SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Seminar iz kolegija:

Objektno oblikovanje

**Aplikacija: Sporty**

Tomislav Šipušić 0036477967

Filip Mutnjaković 0036480548

Ivan Stepanić 0036486989

Zagreb, veljača, 2018.

**Sadržaj:**

[**1. Uvod** 3](#_Toc505569900)

[1.1. Design story 3](#_Toc505569901)

[**2. Zahtjevi** 4](#_Toc505569902)

[2.1. Funkcionalni zahtjevi 4](#_Toc505569903)

[2.2. Nefunkcionalni zahtjevi 5](#_Toc505569904)

[2.3. Use caseovi 5](#_Toc505569905)

[**3. Opis objektnog modela** 11](#_Toc505569906)

[**4. Opis implementacije perzistencije** 16](#_Toc505569907)

[4.1. Objektno relacijsko mapiranje 16](#_Toc505569908)

[4.2. Repozitorij i pristup podacima 18](#_Toc505569909)

[4.3. Unit of Work 19](#_Toc505569910)

[4.4. Dependency injection - Ninject 21](#_Toc505569911)

[**5. Opis izgrađene desktop aplikacije** 23](#_Toc505569912)

[5.1. MVVM obrazac 23](#_Toc505569913)

[5.2. Komunikacija s web servisom 24](#_Toc505569914)

[5.3. Korisničko sučelje 25](#_Toc505569915)

[**6. Opis izgrađene web aplikacije** 27](#_Toc505569916)

[6.1. Arhitektura web aplikacije - MVC (Model-View-Controller) 27](#_Toc505569917)

[6.2. Implementacija programskog rješenja 28](#_Toc505569918)

[6.2.1. Kontroleri 28](#_Toc505569919)

[6.2.2. Modeli 31](#_Toc505569920)

[6.2.3. Pogledi 32](#_Toc505569921)

# **1. Uvod**

Cilj seminara je dokumentacija rada projekta Sporty. Naglasak je na zahtjevima koje program treba podržati, na opisu objektnog modela domene, opisu implementacije perzistencije kao i na opisu desktop i web aplikacije. Naglasak je na tehnikama objektnog oblikovanja i razvoja programske potpore općenito. U poglavlju 1.1. prikazan je design story koji opisuje rad programa, dok su u poglavljima nakon opisani zahtjevi na softver i opisi cjelokupne programske potpore.

## 1.1. Design story

Program koji nam pomaže u pronalaženju događaja gdje se aktivno odvija neki sport. Program podržava da se korisnici prijavljuju na događaje, ukoliko se korisnik prijavi na događaj to se smatra da će sudjelovati tom događaju. Ukoliko se ne prijavi na neki događaj tada program daje informaciju gdje se pojedini sport odvija i korisnik može doći gledati. Neki od događaja koje program podržava su: mali nogomet, košarka, tenis, stolni tenis, badminton i slični sportovi gdje je potrebno dvoje ili više ljudi. Program omogućuje da korisnik kreira svoje događaje, objavi u koje vrijeme i gdje će mu se drugi korisnici pridružiti.

Program podržava i pretragu trenutno aktivnih događaja, ovisno o zadanom kriteriju pretrage. Program omogućuje i uređivanje osobnih podataka korisnika. Također korisniku je omogućen uvid u njegove prošle događaje koje je kreirao.

Dodatno, svaki korisnik ima svoju ocjenu za neki sport, a ocjene mu dodjeljuju ostali sudionici nekog događaja na kojem su bili.

# **2. Zahtjevi**

## 2.1. Funkcionalni zahtjevi

Funkcionalni zahtjevi su oni zahtjevi koji opisuju što bi program trebao raditi. Za naš sustav aktori i njihovi funkcionalni zahtjevi su sljedeći:

* Neregistrirani korisnik
* Prikaz ekrana za prijavu u sustav
* Registracija u sustav
* Registrirani korisnik
* Prijava u sustav
* Prikaz trenutno aktivnih događaja
* Pretraga događaja ovisno o odabranom filteru pretraživanja
* Kreiranje događaja
* Prikaz svih budućih događaja koje je korisnik kreirao
* Prikaz svih prošlih događaja koje je korisnik kreirao
* Prijava na odabrani događaj
* Ocjenjivanje ostalih korisnika
* Uređivanje osobnih podataka
* Odjava iz sustava
* Web aplikacija
* Omogućava razmjenu informacija registriranog korisnika i neregistriranog sa Web API sučeljem unutar web preglednika.
* Desktop aplikacija
  + Omogućava razmjenu informacija registriranog korisnika i neregistriranog korisnika sa Web API sučeljem unutar desktop aplikacije.
* Web API
* Obrađuje zahtjeve kao posrednik između web aplikacije, desktop aplikacije i baze podataka.
* Baza podataka
* Pohrana podataka o registriranim korisnicima
* Pohrana podataka o svim ostalim entitetima sustava

## 2.2. Nefunkcionalni zahtjevi

Nefunkcionalni zahtjevi opisuju svojstva koja program mora imati. Opisuju karakteristike i ograničenja ponašanja programa, a za naš sustav nefunkcionalni zahtjeva su sljedeći:

* Sustav treba raditi na raznim internetskim preglednicima
* Sustav treba biti dostupan korisnicima u bilo koje vrijeme (0-24h)
* Sustav treba raditi na raznim ekranima, različite razlučivosti
* Sustav treba na siguran način čuvati podatke o korisnicima

## 2.3. Use caseovi

Use caseovi ili obrasci uporabe ili načini korištenja prikazuju tijek događaja korištenja sustava ka postizanju određenog cilja. U nastavku su prikazani use case-ovi za naš sustav:

*Tablica 2.3.1. Opis registracije*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC1 |
| **Namjena** | Registracija gosta  (neregistriranog korisnika) |
| **Naziv** | Registracija |
| **Opis** | Definiraju se podaci koje gost mora unijeti kako bi stvorio korisnički račun. |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik ima korisnički račun |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire opciju prijave. |
| **Opis osnovnog tijeka** | 1. Korisnik unosi ime, prezime, email, broj mobitela, lozinku, grad/mjesto i spol. 2. Provjerava se potpunost i ispravnost podataka unesenih u korisničko sučelje. 3. Podaci se preko Web API poslužitelja spremaju u bazu podataka. 4. Gosta se obavještava o uspješnoj registraciji. |
| **Alternativni slijed** | 1. Gost unosi nepotpune, krive ili već postojeće podatke. 2. Od gosta se traži da promijeni unesene podatke i podnese novi zahtjev za registracijom. |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC2 |
| **Namjena** | Prijava registriranog korisnika |
| **Naziv** | Prijava |
| **Opis** | Definiraju se podaci potrebni za prijavu. |
| **Glavni akter** | Neregistrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | - |
| **Pokretač** | Gost u web ili desktop sučelju odabire opciju registracije. |
| **Opis osnovnog tijeka** | 1. Korisnik unosi korisničko ime i lozinku. 2. Korisnik odabire gumb prijava. 3. Provjerava se potpunost i ispravnost podataka. 4. Korisniku se prikazuje početna stranica sa trenutno aktivnim događajima. |
| **Alternativni slijed** | 1. Korisnik je unio nepotpune ili neispravne podatke. 2. Od korisnika se traži da ponovno unese svoje podatke za prijavu. |

*Tablica 2.3.2. Opis prijave*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC3 |
| **Namjena** | Pregled korisnikovih budućih događaja |
| **Naziv** | Pregled korisnikovih budućih događaja |
| **Opis** | Prikazuje se popis korisnikovih budućih događaja. |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen. |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire opciju moji događaji. |
| **Opis osnovnog tijeka** | 1. Korisnik odabire opciju moji događaji. 2. Korisnik odabire opciju budući događaji. 3. Korisniku se prikazuje lista svih njegovih budućih događaja. |
| **Opis mogućih odstupanja** | 1. Korisniku se prikazuje obavijest da trenutno nema budućih događaja. |

*Tablica 2.3.3. Opis pregleda korisnikovih budućih događaja*

|  |  |
| --- | --- |
| **Redni broj** | UC4 |
| **Namjena** | Pregled korisnikovih prošlih događaja |
| **Naziv** | Pregled korisnikovih prošlih događaja |
| **Opis** | Prikazuje se popis korisnikovih prošlih događaja. |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen. |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire opciju moji događaji. |
| **Opis osnovnog tijeka** | 1. Korisnik odabire opciju moji događaji. 2. Korisnik odabire opciju prošli događaji. 3. Korisniku se prikazuje lista svih njegovih prošlih događaja. |
| **Opis mogućih odstupanja** | 1. Korisniku se prikazuje obavijest s porukom da još nije kreirao događaje. |

*Tablica 2.3.4. Opis pregleda korisnikovih prošlih događaja*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC5 |
| **Namjena** | Traženje aktivnih događaja |
| **Naziv** | Pretraga aktivnih događaja |
| **Opis** | Prikazuje se popis trenutno aktivnih događaja ovisno o filteru pretrage. |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen. |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire opciju traži događaje. |
| **Opis osnovnog tijeka** | 1. Korisnik odabire opciju traži događaje. 2. Korisniku se prikazuje ekran sa popisom trenutnih događaja i sa filterom pretrage. 3. Odabirom određenog filtera pretrage i gumbom traži korisniku se prikazuje odabrani popis pretrage. |
| **Opis mogućih odstupanja** | 1. Korisniku se prikazuje prazna lista. |

*Tablica 2.3.5. Opis pregleda korisnikovih prošlih događaja*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC6 |
| **Namjena** | Kreiranje novog događaja |
| **Naziv** | Kreiranje događaja |
| **Opis** | Odabirom na gumb *novi događaj*  otvara se forma za stvaranje novog događaja. |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen, korisnik je odabrao opciju moji događaji. |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire gumb novi događaj. |
| **Opis osnovnog tijeka** | 1. Korisnik odabire opciju moji događaji. 2. Korisnik odabire gumb novi događaj. 3. Korisniku se prikazuje forma za kreiranje novog događaja. |
| **Opis mogućih odstupanja** | - |

*Tablica 2.3.6. Opis kreiranja novog događaja*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC7 |
| **Namjena** | Uređivanje vlastitih podataka |
| **Naziv** | Postavke profila |
| **Opis** | Korisnik uređuje svoje podatke. |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire opciju postavke profila. |
| **Opis osnovnog tijeka** | 1. Korisnik odabire opciju postavke profila. 2. Korisniku se prikazuje ekran sa podacima o njegovom rofile. |
| **Opis mogućih odstupanja** | - |

*Tablica 2.3.7. Uređivanje vlastitih podataka*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC8 |
| **Namjena** | Prijava na neki događaj |
| **Naziv** | Prijava na događaj |
| **Opis** | Korisnik se prijavljuje na neki događaj |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen, korisnik je odabrao opciju traži događaje. |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire  gumb prijava na događaj. |
| **Opis osnovnog tijeka** | 1. Korisniku se prikazuje ekran sa popisom trenutnih događaja i sa filterom pretrage. 2. Korisnik je odabrao gumb prijava na događaj. 3. Korisnik je prijavljen na događaj. |
| **Alternativni slijed** | 1. Korisnik nije prijavljen na događaj jer su mjesta već popunjena. |

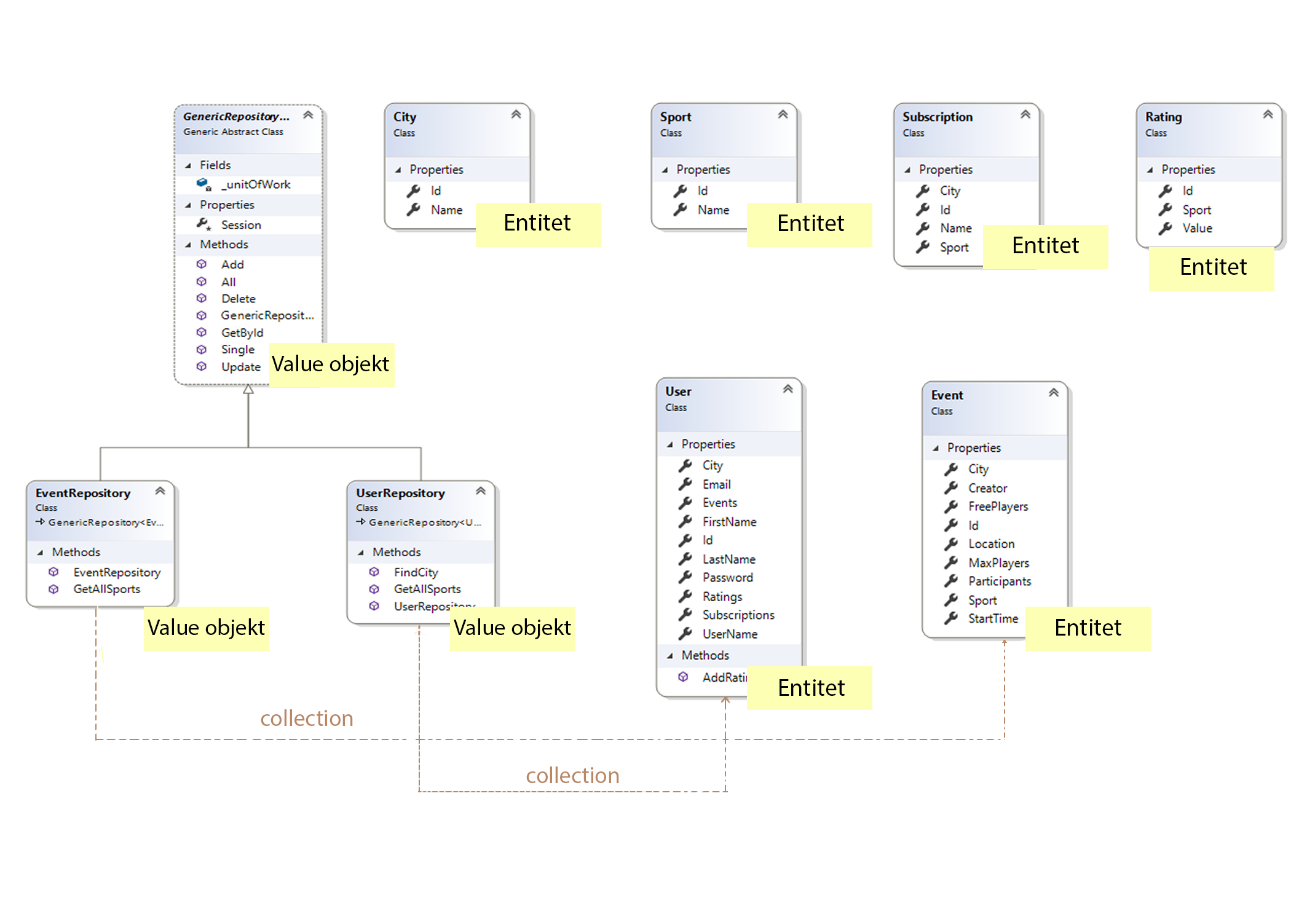
*Tablica 2.3.8. Opis prijave na događaj*

|  |  |
| --- | --- |
| ***Element*** | ***Opis*** |
| **Redni broj** | UC9 |
| **Namjena** | Odjava registriranog korisnika |
| **Naziv** | Odjava korisnika iz sustava |
| **Opis** | Korisnik se odjavljuje iz sustava |
| **Glavni akter** | Registrirani korisnik |
| **Sporedni akteri** | Aplikacija, Web API, baza podataka |
| **Preduvjet** | Korisnik je prijavljen |
| **Pokretač** | Korisnik u web ili desktop sučelju odabire  opciju odjava |
| **Opis osnovnog tijeka** | 1. Korisnik je odabrao opciju odjava |
| **Alternativni slijed** | - |

*Tablica 2.3.9. Opis odjave iz sustava*

# **3. Opis objektnog modela**

Opis objektnog modela sastoji se od dijagrama razreda domenskog problema koji naš sustav rješava. Na slici 3.1. Prikazan je pojednostavljeni dijagram domenskog problema. Kao središnji i glavni entitet Event na kojeg se u nastavku nadovezuju ostali entiteti sustava. Program podržava Evente sportova koji je prikazan entitetom Sport. Svakom korisniku je nakon registracije omogućena prijava na određeni događaj odnosno Time korisnik potvrđuje da će doći sudjelovati tom događaju. Korisnik je prikazan entitetom User dok je grad gdje se događaj odvija i/ili lokacija korisnika prikazana entitetom City. Entitetom Subscription omogućuju se korisnikove pretplate što znači da se korisnik može pretplatiti samo na određene događaje u odabranom gradu i određene sportove koje on želi. Također entitetom Rating se korisniku omogućuje da ocjenjuje druge sudionike koji su prisustvovali događaju. Nadalje, dva value objekta koja sadrže kolekcije Usera i Eventa su UserRepository i EventRepository oba nasljeđuju generički repozitorij sa CRUD i ostalim važnim metodama za rad nad generičkim entitetima.

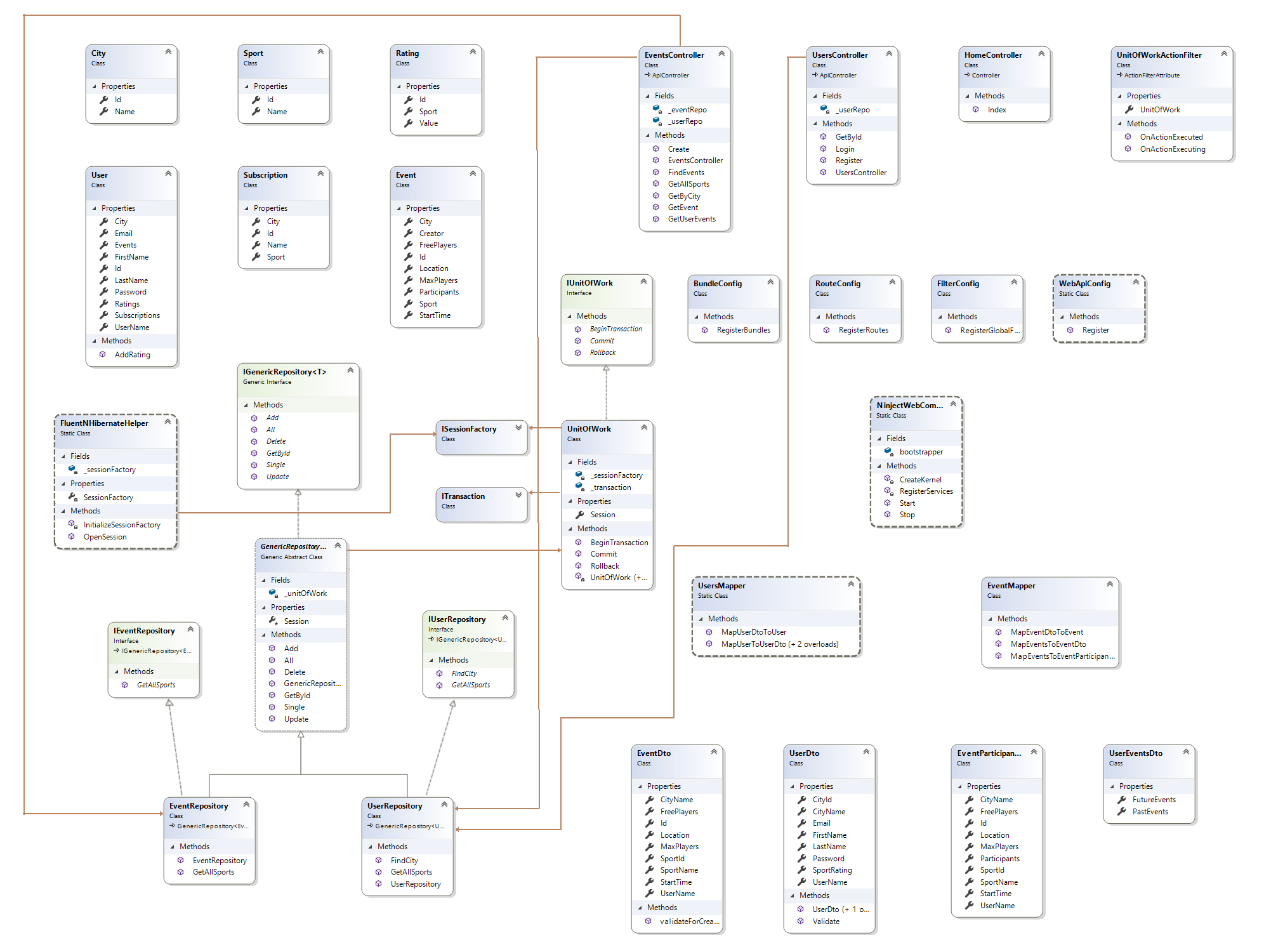


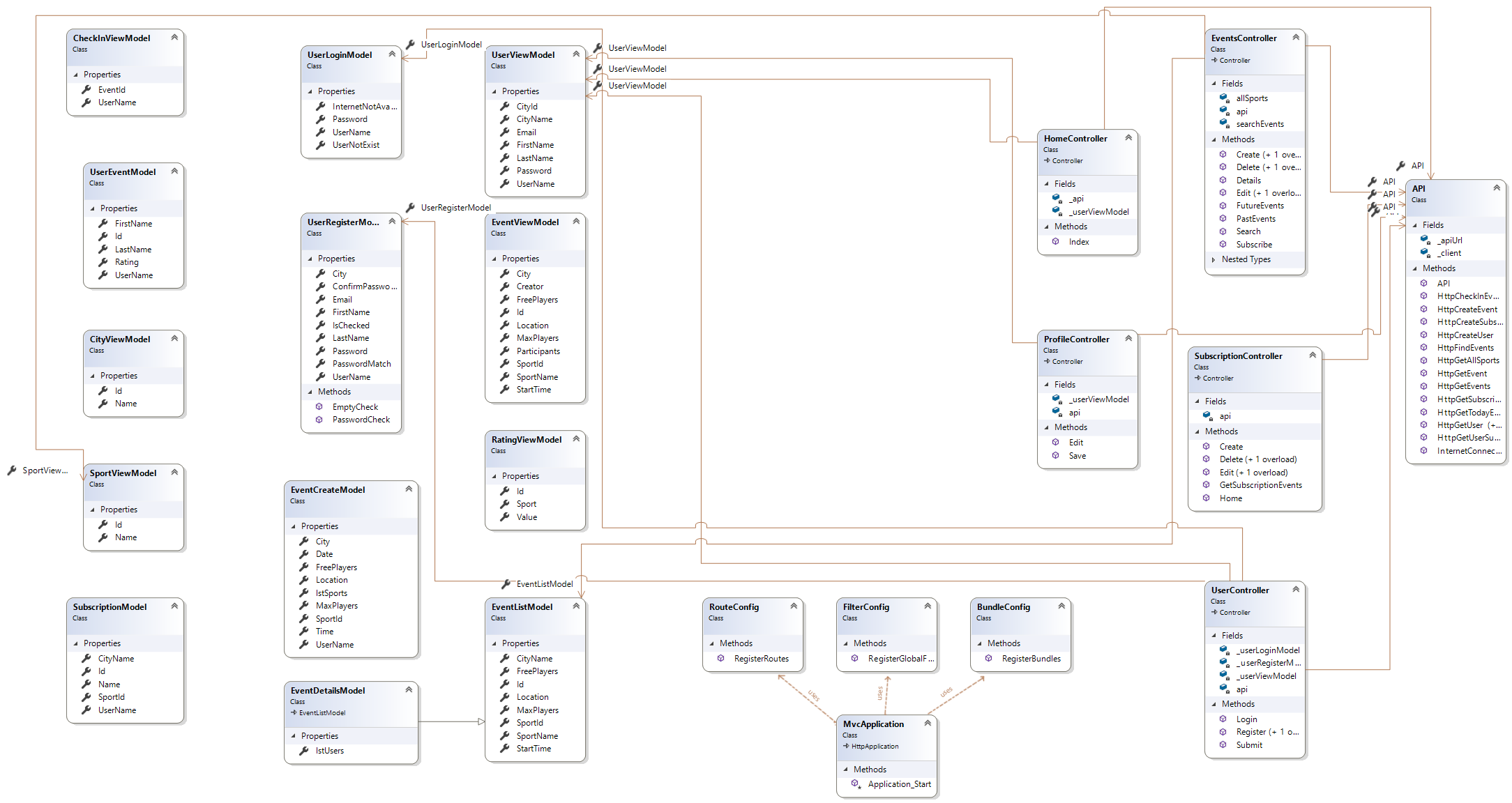
*Slika 3.1. Dijagram razreda objektnog model*

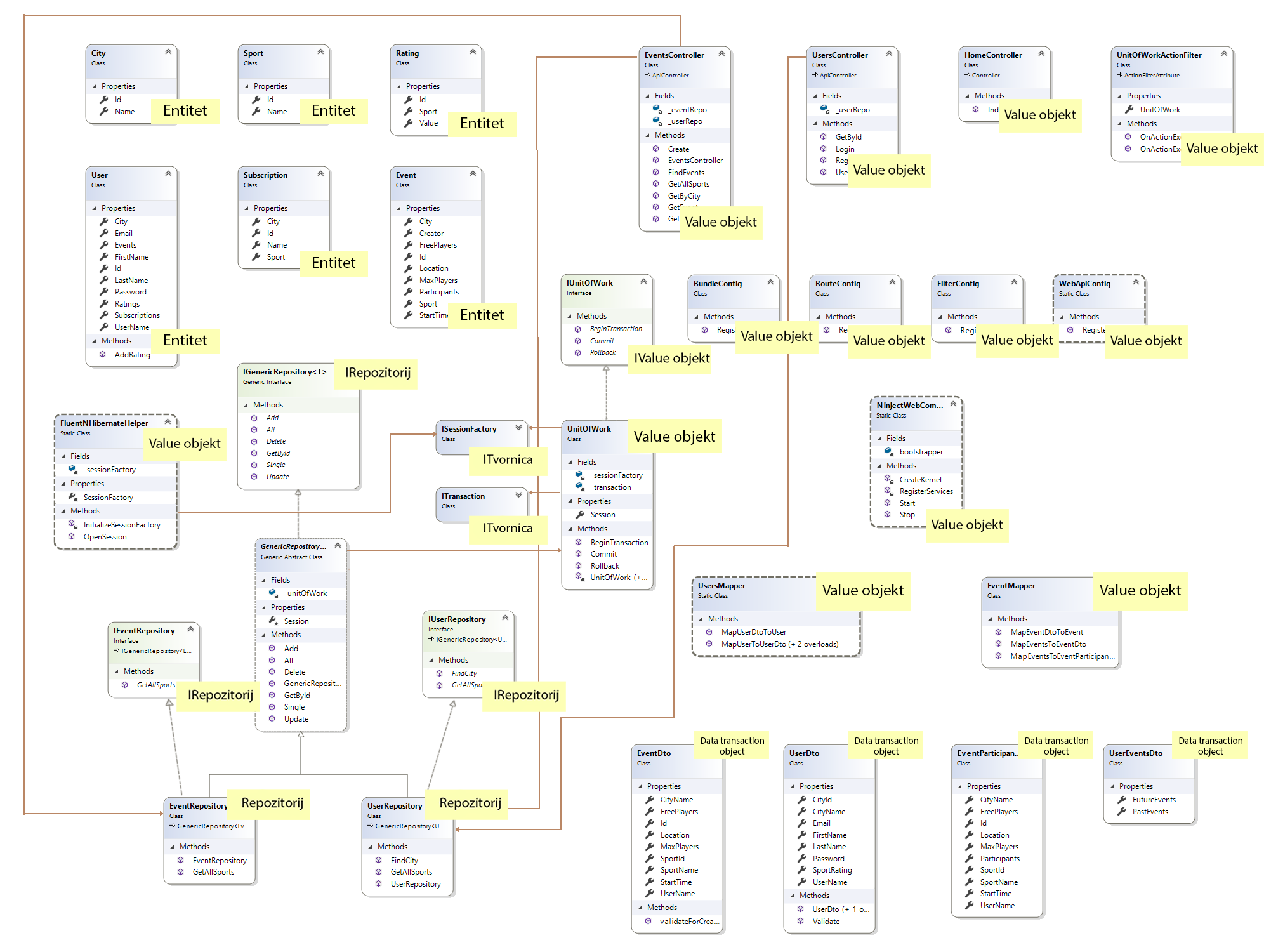
Mnogo složeniji prikaz dijagrama razreda dan je u nastavku. U nastavku na slici 3.2. prikazan je dijagram razreda API programske potpore koji obavlja kontroliran pristup bazi podataka kako bi desktop i web aplikacije mogle primati i slati samo određene i dopuštene zahtjeve. API program sastoji se od velikog broja klasa dok su na slici 3.2. prikazane najvažnije, a to su: modeli domene, generički repozitorij, repozitoriji, obrazac UnitOfWork, koji sadrži ISessionFactory i ITransaction objekte. Također sastoji se od klasa za pokretanje i rad programa, kontrolera i data transfer objekata koji služe za prijenos podataka sa klijentskim aplikacijama.

Nadalje, na slici 3.3. prikazan je dijagram klasa SportyWebApp aplikacije koji prikazuje: view modele, kontrolere, i klase za rad i pokretanje programa.

Nadalje, na slici 3.4. prikazan je API program sa označenim entitetima, value objektima, repozitorijima, data transfer objektima i tvornicama

*3.2. Dijagram razreda API programske potpore*

*Slika 3.3. Dijagram razreda Web aplikacije*

*****3.4. Dijagram razreda API programske potpore sa naznačenim tipovima objekata*

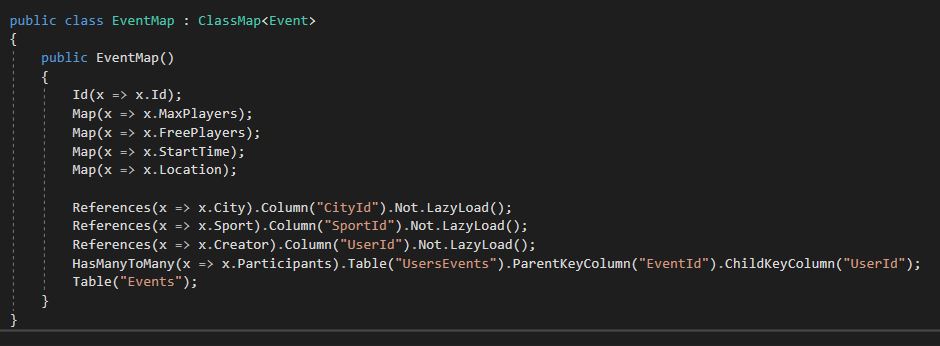
# **4. Opis implementacije perzistencije**

## 4.1. Objektno relacijsko mapiranje

Kako bismo objektni model domene preslikali u relacijski model podataka, potrebno je provesti objektno-relacijsko mapiranje. Postoji više načina kako obaviti mapiranje i ponajviše ovisi o tome kakva nam je hijerarhija objektnog modela i kakvu strukturu tablica u bazi preferiramo. Najjednostavniji pristup je jedna tablica po konkretnoj klasi (eng. Table per concrete class mapping) koji kaže da svaku konkretnu klasu mapiramo u jednu tablicu u bazi, a sva svojstva koja su naslijeđena iz bazne klase postaju stupci u svakoj od tablica.

U našem slučaju nemamo hijerarhije razreda i nisu nam potrebni polimorfni upiti te je zbog toga odabran ovaj tip mapiranja – jedna tablica po konkretnoj klasi. To znači da ćemo svaku klasu posebno mapirati, odnosno odrediti kako se svojstva objekata preslikavaju u stupce u bazi. NHibernate nudi mapiranje u XML ili fluent API zapisu. Fluent API puno je praktičniji za uporabu jer je strong-typed, pisan je u C# kodu i u konačnici ima kraći i pregledniji zapis od XML-a.

Mapiranje modela definirano je u posebnom razredu koji nasljeđuje ClassMap razred unutar FluentNHibernate knjižnice. Ovdje je potrebno navesti sva svojstva domenskog razreda koja se preslikavaju i način na koji se preslikavaju. Ovo je najbolje opisati uz konkretni primjer:



Slika 4‑1 Mapiranje razreda Event

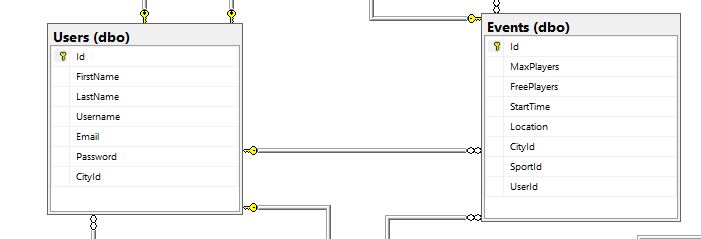
Potrebno je razredu ClassMap proslijediti tip domenskog razreda koji mapiramo kako bi kasnije NHibernate mogao prepoznati da EventMap mapira razred Event. Nakon toga definiramo mapiranje pojedinih svojstava. Prvo definiramo identifikator koji će baza podataka koristiti kao primarni ključ u tablici. Naredba Id(x => x.Id) označava da se identifikator mapira u svojstvo modela koje ima naziv Id. Sljedeća naredba je Map koja radi na sličnom principu, dakle mapira neko svojstvo razreda u jedan stupac tablice. NHibernate sam provjerava kojeg je tipa navedeno svojstvo razreda i određuje mapiranje u prikladni tip u bazi podataka. Možemo definirati i dodatne opcije mapiranja kao što su postavljanje mogućnosti za null vrijednost, dužinu stringa itd. Ako ne definiramo ime stupca u tablici za neko svojstvo onda se uzima ime tog svojstva kako je definirano u razredu domene. Sljedeći blok naredbi odnosi se na povezivanje ovog modela s ostalim modelima koje referencira. Naredbom References stvaramo vezu jedan prema jedan usmjerenu na svojstvo razreda koje sadrži referencu na drugi razred. Uzmimo npr. naredbu

References(x => x.Creator).Column("UserId").Not.LazyLoad();

Definiramo vezu jedan prema jedan s trenutnog entiteta Event na entitet User, a svojstvo razreda Event u kojem je referenca na User razred zove se Creator. Još jedna dobra stvar kod fluent sintakse je ulančavanje naredbi što je vidljivo na gornjem primjeru. Naredbom Column označavamo ime stupca u tablici koji će sadržavati strani ključ entiteta s kojim je povezan. Zadnja naredba u nizu koristi se za konfiguraciju učitavanja po potrebi (LazyLoad). Ukoliko postavimo učitavanje po potrebi, navigacijsko svojstvo Creator biti će učitano tek kada mu se pokuša pristupiti. Na ovaj način sprječavaju se nepotrebni dohvati podataka iz baze.

Nakon dodatne konfiguracije NHibernate knjižnice (definiranje korištenih mapping razreda, definiranje connection stringa itd.) proces O/R mapiranja je gotov. Baza podataka se ovisno o konfiguraciji može sama generirati, a može se i unaprijed ručno stvoriti. Ako se baza stvara ručno treba pripaziti na pravilno imenovanje tablica i stupaca tako da se ono poklapa s definiranim mapiranjem.

Definiranje veze jedan prema jedan u razredu Event prema razredu User rezultira sljedećim tablicama u bazi:

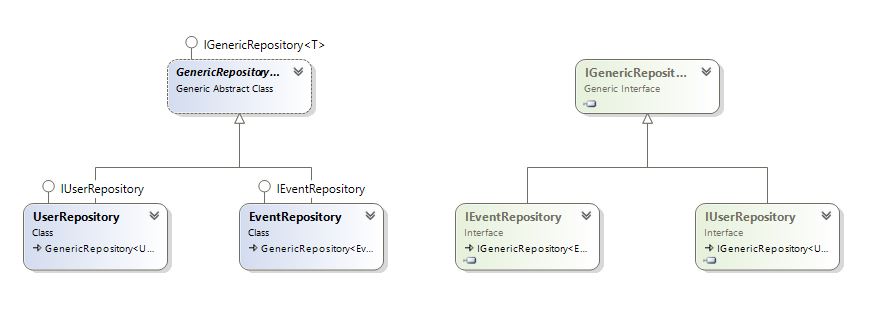


Slika 4‑2 Veza događaja i kreatora događaja

U tablici Events pojavio se stupac UserId koji predstavlja vanjski ključ prema tablici Users. Dijagram prikazuje vezu ove dvije tablice, što potvrđuje da je mapiranje ispravno provedeno. Pošto smo definirali svojstvo Creator za neki događaj i povezali ga na gore navedeni način, sada se korisnik povezan s nekim događajem može dohvatiti preko navigacijskog svojstva.

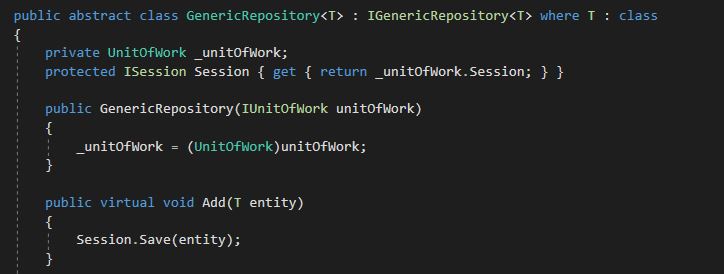
## 4.2. Repozitorij i pristup podacima

U sloju koji se bavi dohvatom podataka implementiran je obrazac repozitorij. Jedna od zadaća ovog obrasca je definirati sučelje kojim se pristupa podacima, a onda po potrebi implementirati konkretne klase za dohvat iz različitih izvora. Još jedna korist repozitorija je olakšano testiranje aplikacijskog sloja jer je za testiranje dovoljno 'mockati' implementaciju repozitorija.



Slika 4‑3 Dijagram razreda repozitorija

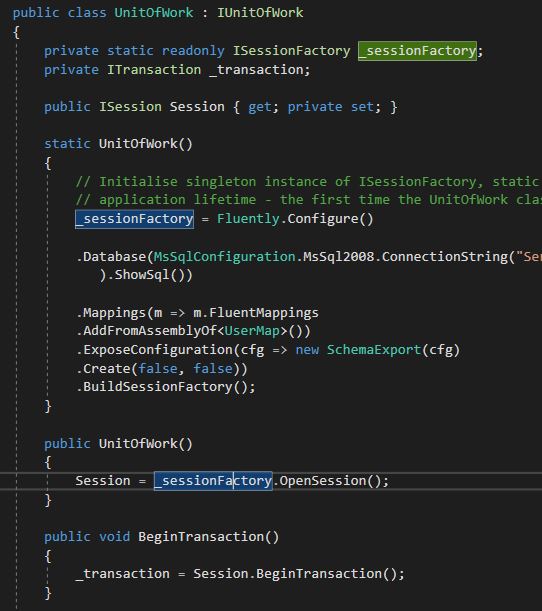
Iz dijagrama razreda vidljiva je hijerarhija razreda u repozitoriju. Osnovno sučelje IGenericRepository sadrži definicije metoda koje nam trebaju u svim daljnjim slojevima, a to su obične CRUD metode (Add, Update, Delete itd.). Druga važna komponenta je apstraktni razred GenericRepository koji nudi konkretnu implementaciju IGenericRepository sučelja, u ovom slučaju to je implementacija vezana uz NHibernate. Ako osnovne metode za pristup podacima nisu dovoljne, potrebno je implementirati dodatne metode specifične za neki entitet. Skup svih specifičnih metoda definiran je npr. u sučelju IEventRepository, a njihova konkretna implementacija u razredu EventReposiroty. Na ovaj način osigurano je da tko god koristi razred tipa IEventRepository može pristupiti osnovnim CRUD metodama i specifičnim metodama.



Slika 4‑4 GenericRepository implementacija

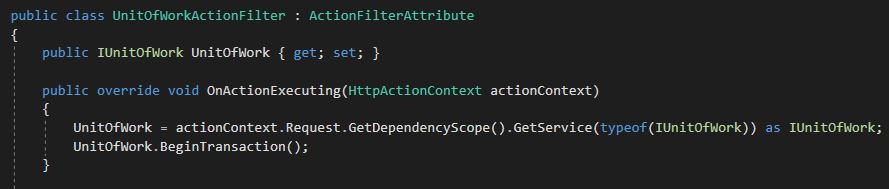
## 4.3. Unit of Work

Veliki prioritet kod svake izmjene podataka je očuvanje perzistencije – svi objekti koji su zahvaćeni nekom transakcijom moraju proći kroz određena stanja i uspješno obaviti zadane promjene. Zadaća Unit of Work obrasca je pratiti stanja objekata, započeti ili završiti transakciju te eventualno vratiti transakciju u početno stanje ako se pojavila greška. Za pristup objektima iz baze koristimo Session objekt kao što je vidljivo u implementaciji generičkog repozitorija (slika 4.4). Stvaranje Session objekta, kao i otvaranje i zatvaranje transakcija prepušteno je razredu UnitOfWork. Metode za stvaranje Session objekata definirane su u ISessionFactory sučelju, a instancu jedne takve tvornice držimo tijekom cijelog izvođenja aplikacije – singleton objekt. Tvornica će se kreirati samo pri prvom pozivu konstruktora UnitOfWork razreda, a dalje će uvijek biti dostupna za stvaranje Session objekata (slika 4.5).



Slika 4‑5 Razred UnitOfWork

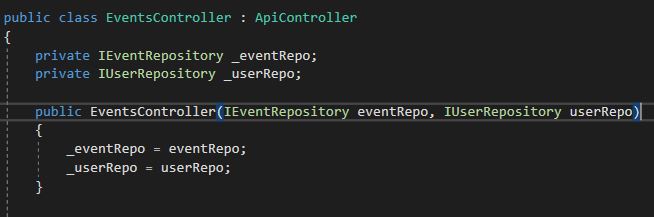
Osnovni princip korištenja je sljedeći – stvara se Session objekt za komunikaciju s izvorom podataka, nakon toga otvara se transakcija, rade se izmjene podataka i na kraju se transakcija zatvara. Dakle dosada je riješeno sve osim otvaranja transakcija, o kojim se također brine UnitOfWork. U njemu se nalaze metode za upravljanje transakcijama (Begin, Commit, Rollback). Kako bi se olakšao rad s transakcijama, pozivi ovih metoda smješteni su u razred UnitOfWorkActionFilter. Ovakva odluka je vođena razmišljanjem da svaka metoda u Web Api kontroleru zapravo radi s resursima iz baze podataka, stoga je uvijek potrebno stvoriti transakciju. ActionFilter omogućava izvođenje akcija prije i nakon izvršavanja metode nekog kontrolera, što je iskorišteno za otvaranje transakcije na početku, i zatvaranje na kraju izvođenja metode. Primjer sa slike 4.6 prikazuje korištenje filtera za otvaranje transakcije.



Slika 4‑6 Action Filter

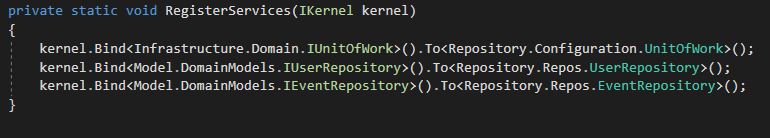
## 4.4. Dependency injection - Ninject

Repozitoriji se koriste u kontrolerima i preko njih se obavljaju operacije nad podacima. Kontroler ne smije znati s kakvim konkretnim tipom repozitorija pristupa podacima, pa tako npr. EventsController ima referencu na objekt tipa IEventRepository. Stvaranje ovog objekta ne bi se trebalo prepustiti kontroleru, nego vanjskoj jedinici za injekciju ovisnosti (eng. Dependency injection). U konkretnom primjeru koristi se injekcija preko konstrutkora, dakle referenca na konkretni repozitorij postavlja se kod stvaranja kontrolera (slika 4.7).



Slika 4‑7 Injekcija u konstruktoru

Kakve će točno konkretne repozitorije kontroler primiti ovisi o konfiguraciji injekcije ovisnosti. U tu svrhu korišten je Ninject framework. U konfiguraciji se navodi koji konkretni tip repozitorija treba instancirati kada se navede neko sučelje repozitorija. Na ovaj način se također postiže laka promjena izvora podataka, odnosno implementacija samog repozitorija. Ako želimo koristiti drugačiji izvor, dovoljno je u konfiguraciji promijeniti konkretni tip koji se vraća.



Slika 4‑8 Konfiguracija DI

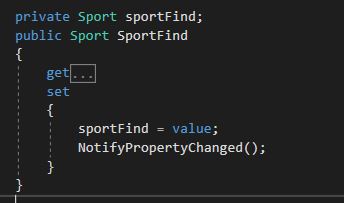
# **5. Opis izgrađene desktop aplikacije**

## 5.1. MVVM obrazac

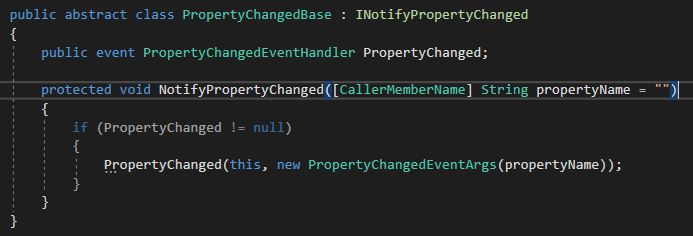
Desktop aplikacija ostvarena je po MVVM (eng. Model-View-ViewModel) obrascu. Namjena ovog obrasca je odvojiti sloj korisničkog sučelja od sloja domenskih modela i logike. Poveznica između ova dva sloja je ViewModel – komponenta koja pretvara domenski model u format pogodan za prezentaciju. Unutra se ne nalaze samo razredi sa svojim svojstvima već i upravljačka logika.

Domenski model realiziran je u class library projektu, a čini ga skup razreda koji reprezentiraju sva bitna svojstva domene. Može se reći da su ti modeli lakša verzija modela kakve imamo u Web Api projektu. Dohvaćanje i slanje podataka prema web servisu također se odvija kroz domenske modele.

ViewModeli nalaze se u aplikacijskom sloju i organizirani su na način da svaki razred odgovara jednom pogledu u korisničkom sučelju. Ovi razredi enkapsuliraju najvažnija svojstva domenskih modela koja su potreba za prikazivanje. ViewModel ima privatna svojstva s pripadajućim javnim getter i setter svojstvima, a na View su povezana preko javnih svojstva (binding). Za praćenje promjena koristi se INotifyPropertyChanged sučelje koje definira metode za obavještavanje View-a o promjenama na VIewModelu. Sučelje je implementirano u apstraktnom razredu, a svakom ViewModelu taj razred postavljen je kao bazni.

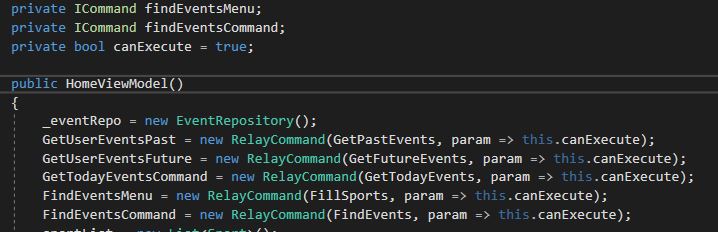


Slika 5-0 Public setter



Slika 5‑1 Implementacija INotifyPropertyChanged

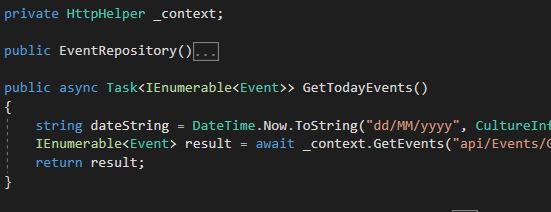
View je logički najjednostavnija komponenta pisana u XML notaciji. Podaci se u korisničkom sučelju prikazuju sukladno promjenama u ViewModelu. Na grafičke elemente postavlja se binding na javna svojstva ViewModela, a moguće je povezati neke komponente, npr. gumb, sa određenom akcijom. Takve akcije definiramo pomoću ICommand sučelja u ViewModelu.



Slika 5‑2 ICommand u VIewModelu

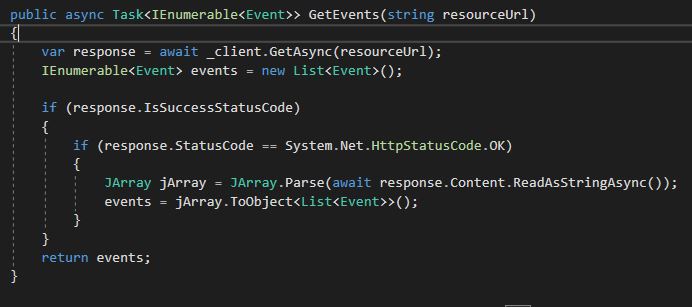
## 5.2. Komunikacija s web servisom

Kao izvor podataka aplikacija koristi udaljeno REST sučelje. Implementiran je jednostavni repozitorij s definiranim metodama dohvata podataka. Konkretna implementacija nalazi se npr. u razredu EventRepository, koji kao kontekst za pristup podacima koristi razred HttpHelper.



Slika 5‑3 Pozivanje metode za dohvat događaja

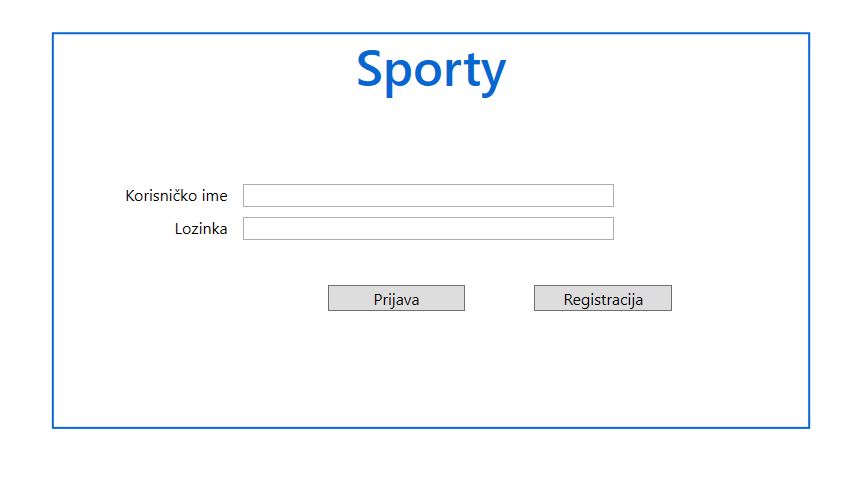
HttpHelper implementira HTTP pozive prema udaljenom sučelju. Također obavlja serijalizaciju i deserijalizaciju JSON objekata u objekte domene. Sve metode rade asinkrono što znači da se prilikom čekanja na odgovor ne blokira korisničko sučelje.



Slika 5‑4 HTTP zahtjev i deserijalizacija

## 5.3. Korisničko sučelje

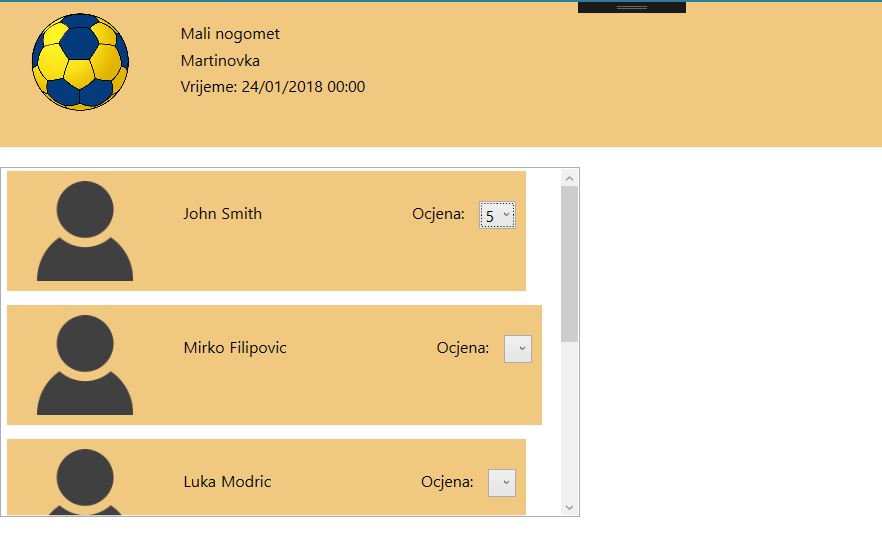
Korisnik se prije korištenja aplikacije mora prijaviti u sustav ili se registrirati kao novi korisnik. Nakon te dvije forme, većina akcija događa se na glavnom prozoru u kojemu se prikazuju liste događaja, opcije pretraživanja i ostale funkcionalnosti zajedno s glavnim izbornikom



Slika 5‑5 Forma za prijavu



Slika 5‑6 Glavni izbornik



Slika 5‑7 Popis sudionika događaja

# **6. Opis izgrađene web aplikacije**

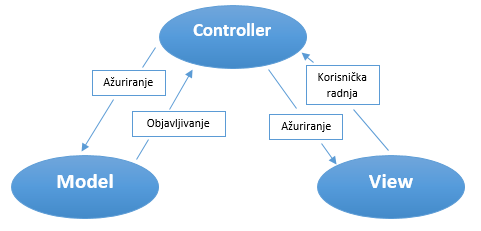
## 6.1. Arhitektura web aplikacije - MVC (Model-View-Controller)

MVC je obrazac softverske arhitekture. Koristi se u softverskom inženjeringu za odvajanje pojedinih dijelova aplikacije u komponente ovisno o njihovoj namjeni. Model se sastoji od podataka, poslovnih pravila, logike, i funkcija ugrađenih u programsku logiku (business logic).

MVC arhitektura aplikaciju dijeli u osnove komponente, a to su: model, pogled (View) i kotroler (Controller). Ovakva podjela utječe na razdvajanje programiranja obrade podataka od programiranja sučelja te zaprimanja zahtjeva korisnika. Time se omogućava razvoj i napredak aplikacije u nekom od objektnih programskih jezika (C++, Java, C#…), a istovremeno pregled i kontrola nad HTML-om, CSS-om i JavaScript-om pomoću kojih se izgrađuje pogled (View).

Kako se svaki dio neke složene aplikacije može posebno razvijati, olakšano je i samo upravljanje razvoja složenih aplikacija. Ova karakteristika MVC aplikacija olakšava organizaciju razvoja aplikacije, ali i nadogradnju novim verzijama. Svaka se komponenta može odvojeno testirati te je tako olakšano testiranje web aplikacije (TDD – Test-Driven Development).

Model-View-Controller aplikacija komunikaciju odvija putem pogleda gdje korisnik unosi zahtjev kojim se šalju naredbe kotroleru, a zatim kontroler traži od modela obavljanje zadane radnje i vraćanje rezultata. Dani rezultat radnje kontroler šalje u pogled gdje ga korisnik, koji je unio zahtjev, može vidjeti.



Slika 6.0‑1. MVC arhitektura

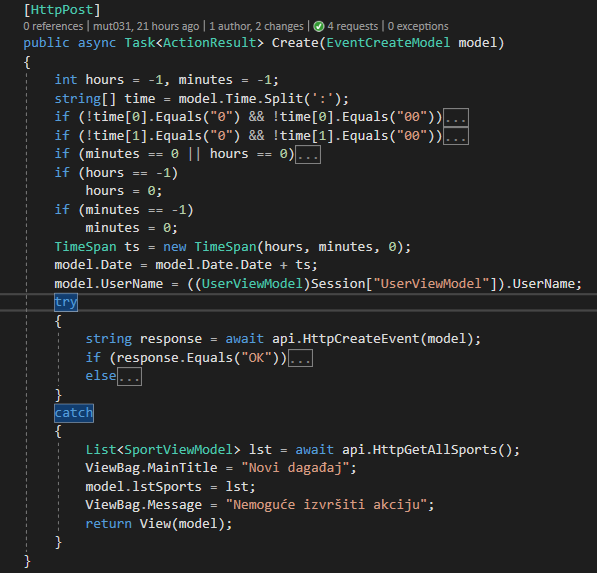
## 6.2. Implementacija programskog rješenja

### 6.2.1. Kontroleri

Aplikacija sadrži 5 kontrolera:

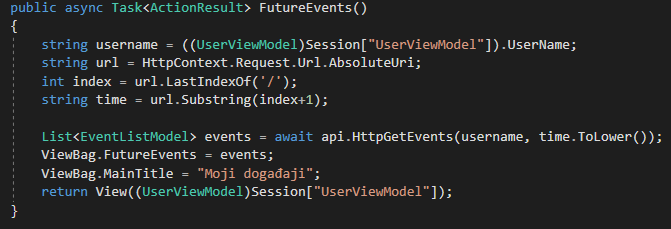
* EventsController
* HomeController
* ProfileController
* SubscriptionController
* UserController

EventsController zadužen je za sportske događaje. Da bi se napravio novi događaj potrebno je navesti podatke kao što su naziv sporta koji će se igrati, grad i lokacija gdje će se održati utakmica i ostale podatke koji opisuju događaj. Korisnik popunjava formu sa svim podacima na pogledu za stvaranje novog događaja te se ti podaci šalju kontroleru u HTTP POST metodu Create (Figure 6.4). U metodi se provjerava valjanost unesenog vremena početka događaja te se podaci šalju na API servis za daljnju obradu.



Slika 6.0‑2. HTTP POST metoda za stvaranje novog događaja

Moguće je pregledati sve događaje u kojima je korisnik sudionik, odnosno događaje u koje se prijavio i događaje koje je stvorio jer čim je stvorio novi događaj, postao je i sudionik istoga. Da bi korisnik pregledao svoje događaje u glavnom izborniku bira karticu Moji događaji te mu se prikazuju događaji gdje može birati između pregleda budućih i prošlih događaja. Za pregled budućih događaja u kontroleru se poziva metoda FutureEvents (Slika 6.5.) iz koje se na API servis šalje korisničko ime korisnika koji je zatražio pregled događaja te još jedan parametar koji govori da se radi o budućim događajima i čita se iz URL-a. Na isti način se obavlja dohvaćanje prošlih događaja funkcijom PastEvents, sada će drugi parametar predstavljati prošle događaje.



Slika 6.4. Dohvaćanje korisnikovih budućih događaja

Korisniku je omogućena i pretraga događaja. Moguće je pretraživati po sportu, gradu, datumu i broju preostalih slobodnih sudionika događaja. Podaci se šalju kontroleru u metodu Search koja poziva API servis te vraća korisniku nazad tražene događaje.

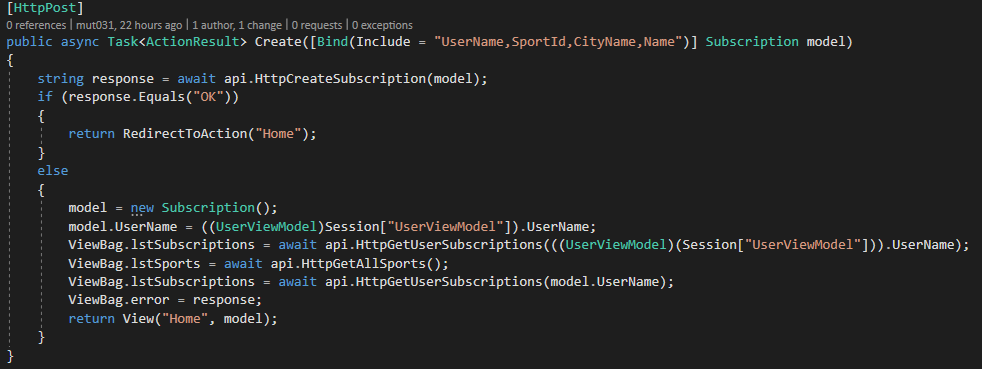


Slika 6.4. Pretraga događaja

HomeController je početna točka nakon logina gdje se korisniku prikazuju svi događaji koji će danas biti u gradu koji je korisnik naveo prilikom registracije.

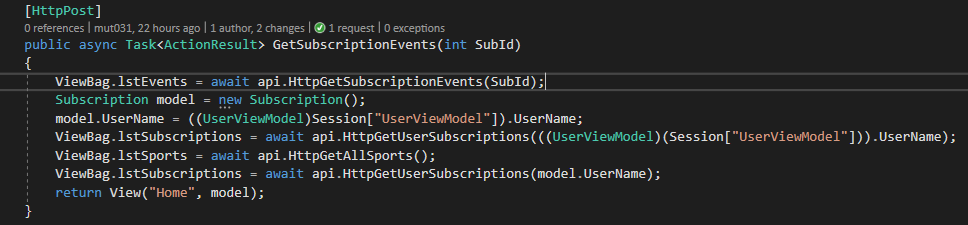
ProfileController ima metodu Edit koja omogućava korisniku da izmjeni svoje postojeće podatke o računu.

SubscriptionController obavlja poslove vezane uz pretplate. Za stvaranje nove pretplate potrebno je odabrati sport i grad koji se želi pratiti te korisnik daje naziv svojoj pretplati po kojem će ju kasnije prepoznati.



Slika 6.5 Stvaranje nove pretplate

U kontroleru se još nalazi metoda GetSubscriptionEvents koja dohvaća sve korisnikove preplate kako bi mu se omogući da iz padajućeg izbornika može izabrati postojeću pretplatu te pregledati odgovarajuće događaje.

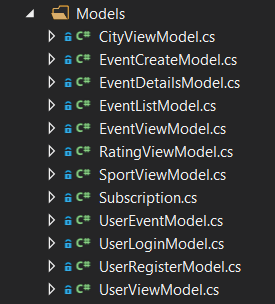


Slika 6.6. Dohvaćanje svih pretplata korisnika

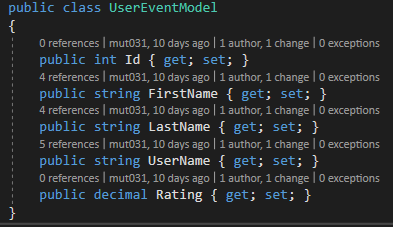
UserController služi za kontrolu korisnika. Ima metodu Register koja stvara novog korisnika te metodu Submit u kojoj se prijavljuje postojećeg korisnika u aplikaciju.

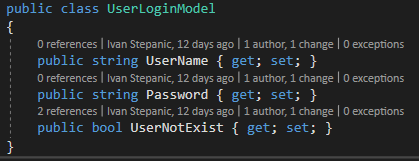
### 6.2.2. Modeli

Svaki model stvoren je za konkretnu situaciju i sadrži određena svojstva, ovisno o tome kakvi su bili zahtjevi, da bi se što lakše upravljalo objektima. Svi modeli korišteni u izradi aplikacije prikazani su na sljedećoj slici.



Slika 6.7. Modeli aplikacije

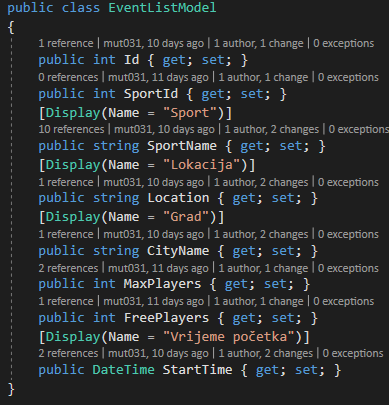
UserRegisterModel sadrži sve informacije o korisniku odnosno taj se model s popunjenim podacima, kod registracije novog korisnika, šalje kontroleru koji dalje registraciju. U nekim drugim situacijama neće biti potrebne sve te informacije pa se stvaraju novi modeli koji su prilagođeni takvim situacijama. UserLoginModel služi za prijavu u aplikaciju i potrebni su samo korisničko ime i lozinka dok UserEventModel sadrži podatke o korisniku koji se prikazuju na pregledu događaja (ime, prezime, korisničko ime i ocjena).



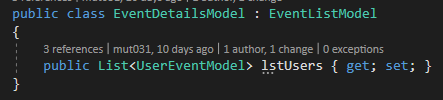
Slika 6.9. Model za prikaz na događaju

Slika 6.8. Model za prijavu korisnika

Kada je potrebno prikazati listu događaja, za prikaz jednog događaja koristi se EventListModel koji sadrži sve podatke o događaju (sport, grad, lokaciju, broj igrača, vrijeme početka). Za prikaz detalja o jednom događaju potrebni su svi ti podaci i još lista svi korisnika koji su prijavljeni u tom događaju. EventDetailsModel se koristi kod prikaza detalja te kako su mu potrebni svi podaci koji se nalaze u EventListModelu dovoljno je da nasljeđuje tu klasu i ima još svoju listu korisnika.



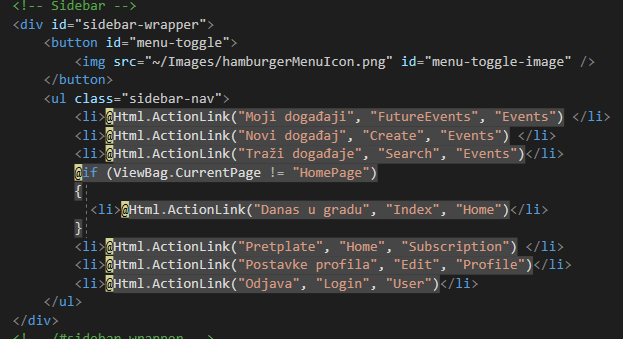
Slika 6.10.



Slika 6.11.

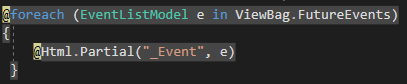
### 6.2.3. Pogledi

Svaki pogled u aplikaciji, osim logina i registracije, ima s lijeve strane glavni izbornik. Taj izbornik napravljen je u parcijalnom pogledu \_Layout i predložak je za ostale poglede.

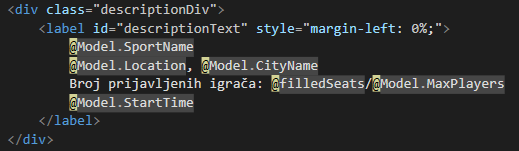


Slika 6.12. Glavni izbornik u parcijalnom pogledu \_Layout

Isto tako za prikaz liste događaja, za svaki događaj poziva se parcijalni pogled \_Event u kojem je opisano kako se prikazuje jedan sportski događaj s podacima o sportu, gradu, lokaciji, početku događaja te broju slobodnih i maksimalnih igrača.



Slika 6.13. Pozivanje parcijalnog pogleda \_Event



Slika 6.14. \_Event – prikaz jednog događaja

Prilikom dolaska na stranicu web aplikacije, korisniku se prikazuje početni pogled Login (Slika 6.15.) gdje se može prijaviti u aplikaciju sa već postojećim podacima ukoliko je registriran ili napraviti novi račun gdje mora upisati svoje korisničke podatke na pogledu za registraciju (Slika 6.16.).



Slika 6.15. Početni zaslon za login postojećeg korisnika



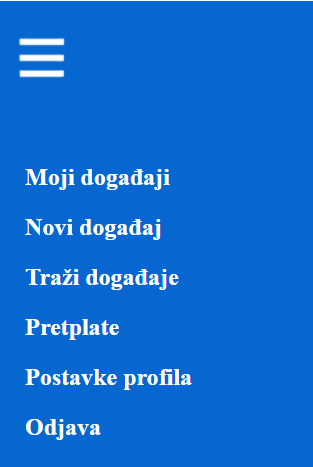
Slika 6.17. Registracija novog korisnika

Prilikom registracije, jedan od podataka koje je korisnik morao unijeti je grad u kojem se nalazi. Nakon uspješne registracije i prijave u aplikaciju, korisniku su prikazani svi događaji koji se danas odvijaju upravo u tom gradu kojeg je naveo. Za svaki događaj prikazano je koji sport se igra, lokacija gdje će se igrati, vrijeme početka, maksimalan broj igrača te broj preostalih mogućih mjesta koja mogu popuniti drugi korisnici aplikacije pritiskom na tipku PRIJAVA (Slika 6.18.). Na prikazu događaja nalazi se i tipka DETALJI koja korisnika vodi na drugu stranicu gdje se, uz već navedene podatke, prikazuju korisnici koji se nalaze u događaju, odnosno korisnici koji su se prijavili na taj događaj.



Slika 6.18. Pregled današnjih događaja u gradu

S lijeve strane korisnikovog sučelja nalazi se glavni izbornik koji nam nudi sljedeće opcije: Moji događaji, Novi događaj, Traži događaje, Pretplate, Postavke profila te Odjava.



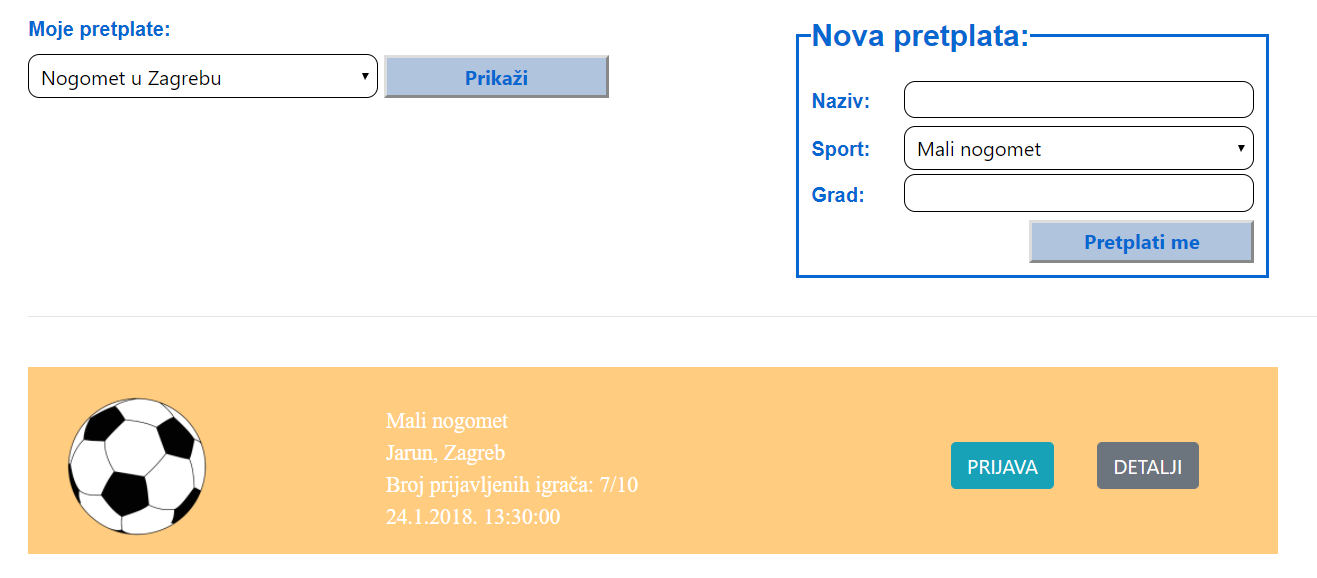
Slika 6.19. Glavni izbornik

Pritiskom na gumb Moji događaji, korisnik može vidjeti listu svojih budućih događaja (s informacijama o događaju: sport, mjesto, broj prijavljenih igrača te vrijeme) na koje se već prijavio ili događaje koje je sam izradio (korisnik je automatski prijavljen kao sudionik tog događaja). Iznad Budućih događaja nalazi se i gumb Prošli događaji te klikom na taj gumb korisnik vidi listu svih događaja koji su završili, a u kojima je sudjelovao.

Sljedeća opcija u glavnom izborniku je Novi događaj. Klikom na taj gumb otvara se novi pogled u kojem korisnik ima mogućnost kreiranja novog sportskog događaja. Polja koja korisnik treba popuniti kako bi se kreirao novi događaj su: sport, datum, vrijeme, grad, lokacija, ukupan broj igrača te broj potrebnih igrača. Klikom na gumb Kreiraj korisnik stvara novi događaj te mu se otvara kartica Moji događaji s dodanim novonastalim događajem.

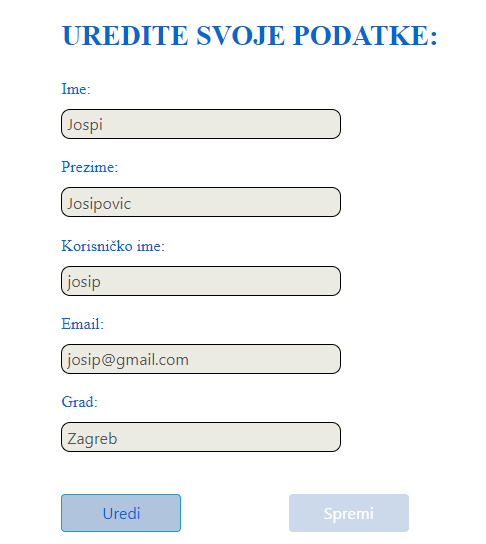
Opcija Traži događaje vodi korisnika na pogled koji omogućuje pretragu svih kreiranih događaja. Korisnik u padajućim izbornicima odabire koji ga sport zanima, datum, grad te broj slobodnih mjesta, a zatim klikom na gumb TRAŽI dobiva rezultate (ako ih ima).

Klikom na Pretplate otvara se pogled koji nudi mogućnost „pretplate“ na neki sport u nekom gradu (grad drugačiji od onoga navedenog prilikom registracije). U ovom pogledu ponuđene su dvije opcije: Moje pretplate (u kojima se može odabrati već postojeće pretplate) te Nova pretplata (korisnik izrađuje novu pretplatu upisivanjem naziva pretplate, odabirom sporta te upisom grada i klikom na gumb Pretplati me).



*Slika 6.20. Korisnikove pretplate te izrada novih*

Opcija Postavke profila otvara novi pogled dopušta korisniku izmjene osobnih podataka koje je unio prilikom registracije. Ispod upisanih podataka ( Ime, Prezime, Korisničko ime, Email i Grad) nalaze se dva gumba: Uredi i Spremi. Klikom na gumb Uredi, omogućene su promjene u navedenim podacima. Nakon željenih promjena, korisnik pritisne na gumb Spremi te se promjene spremaju.



*Slika 6.21. Uređivanje osobnih podataka*

Zadnja opcija u glavnom izborniku je opcija Odjava, a klikom na nju korisnik se odjavljuje iz aplikacije Sporty.