aplikacija za organizaciju i pregled nogometnih natjecanja

Svjetličić, Leo

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:200:130534

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

APLIKACIJA ZA ORGANIZACIJU I PREGLED NOGOMETNIH NATJECANJA

Završni rad

Leo Svjetličić

Osijek, 2023.

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

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

Osijek, 14.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Leo Svjetličić
Studij, smjer:	Programsko inženjerstvo
Mat. br. Pristupnika, godina upisa:	R4569, 28.07.2020.
OIB Pristupnika:	40505996306
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Naslov završnog rada:	Aplikacija za organizaciju i pregled nogometnih natjecanja
Znanstvena grana rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak završnog rad:	Zadatak završnog rada je napraviti aplikaciju za organizaciju i pregled malonogometnih turnira. Aplikacija omogućuje prijavu dvije vrste korisnika, organizator natjecanja i korisnik kojemu je omogućen samo pregled stvorenih natjecanja. Organizator natjecanja pri stvaranju istog može odabrati kakvu vrstu natjecanja želi: ligu, turnir s grupnom fazom i bez nje, koja će biti vidljiva korisniku spajanjem na server. Aplikacija
Prijedlog ocjene završnog rada:	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	14.09.2023.
Datum potvrde ocjene od strane Odbora:	

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

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 23.09.2023.

Ime i prezime studenta:	Leo Svjetličić
Studij:	Programsko inženjerstvo
Mat. br. studenta, godina upisa:	R4569, 28.07.2020.
Turnitin podudaranje [%]:	7

Ovom izjavom izjavljujem da je rad pod nazivom**: Aplikacija za organizaciju i pregled nogometnih natjecanja**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora ,

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

Potpis studenta:

Sadržaj

1. Uvod 6

1.1 Zadatak završnog rada 6

2. Pregled područja teme rada 7

2.1 Postojeća slična rješenja 7

3. Korištene tehnologije 10

3.1 Operacijski sustav Android 10

3.2 Android Studio 10

3.3 Kotlin 10

3.4 Jetpack Compose 10

3.5 Firebase 11

3.6 Koin 11

4. Struktura Aplikacije 12

4.1 Zaslon prijave/registracije 12

4.2 Glavni zaslon 15

4.3 Početni zaslon 17

4.4 Zaslon praćenih natjecanja 24

4.5 Zaslon stvorenih natjecanja 25

4.6 Zaslon detalja natjecanja 26

4.6.1 Zaslon detalja lige 27

4.6.2 Zaslon detalja turnira 29

4.7 Zaslon detalja timova 31

4.8 Zaslon za stvaranje natjecanja 32

5. Arhitektura aplikacije 34

5.1 Paketi 34

5.2 MVVM 34

5.2.1 Model 34

5.2.2 View 35

5.2.3 ViewModel 35
5.3 Koin 35
6. Zaključak 37
Literatura 38
Sažetak 40
Abstract 41
Životopis 42
Dodatak 43

1. Uvod

Nogomet je jedan od najpopularnijih sportova na svijetu, a organizacija i praćenje nogometnih turnira može biti izazovan zadatak. Cilj ovog završnog rada je razviti aplikaciju koja omogućuje organizaciju i pregled nogometnih turnira, kako bi se olakšao ovaj proces. Aplikacija je dizajnirana za prijavu dvije vrste korisnika - organizatora natjecanja i korisnika s ograničenim pristupom koji može samo pregledati stvorena natjecanja. Organizator natjecanja ima ovlasti za stvaranje novih natjecanja i odabir vrste natjecanja koju želi provesti. Moguće opcije uključuju ligu ili turnir. Nakon što organizator stvori natjecanje, ono postaje vidljivo korisnicima koji se povežu s aplikacijom putem poslužitelja (engl. *server*).

Jedna od ključnih funkcionalnosti aplikacije je automatsko generiranje rasporeda utakmica. Nakon što su sve ekipe dodane u natjecanje, aplikacija automatski generira raspored utakmica na temelju odabrane vrste natjecanja. Nakon odigrane utakmice, rezultati se unose u aplikaciju, a tablica ili ždrijeb se automatski ažurira kako bi se odrazilo trenutno stanje natjecanja.

Za razvoj aplikacije koristi se programski jezik *Kotlin* te za korisničko sučelje (engl. *user interface) Jetpack Compose*, koji su sve popularniji u razvoju Android aplikacija.

Kroz ovaj završni rad, detaljnije će se istražiti arhitektura i implementacija aplikacije, uz naglasak na funkcionalnosti kao što su prijava korisnika, stvaranje natjecanja, generiranje rasporeda utakmica i ažuriranje rezultata..

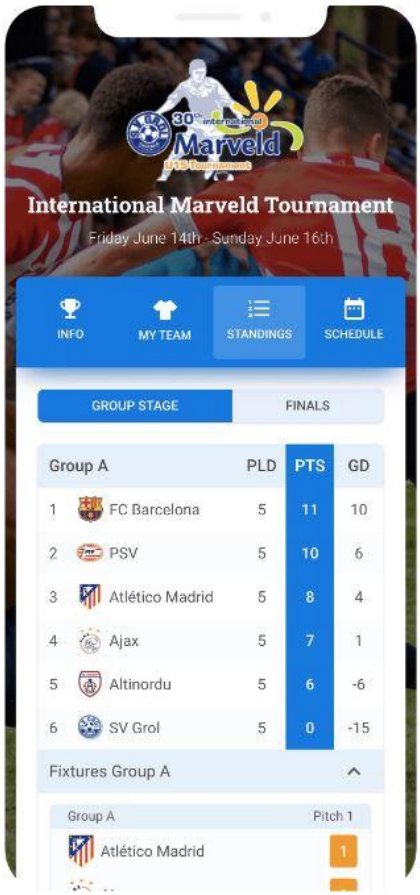
1.1 Zadatak završnog rada

Zadatak završnog rada bio je napraviti aplikaciju za organizaciju i pregled nogometnih turnira. Aplikacija omogućuje prijavu dvije vrste korisnika, organizator natjecanja i korisnik kojemu je omogućen samo pregled stvorenih natjecanja. Organizator natjecanja pri stvaranju istog može odabrati kakvu vrstu natjecanja želi, ligu ili turnir, koja će biti vidljiva korisniku spajanjem na server. Aplikacija automatski stvara raspored utakmica nakon dodavanja svih ekipa i poslije svake odigrane utakmice ažurira se tablica ili ždrijeb. Za stvaranje aplikacije koristi se *Kotlin*.

2. Pregled područja teme rada

2.1 Postojeća slična rješenja

Tournify: Tournify (Slika 2.1.) je aplikacija koja olakšava organiziranje sportskih turnira. [1] Omogućuje izradu turnirskih grafikona, upravljanje prijavama i rezultatima te praćenje napretka turnira. Osim toga nudi prilagodbu postavki turnira, uključujući vrstu sporta, broj ekipa, format natjecanja i raspored utakmica. Tournify također nudi alate za komunikaciju s ekipama i objavljivanje rezultata uživo.



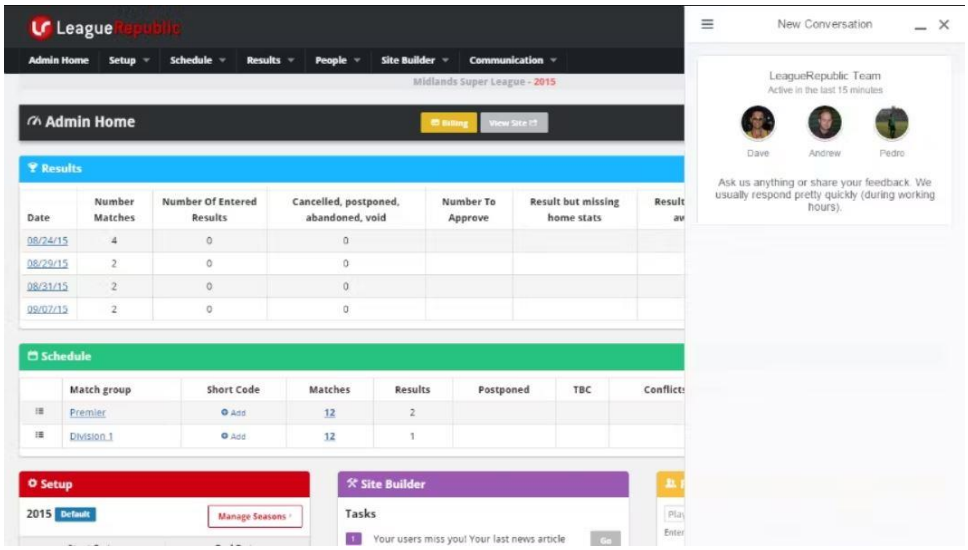
Slika 2.1. Zaslon Tournify aplikacije

Challonge: Challonge (Slika 2.2.) omogućuje brzo stvaranje turnira, postavljanje pravila i rasporeda te upravljanje prijavama i rezultatima. [2] Challonge također ima funkcionalnost za praćenje statistika, rangiranje igrača i generiranje izvještaja o turnirima. Korisnici mogu pratiti napredak turnira putem interneta i sudjelovati u natjecanjima s drugim igračima diljem svijeta.

Slika 2.2. Zaslon Challonge aplikacije

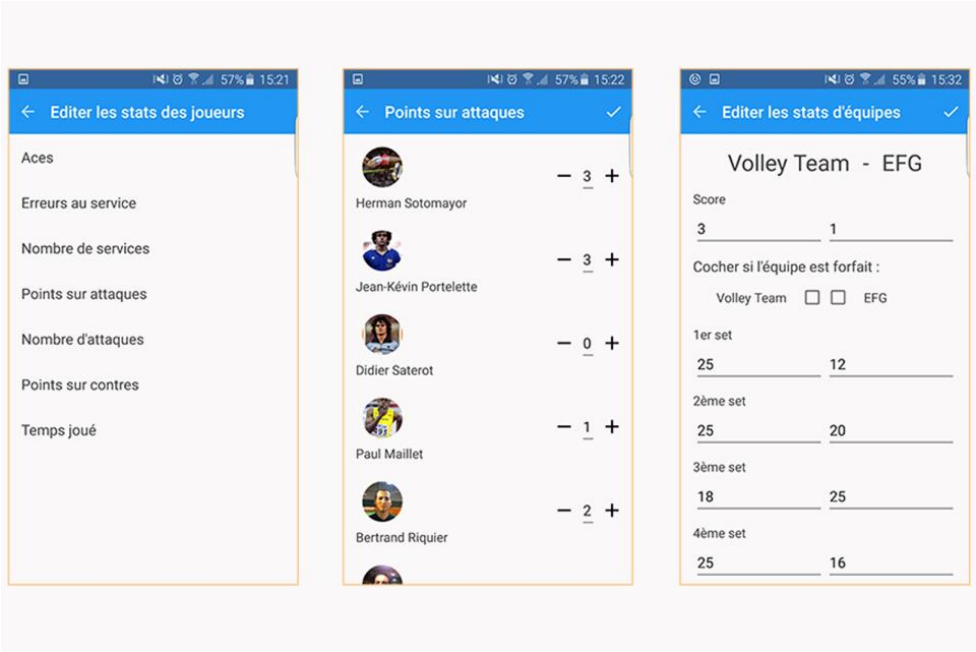
Zuluru: Zuluru je aplikacija za upravljanje sportskim ligama i turnirima. [3] Omogućuje organizaciju i upravljanje rasporedima utakmica, prijavama i registracijama ekipa, te praćenje rezultata i statistika. Zuluru također nudi alate za komunikaciju s ekipama i objavljivanje ažuriranja o turnirima. Aplikacija je posebno korisna za sportske lige i udruge koje organiziraju redovite sezone i turnire s više ekipa.

LeagueRepublic: LeagueRepublic (Slika 2.3.) je platforma koja je prvenstveno dizajnirana za upravljanje sportskim ligama i natjecanjima. [4] Ova platforma omogućuje sportskim organizacijama, poput nogometnih ili košarkaških liga, da olakšaju procese kao što su zakazivanje utakmica, praćenje rezultata i upravljanje timovima i igračima. Ovdje možete stvoriti raspored za utakmice, bilježiti rezultate svake utakmice te pratiti statistiku igrača i timova. Također pruža alate za komunikaciju s članovima timova i objavljivanje vijesti.



Slika 2.3. Zaslon LeagueRepublic aplikacije [15]

SportEasy: SportEasy (Slika 2.4.) je aplikacija za upravljanje sportskim timovima i ligama. [5] Osim općih značajki kao što su zakazivanje utakmica i praćenje rezultata, SportEasy se posebno fokusira na poboljšanje komunikacije između članova timova. To znači da omogućuje bolju koordinaciju između igrača i trenera te olakšava razmjenu informacija i obavijesti unutar tima. Osim toga, pruža alate za stvaranje i upravljanje nogometnim ligama, uključujući sustav za bilježenje i analizu statistike, što može pomoći u unaprjeđenju performansi timova.



Slika 2.4. Zaslon SportEasy aplikacije [16]

3. Korištene tehnologije

3.1 Operacijski sustav Android

Operacijski sustav Android je mobilni operativni sustav razvijen od strane Googlea. [6] On je otvorenog koda i temelji se na Linux jezgri. Android operacijski sustav namijenjen je uređajima kao što su pametni telefoni, tableti, pametni televizori, nosive tehnologije i drugi uređaji.

3.2 Android Studio

Android Studio je službeno integrirano razvojno okruženje za razvoj Android aplikacija. [7] Android Studio pruža programerima alate za pisanje, testiranje, otklanjanje pogrešaka (engl. *debugging*) i izgradnju aplikacija. Ovaj alat dolazi sa skupom funkcionalnosti kao što su grafički uređivač korisničkog sučelja, emulator za testiranje aplikacija na virtualnim uređajima, alati za analizu performansi itd.

3.3 Kotlin

Kotlin je moderni programski jezik koji je sve popularniji u razvoju Android aplikacija. [8] *Kotlin* je kompatibilan s Javom i može se koristiti zajedno s Javom u Android projektima. On pruža brojne značajke koje olakšavaju razvoj aplikacija, kao što su čiste i koncizna sintakse, potpora za nulabilnost, lambda izrazi, funkcionalno programiranje, proširenja funkcionalnosti i još mnogo toga. *Kotlin* je službeno podržan od strane Googlea kao jezik za razvoj Android aplikacija.

3.4 Jetpack Compose

Jetpack Compose je moderni alatni skup za izgradnju korisničkog sučelja (UI) u Android aplikacijama. [9] Umjesto tradicionalnog pristupa korištenjem *XML*-a i *Java/Kotlin* koda za definiranje korisničkog sučelja, *Jetpack Compose* omogućuje izgradnju korisničkog sučelja pomoću deklarativnog pristupa. To znači da se korisničko sučelje definira kao skup funkcija (koje se moraju označiti s *@Composable*) i komponenata, olakšavajući izgradnju i održavanje korisničkog sučelja. *Jetpack Compose* je dizajniran kako bi bio jednostavan, efikasan, fleksibilan i reaktivan.

3.5 Firebase

Firebase je platforma za razvoj mobilnih i web aplikacija koja je razvijena od strane Google-a. [10] Ona pruža razne usluge i alate koji olakšavaju razvoj aplikacija. *Firebase* podržava funkcionalnosti kao što su autentifikacija korisnika, baza podataka u stvarnom vremenu, upravljanje korisnicima, analitika, upuštanje poruka, testiranje aplikacija i još mnogo toga. *Firebase* je popularan izbor za pozadinski sustav usluga (engl. *backend*) u Android razvoju zbog svoje jednostavnosti integracije, skalabilnosti i široke funkcionalnosti koje pruža.

3.6 Koin

Koin je okvir za upravljanje ovisnostima (engl. *dependency injection*) koji se koristi u *Kotlin* programskom jeziku. [11] Glavna svrha *Koin*-a je olakšati upravljanje ovisnostima u *Kotlin* aplikacijama. *Koin* je koristan jer olakšava testiranje i poboljšava čitljivost koda.

4. Struktura Aplikacije

Kako bi se objasnila implementacija samog korisničkog sučelja potrebno je poznavati barem osnovne koncepte unutar *Compose* alata. Osnovna tri grafička elementa u alatu su okvir (engl. *Box*), stupac (engl. *Column*) i red (engl. *Row*). Okvir nam omogućava grupiranje drugih elemenata unutar sebe. Djeluje kao kontejner za razne vizualne elemente kao što su tekst, slike, ikone, gumbi itd. Omogućuje nam postavljanje elemenata jedan pored drugoga ili jedan ispod drugoga, ovisno o potrebama. Stupac je kontejner koji omogućuje vertikalno poravnanje elemenata. Svi elementi koji su smješteni unutar stupac komponente biti će poredani jedan ispod drugoga, počevši od vrha prema dolje. Za razliku od stupca red nam omogućuje poravnavanje elemenata horizontalno s lijeva na desno. Unutar *Compose* alata svaka funkcija može primiti parametar *modifier* koji podalje određuje pojedinosti elementa na zaslonu kao što su izgled, pozicija i slično.

4.1 Zaslon prijave/registracije

Prilikom ulaska u aplikaciju prikazuje se zaslon prijave, odnosno registracije. Stanje zaslona ovisi o vrijednosti *isRegister* (Programski kod 4.1.). To znači da zasloni dijele iste elemente, ali se ovisno o stanju prikazuju drugačije vrijednosti.

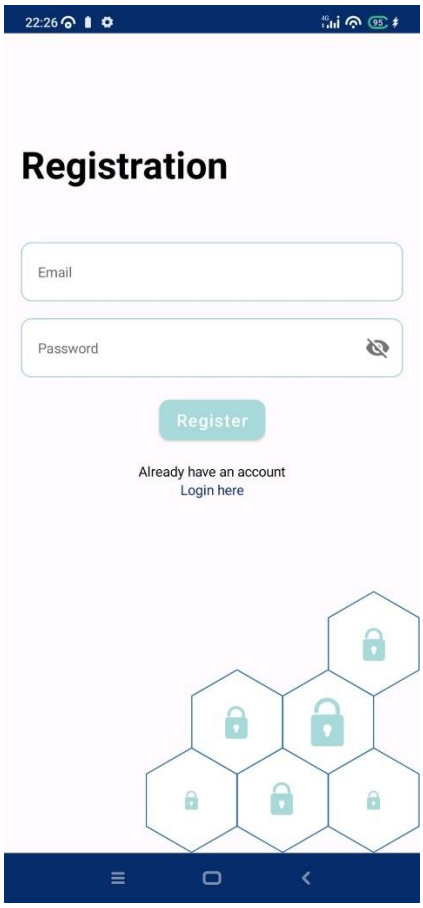
```
var isRegister by remember {
    mutableStateOf(false)
}
```

Programski kod 4.1. Varijabla *isRegister*

Označavanjem varijable *isRegister* s *mutableStateOf()* omogućuje nam rekompoziciju *Compose* elemenata koji ovise o njoj, odnosno prilikom njene promjene biti će obaviješteni svi elementi te se prikladno izmijeniti kao što je prikazano programskim kodom 4.2. Zaslon registracije (engl. *register screen*) (Slika 4.1.) omogućava nam registraciju putem e-pošte i zaporke, koristeći *Firebase* autentifikaciju (engl. *authentification*).

```
Text(
    fontWeight = FontWeight.Bold,
    modifier = Modifier
        .padding(bottom = 50.dp, start = 16.dp)
        .align(Alignment.Start),
    text = if (isRegister) {
        "Registration"
    } else {
        "Login"
    },
    fontSize = 44.sp,
)
```

Programski kod 4.2. Promjena vrijednosti elementa ovisno o varijabli *isRegister*



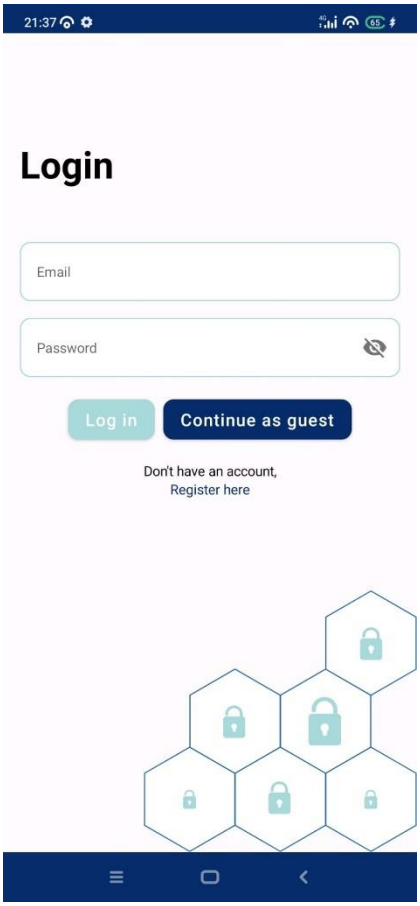
Slika 4.1. Zaslon registracije

Pritiskom na gumb registracije (engl. *register*), poziva se istoimena funkcija (Programski kod 4.3.) koja u slučaju da su polja e-pošte i zaporke puni, stvara novog korisnika s danim parametrima. Ukoliko već postoji korisnik s unesenom e-poštom, registracija se zaustavlja i ispisuje na zaslon pomoću obavijest (engl. *toast*) poruke.

```
fun register(email: String, password: String, context: Context) {
    if (!(email.isEmpty() || password.isEmpty())) {
        auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val user = auth.currentUser
                User.name = user?.email.toString()
                User.isLoggedIn.postValue(true)
            } else {
                Toast.makeText(context, "Registration failed", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

Programski kod 4.3. Funkcija za registraciju korisnika

Zaslon za prijavu (Slika 4.2.) omogućava nam nastavak u aplikaciju kao gost, kojom se ograničavaju mogućnosti u aplikaciji, ili se može prijaviti kao korisnik e-poštom i zaporkom. Nastavak u aplikaciju kao gost realiziran je tako što se mijenja vrijednost *isGuest* unutar objekta korisnik (engl. *user*) (Programski kod 4.4.).



Slika 4.2. Zaslon prijave

```
object User {
    var name = ""
    var isLoggedIn = MutableLiveData(false)
    var isGuest = MutableLiveData(false)
}
```

Programski kod 4.4. Objekt korisnik

S druge strane prijava je realizirana pozivom funkcije prijava (engl. *login*) (Programski kod 4.5.) koja je isto kao i registracija odrađena pomoću *Firebase* autentifikacije. Zaslon prijave, odnosno registracije je prikazan ukoliko su *isLoggedIn* i *isGuest* vrijednosti netočne (engl. *false*).

```
fun login(email: String, password: String, context: Context) {
    if (!(email.isEmpty() || password.isEmpty())) {
        auth.signInWithEmailAndPassword(email, password).addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val user = auth.currentUser
                User.name = user!!.email.toString()
                User.isLoggedIn.postValue(true)
            } else {
                Toast.makeText(context, "Login failed", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

Programski kod 4.5. Funkcija za prijavu korisnika

4.2 Glavni zaslon

Nakon korisnikove prijave, prikazuje mu se glavni zaslon, koji je podijeljen u tri glavna dijela. Gornja traka, dio sa sadržajem te donja navigacijska traka. Gornja traka sastoji se od naziva aplikacije koji ujedno služi i kao gumb za povratak na početni zaslon, te od gumba za odjavu. Broj elemenata u donjoj navigacijskoj traci ovisi tome je li korisnik prijavljen kao gost ili ne. Ukoliko jest tada se sastoji od početnog zaslona i zaslona praćenih natjecanja. U suprotnome, u navigacijskoj traci dodaje se još i zaslon stvorenih natjecanja. Promjena ikona u navigacijskoj traci odrađena je stvaranjem navigacijskog elementa (engl. *navigation item*) klase s pripadajućim objektima (Programski kod 4.6.) te pri stvaranju same trake uspoređuje se trenutna ruta s rutama svakog od objekta.

```
sealed class NavigationItem(
    override val route: String,
    val selectedIconId: Int,
    val unselectedIconId: Int,
    val labelId: Int
) : FutAppDestination(route) {
    object HomeDestination : NavigationItem(
        route = HOME_ROUTE,
        selectedIconId = R.drawable.home_full,
        unselectedIconId = R.drawable.home_empty,
        labelId = R.string.home
    )

    object FollowedDestination : NavigationItem(
        route = FOLLOWED_ROUTE,
        selectedIconId = R.drawable.follow_button_filled,
        unselectedIconId = R.drawable.follow_button_empty,
        labelId = R.string.followed
    )

    object MyCompetitions : NavigationItem(
        route = MY_COMPETITIONS,
        selectedIconId = R.drawable.ic_filled_my,
        unselectedIconId = R.drawable.ic_empty_my,
        labelId = R.string.my
    )
}
```

Programski kod 4.6. Klasa navigacijskog elementa

Dio sa sadržajem predstavljen je komponentom navigacijski voditelj (engl. *NavHost*). Unutar komponente potrebno je definirati rute kao što je prikazano programskim kodom 4.7. za svaki zaslon koji će se prikazivati u komponenti.

```
composable(
    route = CompetitionDetailsDestination.route,
    arguments = listOf(navArgument(COMPETITION_ID) { type = NavType.StringType },
        navArgument(IS_MY_COMPETITION_PREVIOUS) { type = NavType.BoolType }
    )
) {
    val competitionId = it.arguments?.getString(COMPETITION_ID)
    val previousScreenIsMyCompetition =
        it.arguments?.getBoolean(IS_MY_COMPETITION_PREVIOUS)
    val viewModel =
        getViewModel<CompetitionDetailsViewModel>(parameters = {
            parametersOf(competitionId)
        })
    CompetitionDetailsRoute(
        viewModel = viewModel,
        isEditing = previousScreenIsMyCompetition!!,
        onTeamCardClick = { index ->
            navController.navigate(
                TeamDetailsDestination.createNavigationRoute(index)
            )
        },
    )
}
```

Programski kod 4.7. Definiranje rute zaslona detalja natjecanja

Pronalazak rute unutar komponente određuje se parametrom ruta (engl. *route*), koja je predstavljena nizom znakova. Ukoliko je moguće da isti zaslon bude drugačiji ovisno o tome koji su podaci prikazani, recimo dva različita natjecanja neće imati iste detalje, moguće je proslijediti podatke kroz rutu koristeći parametar argumenti (engl. *arguments*). Tada se ruta kreira na sljedeći način:

- Dodaje se korijenski niz znakova (engl. *string*) ruta detalja natjecanja (engl. *COMPETITION_DETAILS_ROUTE*) koji dijele svi detalji zaslona
- Tada se na njega dodaju nizovi znakova kao što je identifikator natjecanja (engl. *COMPETITION_ID)*, koji tu ruto čine jedinstvenom.

```
const val COMPETITION_DETAILS_ROUTE = "CompetitionDetails"
const val COMPETITION_ID = "competitionId"
const val IS_MY_COMPETITION_PREVIOUS = "previousScreen"
const val COMPETITION_DETAILS_ROUTE_WITH_PARAMS = "$COMPETITION_DETAILS_ROUTE/{$COMPETITION_ID}/{$IS_MY_COMPETITION_PREVIOUS}"

object CompetitionDetailsDestination : FutAppDestination(COMPETITION_DETAILS_ROUTE_WITH_PARAMS) {
    fun createNavigationRoute(
        competitionId: String,
        fromMyCompetitionScreen: Boolean = false
    ): String =
        "$COMPETITION_DETAILS_ROUTE/$competitionId/$fromMyCompetitionScreen"
}
```

Programski kod 4.8. Funkcija stvaranja rute zaslona detalja natjecanja

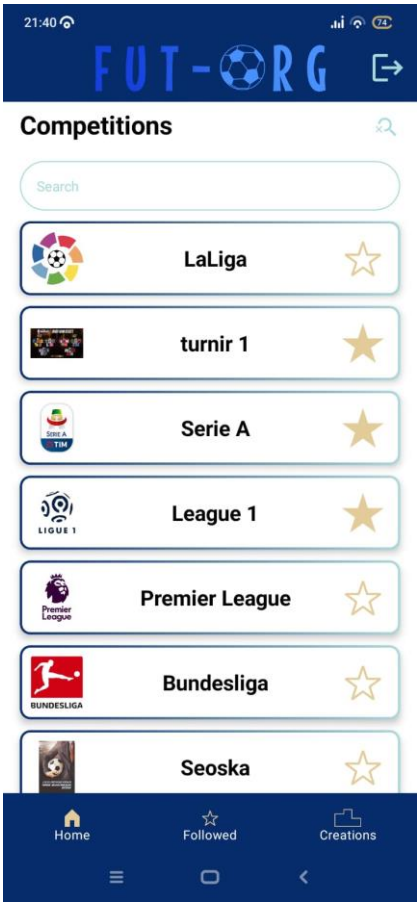
Poziv funkcije na programskom kodu 4.8. prikazan je programskim kodom 4.9. gdje se predaje identifikator (engl. *id*) natjecanja kako bi se kasnije iz rute taj podatak mogao iskoristiti.

```
onCompetitionCardClick = { id, isLeague ->
    if (isLeague) {
        navController.navigate(
            CompetitionDetailsDestination.createNavigationRoute(id)
        )
    } else {
        navController.navigate(
            TournamentDetailsDestination.createNavigationRoute(id)
        )
    }
},
```

Programski kod 4.9. Poziv funkcije stvaranja rute

4.3 Početni zaslon

Prva destinacija glavnog zaslona je početni zaslon (Slika 4.3.). Sastoji se od komponenata lijeni stupac (engl. *LazyColumn*), u kojoj se nalaze elementi tipa *HomeScreenViewState* (Programski kod 4.10.), tekst (engl. *Text*) koji ima vrijednost *Natjecanja* (engl. *Competitions*), te gumba za pretraživanje (engl. *Search*) koji prikazuje ili skriva traku za pretraživanje (engl. *SearchBar*).



Slika 4.3. Početni zaslon aplikacije

```
data class HomeScreenViewState(
    val id: String,
    val isLeague: Boolean,
    val competitionCardViewState: CompetitionCardViewState
)
```

Programski kod 4.10. Klasa *HomeScreenViewState*

Unutar klase *HomeScreenViewState* nalazi se vrijednost *competitionCardViewState* koja čahuri (engl. *encapsulates*) vrijednosti prikazane na pojedinoj kartici unutar početnog zaslona (Programski kod 4.11.).

```
data class CompetitionCardViewState(
    val imageUrl: String,
    val isFollowed: Boolean,
    val name: String
) {
    fun doesMatchSearchQuery(text: String): Boolean = name.contains(text, ignoreCase = true)
    fun startsWithSearchQuery(text: String): Boolean = name.startsWith(text, ignoreCase = true)
}
```

Programski kod 4.11. Klasa *CompetitionCardViewState*

Kartica natjecanja prikazuje sliku natjecanja, ime te reaktivnu komponentu gumb za praćenje (engl. *FollowButton*). Pritiskom na njega poziva se funkcija *toggleFollowed* (Programski kod 4.12.) koja dodaje ili uklanja podatke natjecanja iz lokalne baze podataka *Room* [12] ovisno o tome nalazile li se isti podaci u bazi ili ne. Također se ovisno o tome ažurira izgled gumba.

```
override suspend fun toggleFollowed(competitionId: String, isDelete: Boolean) {
    runBlocking(bgDispatcher) {
        val followedCompetitions = followed.first()
        if (followedCompetitions.any { it.id == competitionId }) {
            removeCompetitionFromFollowed(competitionId)
        } else {
            if (!isDelete)
                addCompetitionToFollowed(competitionId)
        }
    }
}
```

Programski kod 4.12. Funkcija *toggleFollowed*

Za realizaciju lokalne baze podataka potrebna su tri ključna dijela:

- Baza podataka (Programski kod 4.13.) – predstavlja popis entiteta koji se nalaze u bazi

```
@Database(
    entities = [DbFollowedCompetition::class],
    version = 1,
    exportSchema = false
)
```

Programski kod 4.13. Deklaracija baze podataka

- Entitet (Programski kod 4.14.) – data klasa s popisom atributa koje svaki entitet ima

```
@Entity(tableName = "followedCompetitions")
data class DbFollowedCompetition(
    @PrimaryKey val id: String,
    val imageUrl: String,
    val isLeague: Boolean,
    val name: String,
)
```

Programski kod 4.14. Klasa *DbFollowedCompetition*

- Objekt za pristup podacima (Programski kod 4.15.) – sučelje koje sadrži popis funkcija za pristupanje podacima *SQLite* baze podataka

```
@Dao
interface FollowedCompetitionDao {
    @Query("SELECT * FROM followedCompetitions")
    fun followed(): Flow<List<DbFollowedCompetition>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertCompetition(movie: DbFollowedCompetition)

    @Query("DELETE FROM followedCompetitions WHERE id=:id")
    fun deleteCompetition(id: String)}
```

Programski kod 4.15. Objekt za pristup podacima baze podataka

Kako bi se prikazala sva natjecanja na početnom zaslonu, dobivaju se pozivanjem funkcije *fetchCompetitions* (Programski kod 4.16.) koja vraća tok (engl. *flow*) liste natjecanja iz baze. Tok je vrsta podatka u Kotlinu koji nam omogućava automatsko obavještenje promjene podataka. Odnosno svaki puta kada se promijeni neka od vrijednosti unutar toka automatski se obavijeste svi promatrači te vrijednosti. U ovom slučaju to omogućuje da pri kreiranju ili brisanju pojedinog natjecanja, lista na zaslonu se automatski ažurira.

```
override fun fetchCompetitions(): Flow<List<DBCompetition>> {
    return callbackFlow {
        val listenerRegistration = fireStore.collection(FIRESTORE_COLLECTION_COMPETITIONS)
            .addSnapshotListener { snapshot, error ->
                if (error != null) {
                    close(error)
                    return@addSnapshotListener
                }
                if (snapshot != null) {
                    val competitions = snapshot.documents.map { competition ->
                        DBCompetition(
                            id = competition.id,
                            name = competition.getString("name") ?: "",
                            imageUrl = competition.getString("imageUrl") ?: "",
                            isLeague = competition.getBoolean("leagueIs") ?: false,
                            maker = competition.getString("maker") ?: ""
                        )
                    }
                    Constants.competitionNames = competitions.map { it.name }.toMutableList()
                    trySend(competitions).isSuccess
                }
            }
        awaitClose { listenerRegistration.remove() }
    }
```

Programski kod 4.16. Funkcija *fetchCompetitions*

Nakon dohvaćanja svih dokumenata iz baze podataka i njihovog pretvaranja u objekte tipa *DBCompetition* (Programski kod 4.17.) potrebno ih je preslikati ovisno o tome nalaze li se u lokalnoj bazi podataka za praćena natjecanja ili ne.

```
data class DBCompetition(
    val id: String,
    val name: String,
    val imageUrl: String,
    val isLeague: Boolean,
    val maker: String?
) {
```

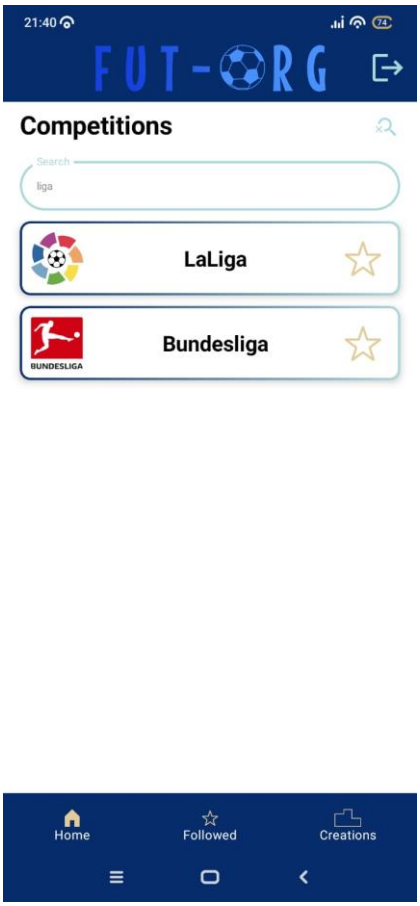
Programski kod 4.17. Klasa *DbCompetition*

Kao što je vidljivo na programskom kodu 4.18. sljedeći korak je dohvaćanje onih elemenata koji su u lokalnoj bazi podataka te njihovo preslikavanje u listu identifikatora. Zatim se prolazi kroz listu svih natjecanja i provjerava se nalazi se njihov identifikator u listi praćenih natjecanja te im se vrijednost *isFollowed* mijenja sukladno tome.

```
val competitions: Flow<List<Competition>> = competitionService.fetchCompetitions()
    .flatMapLatest { competitions ->
        competitionDao.followed().map { followed ->
            competitions.map { competition ->
                competition.toCompetition(isFollowed = followed.any { it.id == competition.id })
            }
        }
    }.shareIn(
        scope = CoroutineScope(bgDispatcher),
        started = SharingStarted.Eagerly,
        replay = 1
    )
```

Programski kod 4.18. Varijabla svih natjecanja

Funkcija za dohvaćanje praćenih natjecanja također vraća vrijednost tok što znači da će svaka promjena u bazi automatski biti prikazana na zaslonu. Pretraživanje (Slika 4.4.) natjecanja po imenu (Programski kod 4.19.) realizirano je na način da ukoliko unos trake za pretraživanje nije prazan, natjecanja se prvo filtriraju po tome započinju li s tim unosom, a zatim sadrže li taj unos. Pritiskom na karticu natjecanja prelazi se na zaslon detalja natjecanja.



Slika 4.4. Rezultati pretraživanja riječi „liga“

```
val homeScreenViewState = searchText.debounce(200)
    .combine(allCompetitions) { text, competitions ->
        if (text.isBlank()) {
            competitions
        } else {
            val filteredByStart = competitions.competitionViewStates.filter {
                it.competitionCardViewState.startsWithSearchQuery(text)
            }
            val filteredByContain = competitions.competitionViewStates.filter {
                it.competitionCardViewState.doesMatchSearchQuery(text) && it !in filteredByStart
            }
            HomeScreenListViewState(filteredByStart + filteredByContain)
        }
    }.stateIn(
        scope = viewModelScope,
        started = SharingStarted.Eagerly,
        initialValue = allCompetitions.value
    )
```

Programski kod 4.19. Varijabla pretraženih natjecanja

4.4 Zaslon praćenih natjecanja

Zaslon praćenih natjecanja (Slika 4.5.) sastoji se od elementa tekst koji ima vrijednost *Praćena natjecanja* (engl. *Followed Competitons*) te lijenog stupca, a realiziran je na način da se entiteti koji se nalaze u lokalnoj bazi podataka *Room* [12] preslikaju u listu *FollowedCompetitionViewState* (Programski kod 4.20.), odnosno *FollowedViewState* te se prikažu na zaslonu.

Slika 4.5. Zaslon praćenih natjecanja

```
data class FollowedCompetitionViewState(
    val id: String,
    val isLeague:Boolean,
    val competitionViewState: CompetitionCardViewState
)

data class FollowedViewState(
    val competitionCardViewStates: List<FollowedCompetitionViewState> = emptyList()
)
```

Programski kod 4.20. Klase FollowedCompetitionViewState i FollowedViewState

Pritiskom na gumb za praćenje natjecanje će se uvijek ukliniti sa zaslona kao i iz lokalne baze podataka. Kao i na početnom zaslonu, pritiskom na karticu otvara se zaslon za detalje natjecanja.

4.5 Zaslon stvorenih natjecanja

Ukoliko je korisnik ulogiran, odnosno nije nastavio kao gost, u donjoj navigacijskoj traci se dodaje ruta za zaslon stvorenih natjecanja (Slika 4.6.). Ovaj zaslon nam prikazuje sva natjecanja koja je prijavljeni korisnik stvorio te mu omogućuje njihovo brisanje i ažuriranje, kao i stvaranje novih natjecanja.

Slika 4.6. Zaslon stvorenih natjecanja

Pritiskom na jednu od kartica otvara se zaslon detalja natjecanja, no za razliku od početnog i zaslona praćenih natjecanja, korisnik će moći ažurirati natjecanje. Pritiskom na ikonu za brisanje poziva se funkcija *onDeleteClick* (Programski kod 4.21.) koja kao parametar prima identifikator natjecanja.

```
fun onDeleteClick(id: String) {
    viewModelScope.launch {
        val diff = async {
            val competition = competitionRepository.competitionDetails(id).first()
            competition.teams.forEach { team ->
                val diff = async {
                    val teamDetails = teamRepository.getTeamDetails(team.id).first()
                    teamDetails.members.forEach { player ->
                        competitionRepository.deletePlayer(player.id)
                    }
                }
                diff.await()
                competitionRepository.deleteTeam(team.id)
            }
            competition.games.forEach { game ->
                competitionRepository.deleteGame(game.id)
            }
        }
        diff.await()
        competitionRepository.deleteCompetition(id)
        competitionRepository.toggleFollowed(id,true)
    }
}
```

Programski kod 4.21. Funkcija onDeleteClick

Funkcija služi za brisanje svih igrača, ekipa i utakmica koje to natjecanje sadrži. Svaka ekipa/utakmica u natjecanju sadrži atribut *competitionId* koji služi kao sekundarni ključ. Isto tako svaki igrač u bazi podataka sadrži atribut *teamId* s istom funkcionalnošću. Za stvaranje novih natjecanja potrebno je pritisnuti ikonu plusa u gornjem desnom kutu zaslona, kojim se otvara zaslon za stvaranje natjecanja.

4.6 Zaslon detalja natjecanja

Klasa Natjecanje (engl. *Competition*) (Programski kod 4.22.) sadrži atribut *isLeague*, pomoću kojeg se prilikom prelaska na zaslon detalja natjecanja odlučuje hoće li se prikazati zaslon detalja lige ili zaslon detalja turnira.

```
data class Competition(
    var id: String,
    val name: String,
    val isLeague: Boolean,
    val imageUrl: String,
    var isFollowed: Boolean,
    val maker: String = ""
)
```

Programski kod 4.22. Klasa natjecanja

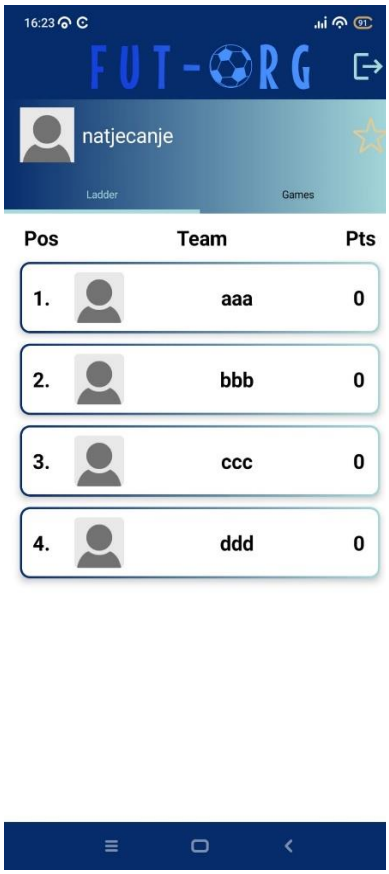
Za oba zaslona prilikom prikazivanja koristi se ista klasa Detalji natjecanja (engl. *CompetitionDetails*) (Programski kod 4.23.), koja se puni drugačijim vrijednostima ovisno o zaslonu.

```
data class CompetitionDetails(
    val competition: Competition,
    val games: List<Game> = emptyList(),
    val teams: List<ITeam>,
) {
```

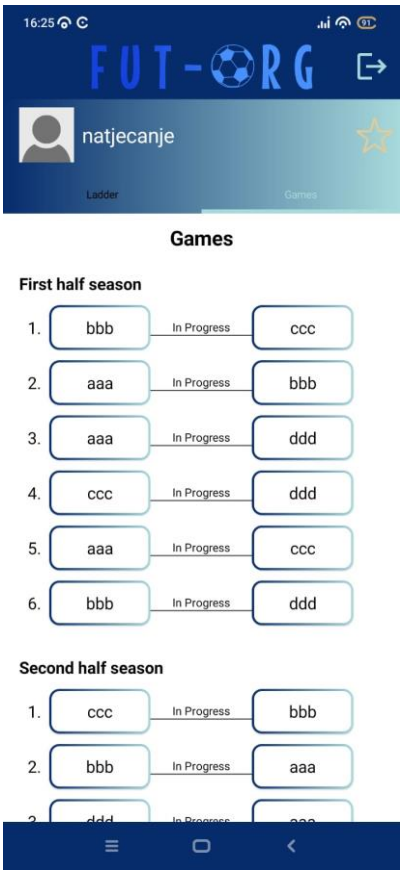
Programski kod 4.23. Klasa detalja natjecanja

4.6.1 Zaslon detalja lige

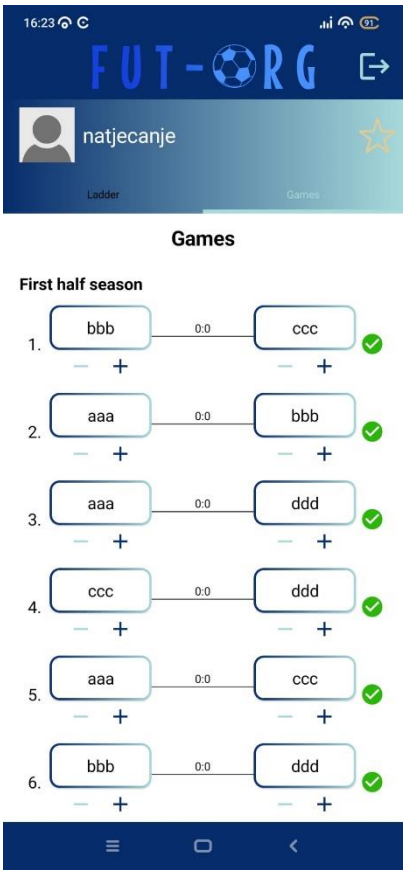
Zaslon detalja lige sastoji se od istih vrijednosti koji su sadržani i na kartici natjecanja, navigacijske trake te sadržaja. Navigacijskom trakom se odabire prikaz tablice natjecanja (Slika 4.7.) ili rasporeda utakmica (Slika 4.8.) u prostoru sadržaja.



Slika 4.7. Tablica natjecanja



Slika 4.8. Raspored Utakmica



Slika 4.9. Ažuriranje utakmica

Način prikaza utakmica o tome je li korisnik ušao na zaslon sa zaslona stvorenih natjecanja (korisnik je stvorio natjecanje) ili s nekog drugog zaslona. Ukoliko je prvi slučaj istinit korisnik će moći ažurirati utakmice (Slika 4.9.) odabirom rezultata. Kada se pritisne gumb kvačice, poziva se funkcija *onCheckClick* (Programski kod 4.24.) koja sprema promjene u hash mapu te ih zatim ažurira u bazi podataka. Također se broj bodova timova ažurira sukladno tome. Prilikom pritiska na karticu tima prikazuje se zaslon detalja tima.

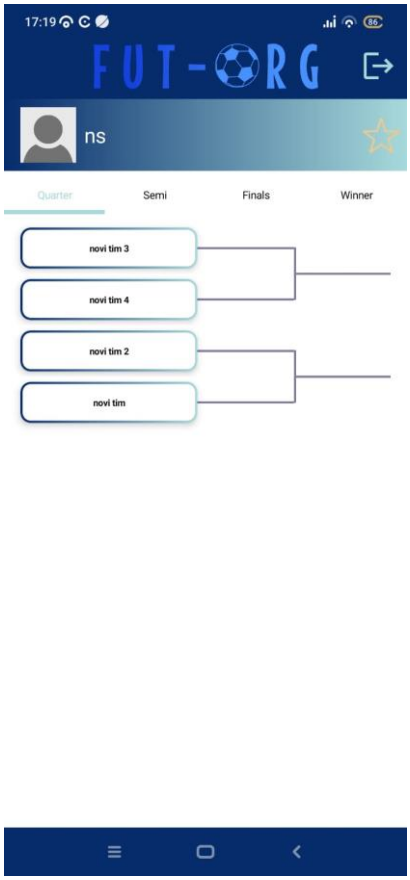
```
fun onCheckClick(
    gameId: String,
    homeTeamId: String,
    awayTeamId: String,
    homeGoals: Int,
    awayGoals: Int
) {
    viewModelScope.launch {
        val changes = hashMapOf<String, Any>()
        val homeChanges = hashMapOf<String, Any>()
        val awayChanges = hashMapOf<String, Any>()

        changes["over"] = true
        changes["homeGoals"] = homeGoals
        changes["awayGoals"] = awayGoals
        val diff = async {
            val homeTeam = findTeam(homeTeamId)
            val awayTeam = findTeam(awayTeamId)

            if (homeGoals > awayGoals) {
                homeChanges["numberOfPoints"] = homeTeam.points + 3
                competitionRepository.updateTeam(homeTeamId, homeChanges)
            } else if (awayGoals > homeGoals) {
                awayChanges["numberOfPoints"] = awayTeam.points + 3
                competitionRepository.updateTeam(awayTeamId, awayChanges)
            } else {
                homeChanges["numberOfPoints"] = homeTeam.points + 1
                awayChanges["numberOfPoints"] = awayTeam.points + 1
                competitionRepository.updateTeam(homeTeamId, homeChanges)
                competitionRepository.updateTeam(awayTeamId, awayChanges)
            }

            competitionRepository.updateGame(gameId, changes)
        }
        diff.await()
        competitionRepository.competitionDetails(id).collect { details ->
            _detailsViewState.value =
                competitionDetailsMapper.toCompetitionDetailsViewState(details)
        }
    }
}
```

Programski kod 4.24. Funkcija *onCheckClick*

4.6.2 Zaslon detalja turnira

Zaslon detalja turnira sastoji se od istih elemenata kao i zaslon detalja lige, no u navigacijskoj traci se nalaze faze ždrijeba, a u sadržaju se nalaze timovi koji su došli do određene faze. Kao i kod zaslona detalja lige, ovisno o tome koji je prijašnji zaslon, korisniku je omogućeno ažuriranje natjecanja (Slika 4.10.) ili nije (Slika 4.11.).

Slika 4.10. Ažuriranje turnira



Slika 4.11. Zaslon detalja turnira

Prilikom ažuriranja turnira, odnosno odabira koji tim od para prolazi u sljedeću fazu, pritišće se ikona kvačice u ravnini tog tima, te se poslije toga stvara završna kvačica na sredini para timova. Pritiskom na taj gumb poziva se funkcija *onFinishedCheckClick* (Programski kod 4.25.) koja sprema promjene i ažurira ih u bazi podataka. Tim na zaslonu predstavljen je klasom turnirski tim (engl. *TournamentTeam*) (Programski kod 4.26.) u bazi podataka. Uspjeh tima prikazan je atributom *success*, koji je zapravo niz znakova te predstavlja uspjeh pojedinog tima u turniru. Vrijednosti koje se mijenjaju pri ažuriranju su *success* te *isOver*. Pobjedničkom timu se produljuje niz znakova za jedan isti znak, a gubitničkom timu se mijenja vrijednost *isOver* u istinu.

```
fun onFinishedCheckClick(winnerId: String, looserId: String, isFinal: Boolean) {
    val winnerChanges = hashMapOf<String, Any>()
    val looserChanges = hashMapOf<String, Any>()

    val tempTeamSuccess = findTeam(winnerId).success
    winnerChanges["success"] = tempTeamSuccess + tempTeamSuccess.first()
    if (isFinal) {
        winnerChanges["over"] = true
    }
    looserChanges["over"] = true

    viewModelScope.launch {
        val dif = async {
            tournamentRepository.updateTeam(winnerId, winnerChanges)
            if (looserId.isNotEmpty()) {
                tournamentRepository.updateTeam(looserId, looserChanges)
            }
        }
        dif.await()
        tournamentRepository.competitionDetails(id).collect { details ->
            _tournamentDetailsViewState.value =
                tournamentDetailsMapper.toTournamentDetailsViewState(details)
        }
    }
}
```

Programski kod 4.25. Funkcija *onFinishedCheckClick*

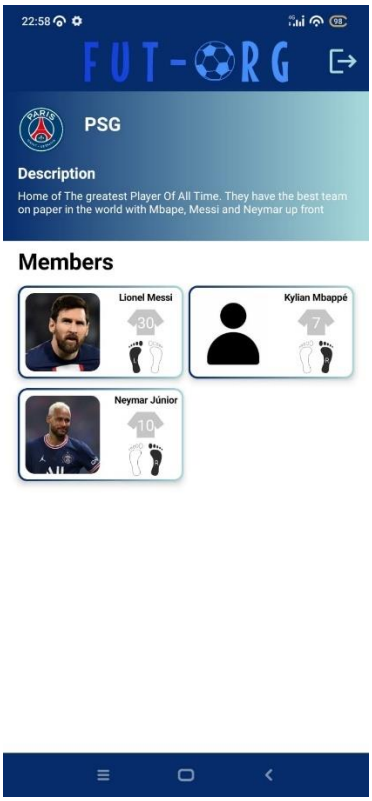
```
data class TournamentTeam(
    override var id: String = "1",
    override var imageUrl: String = "",
    override var name: String="",
    override var description: String = "",
    var isOver:Boolean = false,
    val tournament: String = "",
    val success:String
):ITeam
```

Programski kod 4.26. Klasa *TournamentTeam*

Redoslijed timova u pojedinoj fazi odrađen je pri stvaranju natjecanja pomoću nasumičnog dodjeljivanja jednog slova, pa je moguće pozvati na listi funkciju poredaj po (engl. *orderBy*). Pritiskom na karticu tima prikazuje se zaslon detalja tima.

4.7 Zaslon detalja timova

Na zaslonu detalja tima (Slika 4.12.) prikazana je slika, naziv, opis te lista igrača koji pripadaju tome timu. Igrač je sadržan u bazi podataka kao klasa član (engl. *Member*) (Programski kod 4.27.) u kojemu *teamId* predstavlja sekundarni ključ, kojim se filtriraju igrači u timove (Programski kod 4.28.).



Slika 4.12. Zaslon detalja tima

```
data class Member(
    var id: String = "1",
    val name: String = "",
    val teamId: String = "",
    val number: Int = 10,
    val rightFooted: Boolean = true,
    val imageUrl: String? = "",
)
```

Programski kod 4.27. Klasa član

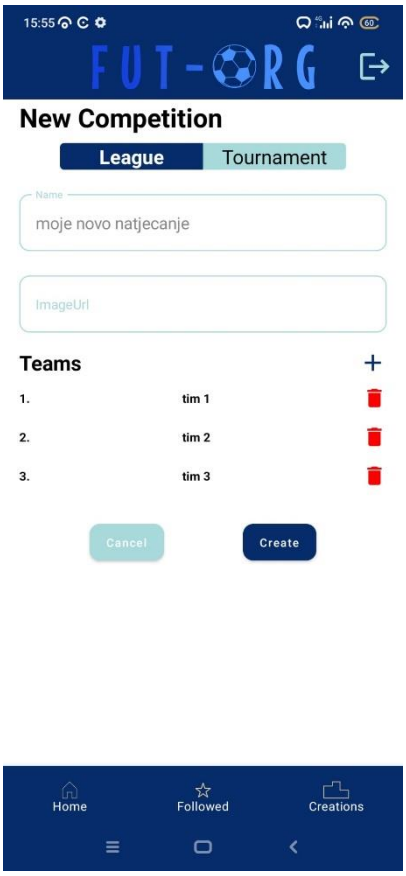
```
override suspend fun getTeamDetails(teamId: String): Flow<TeamDetails> = flow {
    val members = competitionService.fetchMembers()
    val team = findTeam(teamId)
    val teamDetails = TeamDetails(
        team = team,
        members = members.map { it.toMember() }.filter { it.teamId == team.name })
    emit(teamDetails)
}.shareIn(
    scope = CoroutineScope(bgDispatcher),
    started = SharingStarted.WhileSubscribed(1000L),
    replay = 1
)
```

Programski kod 4.28. Funkcija *getTeamDetails*

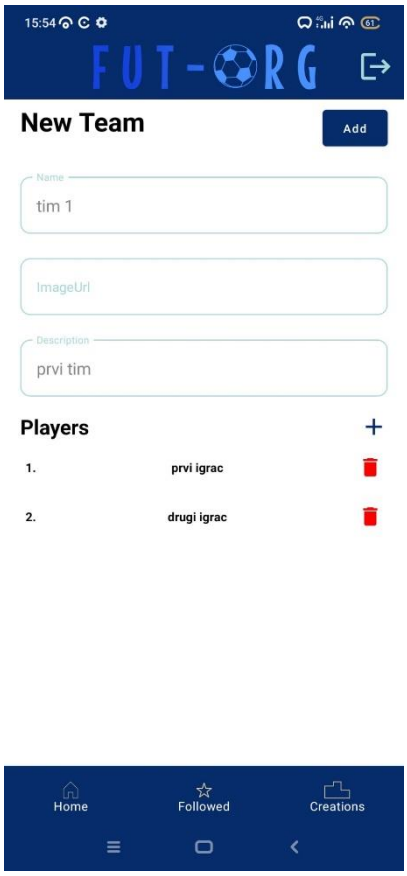
Igrači se na zaslonu prikazuju pomoću komponente Mreža s odgađanjem (engl. *LazyVerticalGrid*) koja omogućuje prikaz elemenata u predefiniran broj stupaca, koji je u ovome slučaju dva.

4.8 Zaslon za stvaranje natjecanja

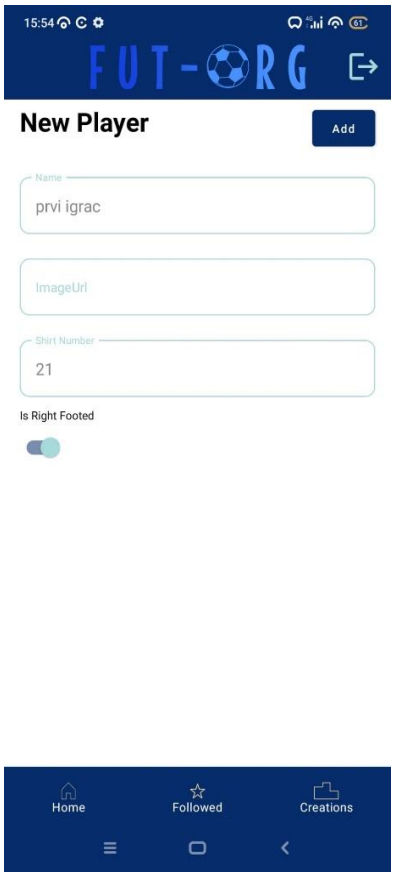
Zaslonu za stvaranje natjecanja (Slika 4.13.) mogu pristupiti samo prijavljeni korisnici sa zaslona stvorenih natjecanja. Prilikom stvaranja natjecanja korisnik odabire koju vrstu natjecanja želi, turnir ili liga, naziv te sliku natjecanja. Korisnik može ili odbaciti svoje natjecanje pritiskom na gumb za prekid (engl. *Cancel*) ili potvrditi njegovo natjecanje pritiskom na gumb za stvaranje (engl. *Create*), koja sprema natjecanje, sve timove i igrače u natjecanju u bazu podataka. Timovi se dodaju pritiskom na ikonu plusa, koja prikazuje raspored zaslona za dodavanje timova (Slika 4.14.). Korisniku je omogućeno upisivanje imena, slike, opisa te igrača unutar tima. Dodavanje igrača obavlja se pritiskom na ikonu plusa, koji otvara raspored zaslona za dodavanje igrača (Slika 4.15.). Korisnik za svakog igrača može unijeti ime, sliku, broj dresa te je li ljevak ili dešnjak. Potvrda stvaranja tima, odnosno igrača, vrši se pritiskom na gumb dodaj (engl. *Add*), ovisno o tome koji je raspored zaslona prikazan. Ukoliko korisnik želi obrisati tim, iz natjecanja, može to učiniti pritiskom na ikonu za brisanje u listi timova, isto vrijedi za brisanje igrača iz liste u timu. Prilikom stvaranja svakog od elemenata nije nužno unijeti vrijednost *imageUrl*, te se u tom slučaju sprema predefinirana (engl. *default*) vrijednost.



Slika 4.13. Dodavanje natjecanja



Slika 4.14. Dodavanje tima



Slika 4.15. Dodavanje igrača

Prilikom pritiska na gumb za stvaranje, poziva se funkcija *postCompetition* koja automatski stvara sve utakmice u ligi, kao i ždrijeb u turniru, te poziva prikladne funkcije za njihovo spremanje.

5. Arhitektura aplikacije

Glavni ciljevi arhitekture aplikacije [13] su organizacija koda, održavanje, odnosno lako proširivanje aplikacije te čitljivost i razumljivost. [14]

5.1 Paketi

Paketi aplikacije podijeljeni su u tri dijela: podatci, modeli te korisničko sučelje. Unutar podatkovnog dijela paketa nalaze se sve datoteke koje su povezane s bazom podataka ili služe za spremanje određenih vrijednosti. Tako u njemu pronalazimo datoteke kao što su *CompetitionAppDatabase* koja predstavlja *Room* lokalnu bazu podataka, repozitorij za timove (engl. *TeamRepository*) i repozitorij za natjecanje *(engl. CompetitionRepository*) u kojima se nalaze funkcije za pretvaranje elemenata dohvaćenih iz *Firebase* baze podataka, koja je predstavljena u aplikaciji klasom *CompetitionServiceImpl*. U paketu model nalaze se sve data klase koje predstavljaju stvarne objekte koji se koriste u aplikaciji, dok se u paketu korisničko*g* sučelja nalazi kompletno korisničko sučelje, odnosno navigacija i svi zasloni aplikacije s pripadajućim modelom prikaza.

5.2 MVVM

MVVM je arhitekturni obrazac koji se koristi u razvoju aplikacija kako bi se omogućila bolja organizacija koda i odvojenost logike korisničkog sučelja od logike podataka. Sastoji se od tri glavne komponente: model, prikaz, model prikaza.

5.2.1 Model

Model predstavlja sloj aplikacije koji je odgovoran za upravljanje podacima i poslovnom logikom. Ovdje se definiraju data klase, pravila validacije, operacije nad podacima i komunikacija s izvorima podataka (npr. baza podataka, web usluge). Model nema izravan kontakt s korisničkim sučeljem i ne zna kako će podaci biti prikazani.

5.2.2 View

Prikaz je komponenta koja predstavlja korisničko sučelje aplikacije. Ovdje se definira kako će podaci iz Modela biti prikazani korisniku. Prikaz se ne bavi poslovnom logikom ni podacima, već samo prikazuje podatke i šalje korisničke interakcije modelu prikaza.

5.2.3 ViewModel

Model prikaza je posrednički sloj između Modela i prikaza. Ovdje se nalazi logika koja prilagođava podatke iz modela za prikaz. Model prikaza također prima korisničke interakcije s prikaza i prenosi ih modelu za obradu. Model prikaza omogućava odvojenost logike prikaza od logike podataka i olakšava testiranje. Također često implementira oblikovne obrasce poput objekta promatranja (engl. *Observable*) za automatsko ažuriranje prikaza kada se podaci u Modelu promijene, koji su u aplikaciji realizirani pomoću tokova.

5.3 Koin

Koin [11] se u aplikaciji koristi za jednostavnije korištenje klasa i objekata koji trebaju biti jedinstveni. Pri pokretanju aplikacije potrebno je započeti *Koin* sa svim definiranim modulima (Programski kod 5.1.), jedan od kojih je *followModule* (Programski kod 5.2.).

```
startKoin {
    androidLogger(Level.INFO)
    androidContext(this@MainActivity)
    modules(
        dataModule,
        tournamentDetailsModule,
        addCompetitionModule,
        followedModule,
        loginModule,
        competitionDetailsModule,
        homeModule,
        teamDetailsModule,
        teamModule,
        competitionModule,
        databaseModule,
        myCompetitionModule,
        firebaseModule
    )
    Log.d("CompetitionApp", "App started")
}
```

Programski kod 5.1. Započinjanje *Koin*-a

```
val followedModule = module {
    viewModel {
        FollowedScreenViewModel(
            competitionRepository = get(),
            followedMapper = get()
        )
    }
    single<FollowedMapper> { FollowedMapperImpl() }
}
```

Programski kod 5.2. Stvaranje modula

Zatim je vrlo jednostavno pozivati pojedini element u kodu (Programski kod 5.3.) pomoću prikladne funkcije za dohvaćanje (engl. *get*). U programskom kodu 5.3 to je funkcija *getViewModel()*.

```
FollowedScreenRoute(
    onCompetitionCardClick = { id, isLeague ->
        if (isLeague) {
            navController.navigate(
                CompetitionDetailsDestination.createNavigationRoute(id)
            )
        } else {
            navController.navigate(
                TournamentDetailsDestination.createNavigationRoute(id)
            )
        }
    },
    followedScreenViewModel = getViewModel(),
    modifier = Modifier.padding(start = 16.dp, end = 16.dp)
)
```

Programski kod 5.3. Pozivanje *Koin* funkcije *getViewModel()*

6. Zaključak

Cilj ovog završnog rada bio je napraviti Android aplikaciju kojom se omogućuje stvaranje i pregled nogometnih natjecanja. Aplikacija je u potpunosti responzivna i funkcionalna, odnosno sama se ažurira ovisno o podacima u bazi podataka. Za stvaranje korisničkog sučelja korišten je *Jetpack Compose,* a programski jezik je *Kotlin*, čije korištenje je osiguralo visoku razinu fleksibilnosti i jednostavnosti u razvoju. *Firebase Firestore* kao *noSQL* baza podataka omogućila je pouzdano pohranjivanje i brzu dostupnost podataka korisnicima putem mreže.

Ova aplikacija nije samo alat za olakšavanje nogometnih natjecanja, već također predstavlja primjer korištenja suvremenih tehnologija za razvoj mobilnih aplikacija. Kroz ovaj projekt, uspješno je razvijena Android aplikacija koja pruža praktično i intuitivno rješenje za stvaranje i pregled nogometnih natjecanja. Aplikacija postavlja temelje za olakšavanje organizacije sportskih događanja i pruža korisnicima mogućnost brzog i učinkovitog praćenja rezultata, rasporeda te informacija o natjecanjima i timovima.

Literatura

[1] A. Menkveld, J. v. Huët, Jesse Bouma, »Tournify: Online tournament planner« [Mrežno]. Dostupno: https://www.tournifyapp.com/. [Pokušaj pristupa 8 Kolovoz 2023.].

[2] »Challonge - Tournament Brackets,« [Mrežno]. Dostupno: https://challonge.com/. [Pokušaj pristupa 7 Kolovoz 2023.].

[3] »Zuluru« [Mrežno]. Dostupno: https://www.zuluru.org/. [Pokušaj pristupa 7 Kolovoz 2023.].

[4] »LeagueRepublic« [Mrežno]. Dostupno: https://www.leaguerepublic.com/. [Pokušaj pristupa 7 Kolovoz 2023.].

[5] »SportEasy« [Mrežno]. Dostupno: https://www.sporteasy.net. [Pokušaj pristupa 7 Kolovoz 2023.].

[6] »Operacijski sustav Android,« [Mrežno]. Dostupno: https://www.android.com. [Pokušaj pristupa 7 Kolovoz 2023.].

[7] »Android Studio« [Mrežno]. Dostupno: https://developer.android.com/studio. [Pokušaj pristupa 7 Kolovoz 2023.].

[8] »Kotlin,« [Mrežno]. Dostupno: https://developer.android.com/kotlin/. [Pokušaj pristupa 7 Kolovoz 2023.].

[9] »Jetpack Compose« [Mrežno]. Dostupno: https://developer.android.com/jetpack/compose/documentation. [Pokušaj pristupa 7 Kolovoz 2023.].

[10] Google, »Firebase« [Mrežno]. Dostupno: https://firebase.google.com/. [Pokušaj pristupa 7 Kolovoz 2023.].

[11] A. Giuliani, »Koin - The pragmatic Kotlin Injection Framework,« [Mrežno]. Dostupno: https://insert-koin.io/. [Pokušaj pristupa 7 Kolovoz 2023.].

[12] »Room Documentation« [Mrežno]. Dostupno: https://developer.android.com/training/data-storage/room. [Pokušaj pristupa 20 Kolovoz 2023.].

[13] »Interviewbi,« [Mrežno]. Dostupno: https://www.interviewbit.com/blog/android-architecture/. [Pokušaj pristupa 20 Kolovoz 2023.].

[14] I. Lake, R. Meier, Professional Android, John Wiley & Sons, 2018.

[15] »Getapp« [Mrežno]. Dostupno: https://www.getapp.com.au/software/110155/leaguerepublic. [Pokušaj pristupa 7 Kolovoz 2023.].

[16] »Capterra« [Mrežno]. Dostupno: https://www.capterra.com.au/software/162745/sporteasy. [Pokušaj pristupa 7 Kolovoz 2023.].

Sažetak

Ovaj završni rad omogućava jednostavnije stvaranje vlastitih natjecanja, turnira kao i liga, koje drugi korisnici mogu pratiti. U svakom gradu ili selu se najčešće odražava nekakav oblik turnira ili lige. Stvaranje takvog natjecanja može biti komplicirano, kao i praćenje istoga. Ova aplikacija olakšava i stvaranje natjecanja i njegovo praćenje. Kako bi osoba kreirala natjecanje potrebno je imati korisnički račun koji se može napraviti unutar aplikacije. Prilikom stvaranja natjecanja stvaranju se timovi i igrači koji će sudjelovati u natjecanju. Osoba koja je stvorila natjecanje također ga može ažurirati, a ostali korisnici mogu to natjecanje pratiti. U prvom poglavlju opisan je zadatak te već postojeća rješenja. U drugom poglavlju opisane su tehnologije koje su se koristile u aplikaciji. Treće poglavlje opisuje kako aplikacija izgleda, navigacija između zaslona i kako je implementirana. U četvrtom poglavlju opisana je arhitektura aplikacije, odnosno kako je sve raspoređeno unutar aplikacije, od samih paketa tako i do klasa, odnosno obrasca MVVM. Potom je u zaključku opisan cilj te ukratko aplikacija.

Ključne riječi: Android, aplikacija, Compose, Firebase, Koin, Kotlin, kreiranje nogometnog natjecanja

Abstract

Title: Application for Organizing and Viewing Football Competitions

This final paper enables the easier creation of custom competitions, tournaments, and leagues that other users can follow. In most towns or villages, some form of tournament or league is commonly held. Creating such a competition can be complicated, as can tracking it. This application simplifies both the creation and monitoring of competitions. To create a competition, a person needs to have a user account that can be created within the application. When creating a competition, teams and players who will participate in the competition are added. The person who created the competition can also update it, and other users can follow that competition. The first chapter describes the task and existing solutions. The second chapter discusses the technologies used in the application. The third chapter describes how the application looks, screen navigation, and its implementation. The fourth chapter describes the architecture of the application, including how everything is organized within the application, from packages to classes, following the MVVM pattern. Finally, the conclusion outlines the purpose of the application briefly.

Keywords: Android, application, Compose, Koin, Kotlin, football competition creation, Firebase

Životopis

Leo Svjetličić rođen je 15. prosinca 2001. godine u Pakracu. Pohađao je Osnovnu školu braće Radića Pakrac, nakon čega se upisuje u Srednju školu Pakrac, smjer opća gimnazija koju završava 2020. godine. Iste godine upisuje preddiplomski studij Programsko inženjerstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek. U ožujku 2023. godine započinje šestomjesečnu stručnu praksu u tvrtki Barrage, gdje je proširio svoje znanje o razvoju Android aplikacija.

Leo Svjetličić

Dodatak

- Programski kod aplikacije može se pronaći na Git repozitoriju na linku: https://github.com/LeoSvjetlicic/Fut-Org