Interaktivna aplikacija za obožavatelje nogometnih klubova

Tomaić, Josip

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

Diplomski studij

INTERAKTIVNA APLIKACIJA ZA OBOŽAVATELJE NOGOMETNIH KLUBOVA

Diplomski rad

Student: Josip Tomaić, br. indeksa D-817R

Mentor: doc.dr.sc. Ivica Lukić

Osijek, prosinac 2017.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 07.12.2017.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Josip Tomaić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 817 R, 12.10.2015.
OIB studenta:	25004713938
Mentor:	Doc.dr.sc. Ivica Lukić
Sumentor:	
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Doc.dr.sc. Mirko Köhler
Član Povjerenstva:	Doc.dr.sc. Zdravko Krpić
Naslov diplomskog rada:	Interaktivna aplikacija za obožavatelje nogometnih klubova
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Napraviti aplikaciju koja će voditi detaljne podatke o radu kluba, njegovoj povijesti, uspjesima i igračima. Različite statistike biti će prikazane pomoću dijagrama i osvježavane sa novim podacima. Prijavljeni korisnici će moći sastaviti svoju omiljenu nogometnu formaciju od igrača kluba te odigrati kviz o poznavanju kluba.
Prijedlog ocjene pismenog dijela ispita (diplomskog 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: 2 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	07.12.2017.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 17.12.2017.

Ime i prezime studenta:	Josip Tomaić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D 817 R, 12.10.2015.
Ephorus podudaranje [%]:	4%

Ovom izjavom izjavljujem da je rad pod nazivom**: Interaktivna aplikacija za obožavatelje nogometnih klubova**

izrađen pod vodstvom mentora Doc.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:

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA

Ja, Josip Tomaić, OIB: 25004713938, student/ica na studiju: Diplomski sveučilišni studij Računarstvo, dajem suglasnost Fakultetu elektrotehnike, računarstva i informacijskih tehnologija Osijek da pohrani i javno objavi moj **diplomski rad**:

Interaktivna aplikacija za obožavatelje nogometnih klubova

u javno dostupnom fakultetskom, sveučilišnom i nacionalnom repozitoriju.

Osijek, 17.12.2017.

_____

potpis

Sadržaj

1. Uvod

Kako svaki projekt zahtijeva izradu dokumentacije tako je potrebno izraditi dokumentaciju i za ovaj projekt koji će biti opisan u ovoj dokumentaciji. Na samom početku, prije nego je i razrađena tema u potpunosti, bilo je potrebno odlučiti se za područje znanosti unutar kojeg će biti odabrana tema diplomskog rada. Naravno, programiranje je jako širok pojam i nije bilo jednostavno odabrati točno određeni segment, točno određeni pojam i na kraju točno određenu temu. Tijekom zadnje godine studija bilo je potrebno odraditi i praksu kako bi se studenti bolje upoznali sa radnim okruženjem i možda i odredili točno određeno područje svog zanimanja za daljnji razvoj i napredak. Na temelju znanja i vještina stečenih na stručnoj praksi, odabir teme između različitih mogućnosti suzio se na točno određeno područje, a to je kombinacija C# programskog jezika i ASP .NET razvojnog okruženja uz kombinaciju okruženja za prikaz programskog dijela unutar web preglednika. Tako se došlo do teme diplomskog rada koja će biti objašnjena do sitnih detalja u ovoj dokumentaciji.

1.1. Zadatak diplomskog rada

Tema ovog diplomskog rada je izrada interaktivne web aplikacije za obožavatelje pet najpopularnijih nogometnih klubova. Aplikacija sadržava zanimljivosti vezane uz klubove koji se nalaze u aplikaciji kao što su povijest kluba, trenutni sastav, broj osvojenih trofeja, statistike igrača i druge.

Da bi se aplikacija realizirala bilo je potrebno koristiti sljedeće tehnologije i programske jezike:

- C#
- ASP .NET
- Angular (2-4)
- HTML
- CSS
- Visual Studio
- SQL Server Management Studio

U nastavku će biti objašnjene svaka od navedenih tehnologija te svaki od navedenih programskih jezika.

2. Korištene tehnologije

2.1. Programski jezik C#

C# je objektno-orijentirani programski jezik razvijen od strane Microsofta. Razvijen je 2000. godine od strane razvojnog tima sa Andersom Hejlsbergom na čelu. Razlog nastanka ovog programskog jezika je nedostatak jezika za Microsoftovo .NET okruženje. Naime, tvrtka Sun koja je vlasnik Java programskog jezika nije dozvolila Microsoftu da mijenja Javu. Microsoft je bio počeo sa razvojem Visual J++ programskog jezika ali zbog nesuglasica sa tvrtkom Sun taj projekt je propao. Razlog nesuglasica je što su iz tvrtke Sun tvrdili da Microsoft stvara kopiju Java programskog jezika. Nakon što je razvijen programski jezik C# tvrtka Sun je smatrala taj programski jezik imitacijom njihovog izuma (Jave). Anders Hejlsberg je demantirao te tvrdnje te objasnio da je C# programski jezik koji je više sličan programskom jeziku C++ nego Java programskom jeziku. Nakon 2005. godine Java i C# su postali manje slični jer su se počeli razvijati različitim putem. Iduća bitna stavka koja je implementirana u C# programsku jeziku je generičko programiranje. To je pojam koji označava stil programiranja u kojem se pišu algoritmi uz uvjet da se tip podataka specificira prilikom implementacije algoritma ovisno o tipu parametara koji su potrebni algoritmu. Ovakav stil programiranja je od velike koristi jer se na taj način ne kreiraju metode koje se razlikuju samo u tipu parametara koje primaju tj. sprječava se dupliciranje koda jer se kreira samo jedna jedinstvena metoda ili klasa koja može biti bilo kojeg tipa prilikom implementacije. Nakon daljnjeg razvoja ovog programskog jezika došlo je do uvođenja LINQ dodataka u C# programski jezik te je na taj način omogućeno korištenje lambda izraza (eng. lambda expressions) i anonimnih tipova. Korištenjem tih dodatnih mogućnosti programeri mogu primjenjivati C# delegate. Da ne bi neki od pojmova ostali nejasni potrebno je objasniti što su lambda izrazi, anonimni tipovi (podataka i metoda) te što su delegati. Dakle, lambda izrazi su prema službenoj definiciji anonimne funkcije koje mogu sadržavati izraze i tvrdnje i mogu se koristiti za izradu delegata. Korištenjem lambda izraza može se napisati lokalna funkcija koja se može koristiti kao parametar ili vraćena kao vrijednost poziva funkcije, a uglavnom se koriste kod pisanja LINQ upiti (eng. query) kao npr. provjera da li jednodimenzionalno polje cijelih brojeva sadrži određeni cijeli broj. Anonimni tipovi su značajka koja dozvoljava podatkovnim tipovima enkapsulaciju atributa u jednom objektu bez prethodnog navođenja tipa tog objekta. Takav način definiranja objekata je veoma sličan SQL-u. Na taj način se npr. objektu „Osoba“ mogu pridružiti atributi: „Ime“, „Prezime“, „OIB“ itd. Još je potrebno objasniti što su to delegati. Delegati su klase koje se koriste za prenošenje metode kao parametra. To znači da se prilikom instanciranja objekta klase tipa „delegate“ može metoda iz klase u kojoj se instancira objekt postaviti kao parametar tog objekta. Delegatima se mogu pozvati i više metoda odjednom korištenjem operatora „+=“.

2.1.1. Dependency injection

Dependency injection je shema za dizajn softvera koja dozvoljava da kod bude napisan „loosely coupled“ stilom. To znači da promjena jednog dijela koda neće uzrokovati neočekivane promjene na drugim dijelovima koda. Drugi način razumijevanja „loosely

coupled“ stila je promatranjem međuovisnosti između dviju klasa. Općenito se promatra koliko dizajn klase A ovisi o dizajnu klase B tj. koliko promjene u klasi A utječu na dizajn u klasi B u slučaju da se u klasi B instancira klasa A. Dakle, „loosely coupled“ znači da su klase A i B neovisne jedna o drugoj te da promjene u bilo kojoj od tih dviju klasa neće uzrokovati promjene u obje klase. Suprotno od „loosely coupled“ je „tight coupled“ što označava da bilo kakva promjena u klasi A uzrokuje promjenu i u klasi B što je veoma loša praksa jer zadaje velike probleme prilikom održavanja i unapređivanja programskog koda. Razlog tome je što ukoliko se instanca promijenjene klase nalazi u još desetak ili ponekad čak i više klasa, programer mora mijenjati dizajn svake od tih klasa što je dugotrajan posao. Iz ovoga se da zaključiti da puno efikasnije i bolje koristiti „loosely coupled“ stil pisanja koda u odnosu na „tight coupled“ stil. Najčešći primjer korištenja dependency injection sheme za dizajn softvera je kreiranje sučelja (eng. interface) za klase te na taj način svaka druga klasa koja treba pristupiti metodama iz klase pristupa im preko sučelja, a ne direktno preko instance klase. Pri tome se za instanciranje sučelja ne koristi ključna riječ *new* nego se koristi konstruktor klase te se objekt sučelja te klase predaje kao parametar konstruktoru.

Postoje tri vrste implementacije dependency injection sheme, a to su: preko konstruktora, preko atributa i preko metode klase.

2.1.2. Ninject

Ninject je jednostavno okruženje koje se koristi zajedno sa dependency injection shemom za dizajn softvera. Ninject okruženje pomaže prilikom rastavljanja aplikacije na manje „loosely coupled“ dijelove te nakon toga ih povezuje natrag u jednu cjelinu.

2.1.3. Automapper

Automapper je alat koji se koristi za kopiranje podataka iz jednog objekta u drugi. Svrhu automappera je najlakše objasniti na primjeru. Ako postoje klasa A i klasa B te je cilj preslikati vrijednosti atributa klase A u klasu B. Uobičajeni način je da se stvore instance obje klase korištenjem ključne riječi *new* te se nakon toga pristupi atributima svake od klasa korištenjem znaka . i na taj način se obavi preslikavanje vrijednosti atributa iz jedne klase u drugu. Međutim to je veoma loš način ukoliko je potrebno to obaviti za desetak i više klasa. Tada do izražaja dolazi automapper i njegova funkcionalnost. Automapper jednom linijom koda rješava problem preslikavanja te ukoliko postoji deset klasa koje je potrebno povezati taj posao će automapper riješiti u deset linija programskog koda. Takav način povezivanja je od velike koristi kod povezivanja aplikacija koje se sastoje od više razina te će takva primjena biti vidljiva u ovom diplomskom radu.

2.2. ASP .NET

ASP .NET je open-source okruženje za backend web aplikacija kreiran u svrhu izrade dinamičnih web stranica. Pod dinamične web stranice se misli na stranice čiji sadržaj kontrolira neki server ili API. Ovo okruženje je razvio Microsoft kako bi omogućio programerima da razvijaju web aplikacije, web servise i dinamične web stranice.

Način na koji se programski kod pretvara u prikaz u web pregledniku korisnika koji je pokrenuo web aplikaciju ili otvorio web stranicu je sljedeći:

- Kod se izvršava u CLR-u (eng. Common Language Runtime)
- Pretvorba instrukcija u strojni kod u realnom vremenu pomoću JIT kompajlera (eng. Just In Time)
- Podaci se prikazuju korisniku, a rezultat kompajliranja se sprema u međuspremnik za kasniju upotrebu

Razlog spremanja rezultata kompajliranja u međuspremnik je taj što samo kompajliranje zahtjeva dosta vremena te bi u slučaju konstantnog kompajliranja prilikom svakog otvaranja web sadržaja korisnik morao dugo čekati na prikaz sadržaja koji želi vidjeti.

Web stranice u ASP .NET-u se službeno zovu web forme i smatraju se temeljnim dijelom za izradu web aplikacija u navedenom okruženju. Postoje dvije metodologije koje se mogu primijeniti za izradu web formi, a to su: format web stranice i format web aplikacije. Razlika između te dvije metodologije je što se web aplikacije prvo moraju kompilirati prije implementacije dok web stranice dozvoljavaju kopiranje datoteka i sadržaja web stranica na server bez potrebe za prethodnom kompilacijom.

Također, bitno je spomenuti i web servise koju su novost u .NET-u. Točnije, radi se o standardu koji je prihvaćen, a odnosi se na komunikaciju između aplikacija prenošenjem SOAP poruka putem web-a. Putem prihvaćenog standarda omogućena je integracija raznih aplikacija i sustava putem standardiziranog načina komunikacije.

2.2.1. MVC

Kratica MVC označava strukturu web aplikacija i glasi „Model-View-Controller“. Ta kratica označava tri sloja bilo koje aplikacije koja koristi spomenutu strukturu, a ta tri sloja su:

- Model (eng. bussiness layer – sloj u kojem se npr. pristupa bazi podataka)
- View (eng. display layer – sloj za prikaz podataka korisniku)
- Controller (eng. input controll – sloj za manipulaciju podacima iz poslovnog sloja (eng. bussiness layer) i manipulaciju prikazanog sadržaja korisniku)

2.2.2. Web API

Web API označava okruženje koje se koristi za izradu RESTful aplikacija. Primjena ovog okruženja je vidljiva u manipulaciji HTTP zahtjeva koji se šalju iz Web API okruženja prema backend-u (najčešće je to server). Odgovor (eng. Response) koji dolazi sa servera je najčešće u XML-u pa je i lagano njime manipulirati. Tip odgovora je sasvim nebitan jer Web API omogućava manipulaciju bilo kakvog tipa odgovora od strane backend-a.

2.2.3. Entity Framework

Entity Framework je skup tehnologija koje se koriste u sklopu ADO .NET koji podržava razvoj podacima orijentiranih aplikacija. Prilikom izrade web aplikacija programeri se suočavaju sa problemom modeliranja entiteta, veza između njih te logike problema koji bi ta aplikacija trebala rješavati.

Entity Framework omogućava programerima da rade sa podacima bez da se brinu o serveru (bazi podataka) i sadržaju koji je pohranjen na serveru. Primjenom Entity Framework okruženja programeri mogu raditi sa podacima na višoj razini abstrakcije kada manipuliraju podacima te mogu kreirati nove i upravljati postojećim podacima orijentiranim aplikacijama pisanjem manje količine programskog koda nego u uobičajenim aplikacijama za manipulaciju podacima. Bitna činjenica je da je Entity Framework okruženje dio .NET okruženja pa se može koristiti na bilo kojem računalu na kojem je instaliran .NET.

Postoje dva pristupa realizaciji aplikacija, a to su:

- Code First pristup
- Database First pristup

2.2.4. Code First pristup realizaciji projekta

Code first pristup realizaciji projekta je dio Entity Framework okruženja koji je baziran na izradi klasa koje predstavljaju modele na temelju kojih će biti kreirana baza podataka. Kreirana baza podataka će sadržavati sve tablice (modele) koji su kreirani u projektu te sve atribute koji se nalaze u pojedinoj klasi. Također potrebno je u klasama koje predstavljaju modele definirati i vezu sa ostalim modelima ukoliko postoje. Ukoliko određena klasa treba imati pristup većoj količini podataka druge klase koriste se kolekcije (eng. Collection) objekata, a ukoliko je potrebno pristupati samo određenom podatku iz klase onda je dovoljno u klasi navesti samo objekt klase kao atribut.

2.2.5. Database First pristup realizaciji projekta

Za razliku od Code First pristupa, Database First pristup se temelji na izradi baze podataka iz koje se generiraju modeli sa odgovarajućim atributima. Dovoljno je samo napraviti

bazu podataka i u njoj definirati veze između tablica, a Entity Framework okruženje će se pobrinuti za kreiranje modela i svih atributa. Svaki model će kao i kod Code First pristupa predstavljati jedna klasa koja će imati iste atribute kao i tablica sa istim imenom u bazi podataka. U ovom diplomskom radu je korišten Database First pristup realizacije te će biti detaljnije objašnjen u razradi same aplikacije.

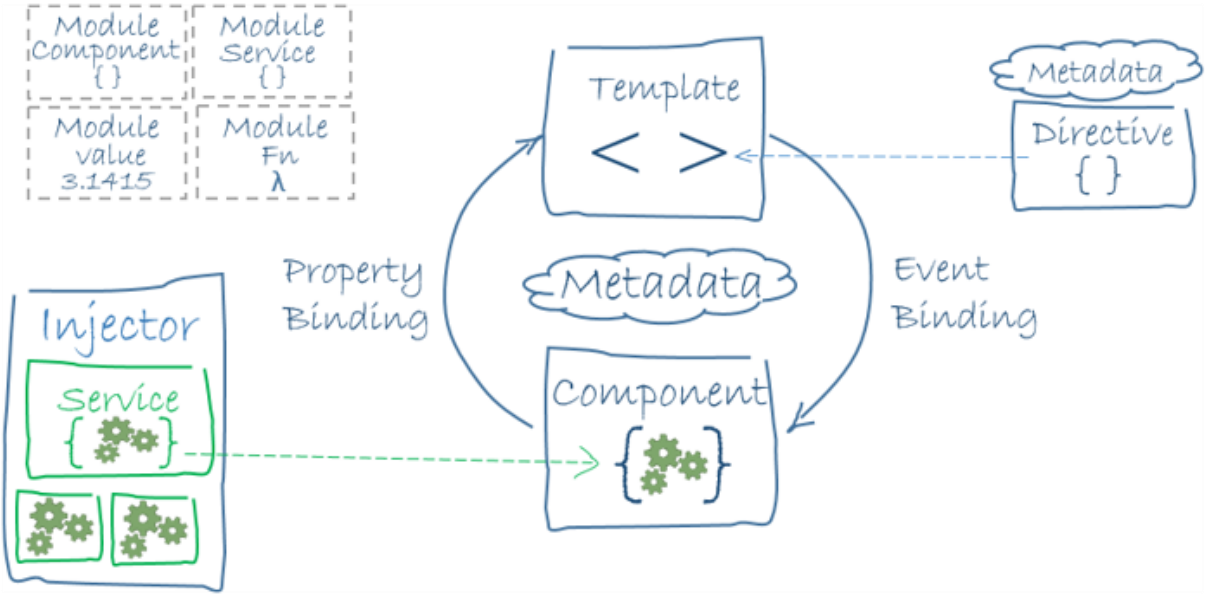
2.2.6. ASP.NET Identity

ASP.NET Identity sustav je dio ASP.NET okruženja koji se koristi za kreiranje jednostavnog sustava za upravljanje korisnicima unutar sustava određene aplikacije bilo kojeg tipa unutar ASP.NET okruženja. Identity kreira sve potrebne tablice za korisnike koje sadrže, između ostalog, osnovne podatke koje korisnik unosi prilikom registracije, a to su e-mail adresa i lozinka za korisnički račun određenog korisnika koji se želi registrirati. Da bi Identity sustav znao gdje treba kreirati korisničke tablice potrebno je promijeniti određene postavke unutar koda u aplikaciji o kojima će biti riječ u poglavlju 2.2.1.

2.3. Angular

Angular je okruženje koje se koristi za izradu klijentskih aplikacija u HTML-u i JavaScript okruženju ili jeziku kao što je TypeScript koji je podržan od strane JavaScript okruženja.

Način na koji funkcioniraju aplikacije koje koriste Angular je da se izrađuju HTML predlošci koji se pridjeljuju Component klasama. Nakon toga se sama logika aplikacije realizira u Services klasama te se na samom kraju servisi i komponente grupiraju i povezuju u modulima. Svi moduli moraju biti navedeni u korijenskom modulu (eng. root module) koji je zadužen za prikaz cjelokupnog sadržaja aplikacije u web pregledniku korisnika.



Slika 2.1.: Grafički prikaz međudjelovanja dijelova Angular okruženja

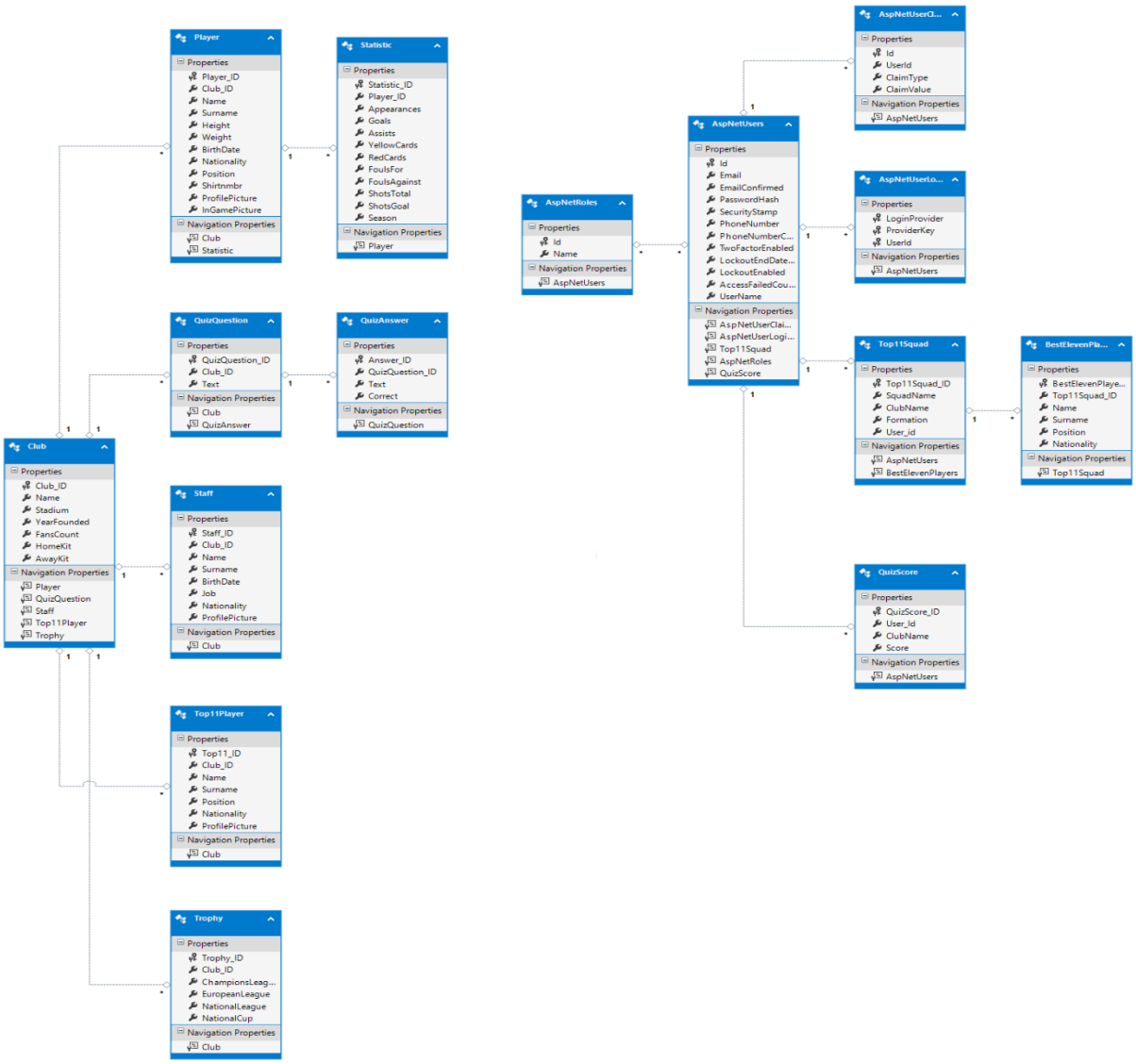
Kao što je vidljivo komponenta i predložak su usko povezani jer predložak prikazuje elemente i funkcije koje su kreirane unutar komponente. Elementi predloška se mogu povezati sa elementima komponente na tri načina, a to su: vitičaste zagrade ({{element}}), uglate zagrade ([atribut]="element") i ključna riječ bind ispred naziva atributa neke oznake u predlošku. Servisi su također sastavni dio angular aplikacija. U njima se nalaze funkcije koje bi se trebale pozivati na više lokacija te kako se nebi jedan te isti kod ponavljao više puta kreirana je funkcija unutar servise koja se samo poziva kada nam je potrebna. Također, unutar servisa se nalaze svi pozivi prema serveru (backend dijelu aplikacije). Direktive su povezane sa predlošcima i koriste se u predlošcima za razne svrhe kao npr. u tablicama kada je potrebno podatke iz liste prikazati u određenoj tablici (*ngFor). Ostale direktive koje se često koriste su *ngIf, [hidden] i druge. Bitno je i spomenuti da je moguće kreirati i svoje vlastite direktive za manipulaciju elementima unutar predloška pomoću ključne riječi Directive.

3. Izrada aplikacije

U ovom dijelu dokumentacije biti će objašnjena cjelokupna struktura aplikacije od samih početaka kreiranja pa sve do konačnih rezultata. Na početku je bilo potrebno kreirati bazu podataka pa i prva točka ovog dijela dokumentacije i govori o tome.

3.1. Izrada baze podataka

Prije početka razvoja aplikacije bilo je potrebno kreirati bazu podataka. Struktura baze podataka je prvo bila kreirana na papiru, a nakon određenih preinaka i na računalu korištenjem softvera „Star UML“. Tijekom razvoja projekta dolazilo je do određenih promjena u bazi podataka, a konačni oblik baze podataka moguće je vidjeti na slici dolje. Za kreiranje baze podataka koristio se softver „SQL Server Management Studio“.



Slika 3.1.: Shematski prikaz baze podataka

Najbitnije tablice sa pripadajućim atributima koje se nalaze u bazi su:

Club – identifikacijski niz znakova kluba (Club_ID), naziv kluba, naziv stadiona, godina osnutka, broj navijača, dres za domaće utakmice i dres za gostujuće utakmice.

Player – identifikacijski niz znakova za igrača i klub koji predstavlja (Player_ID i Club_ID), ime igrača, prezime igrača, visinu, težinu, datum rođenja, nacionalnost, pozicija na kojoj igra, broj na dresu, slika profila i dinamička slika (slika tijekom igranja ili za vrijeme slobodnog vremena) igrača.

Staff – identifikacijski niz znakova za člana osoblja i kluba koji predstavlja (Staff_ID i Club_ID), ime člana osoblja, prezime člana osoblja, datum rođenja člana osoblja, posao koji obavlja određeni član osoblja, nacionalnost člana osoblja i slika profila člana osoblja.

Trophy – identifikacijski niz znakova za niz trofeja i klub na koji se odnose (Trophy_ID i Club_ID), broj osvojenih „Champions League“ naslova, broj osvojenih „European League“ naslova, broj osvojenih naslova u državnom prvenstvu i broj osvojenih naslova u državnom kupu.

Statistic – identifikacijski niz znakova za statistički unos i za igrača na kojeg se taj unos odnosi (Statistic_ID i Player_ID), broj nastupa igrača (kada je igrač igrao ili od samog početka utakmice ili je ušao u igru kao zamjena za igrača koji je bio igrao od početka utakmice), broj postignutih zgoditaka u jednoj sezoni, broj asistencija u jednoj sezoni, broj žutih kartona u jednoj sezoni, broj crvenih kartona u jednoj sezoni, ukupni broj pretrpljenih prekršaja u jednoj sezoni, ukupni broj napravljenih prekršaja u jednoj sezoni, ukupni broj udaraca prema golu u jednoj sezoni, ukupni broj udaraca u okvir gola u jednoj sezoni te sezona na koju se unos odnosi.

Top11Player – identifikacijski niz znakova za nogometnu legendu i klub u kojem je određeni igrač postao legenda (Top11_ID i Club_ID), ime legende, prezime legende, pozicija na kojoj je igrala legenda, nacionalnost legende i slika profila legende.

QuizQuestion – identifikacijski niz znakova za pitanje u kvizu i klub kojem to pitanje u kvizu pripada (QuizQuestion_ID i Club_ID) te tekst pitanja na koje se traži odgovor.

QuizAnswer – identifikacijski niz znakova za odgovor na pitanje u kvizu te za pitanje kojem određeni odgovor pripada (Answer_ID i QuizQuestion_ID), tekst koji predstavlja odgovor i logička oznaka da li je odgovor točan ili netočan.

AspNetUsers – identifikacijski niz znakova za korisnika koji je registriran u sustavu (User_Id), e-mail adresa kojom se korisnik registrirao, lozinka koju je korisnik naveo prilikom registracije te ju koristi prilikom svake prijave u sustav i korisničko ime koje je korisnik naveo prilikom registracije (ostali atributi se ne koriste unutar aplikacije).

Top11Squad – identifikacijski niz znakova za momčad koju je kreirao korisnik te za korisnika koji je kreirao tu momčad (Top11Squad_ID i User_Id), naziv momčadi, naziv kluba od čijih igrača i legendi je sastavljena momčad i formacija koju je korisnik odabrao prilikom kreiranja momčadi.

BestElevenPlayers – identifikacijski niz znakova za člana momčadi i za momčad kojoj pripada određeni igrač (BestElevenPlayer_ID i Top11Squad_ID), ime člana momčadi, prezime člana momčadi, nacionalnost i pozicija člana momčadi.

QuizScore – identifikacijski niz znakova za rezultat u kvizu te za korisnika koji je taj rezultat ostvario (QuizScore_ID i User_Id), naziv kluba za koji su bila vezana pitanja u kvizu te ostvareni broj bodova u kvizu.

Tablice koje se nalaze u bazi, a nisu navedene kao bitne se ne koriste u aplikaciji ali su automatski kreirane korištenjem ASP.NET Identity sustava.

Da ne bi ostala bilo koja nedoumica ili nejasnoća vezana za ovo poglavlje potrebno je detaljno objasniti cjelokupnu strukturu baze podataka i međudjelovanje tablica koje se nalaze u bazi podataka. Cijeli proces kreće od glavne tablice u bazi, a to je tablica Club. Tablica Club je povezana kao što je vidljivo na slici sa tablicama Player, Staff, Trophy, Top11Player i QuizQuestion. Sa svakom od povezanih tablica, tablica Club ima vezu u omjeru 1:* (one to many) što znači da jedan klub može posjedovati 1 ili više članova u povezanim tablicama. Na isti način su povezane tablice Player i Statistic, QuizQuestion i QuizAnswer te Top11Squad i BestElevenPlayers. Također tablice koje je kreirao ASP.NET Identity sustav imaju vezu u istom omjeru kao već navedene tablice, točnije, sve tablice koje je Identity sustav kreirao su povezane sa tablicom AspNetUsers u omjeru 1:*.

3.2. Kreiranje i izrada potrebnih projekata

Prije samog početka izrade programskog dijela aplikacije bilo je potrebno kreirati projektnu arhitekturu te kreirati sve potrebne projekte. Sljedeća projektna arhitektura je konačna arhitektura aplikacije i sastoji se od sljedećih projekata:

- FootballFanApp.DAL
- FootballFanApp.DAL.Common
- FootballFanApp.DependencyResolver
- FootballFanApp.Model
- FootballFanApp.Model.Common
- FootballFanApp.Repository
- FootballFanApp.Repository.Common
- FootballFanApp.Service
- FootballFanApp.Service.Common
- FootballFanApp.MVC_WebAPI
- FootballFanApp.Token

Svi navedeni projekti u projektnoj strukturi međusobno komuniciraju pri čemu postoje određena pravila koja se poštivaju prilikom izrade projekta. Ta pravila se odnose na uporabu podataka određenih projekata iz projektne strukture o kojima će biti riječ u idućim poglavljima.

Da bi ova projektna arhitektura bila što jasnija potrebno je objasniti troslojnu arhitekturu web aplikacija. Ta arhitektura se sastoji od:

- Sloja za bazu podataka (Database layer)
- Sloja za obavljanje raznih operacija (Bussiness layer)
- Sloja za prezentiranje konačnih rezultata (Presentation layer)

Sloj za pristup bazi podataka (eng. DAL – Data Access Layer) se odnosi na sve modele koji su kreirani na temelju tablica iz baze podataka. Modeli sadrže atribute koji su jednaki onima iz tablice koja se nalazi u bazi podataka. Svi modeli (klase) imaju isti naziv kao i tablice iz baze podataka. Unutar tih modela moguće je vidjeti i vezu između tablica u bazi podataka. Kako u ovoj aplikaciji postoji veza u omjeru 1:* ta veza će se koristiti i kao primjer. Tablica Club može imati jedan ili više članova (objekata) tablice (klase) Player. Ta veza se u klasi Club prikazuje kao kolekcija objekata klase Player. Da bi veza bila potpuna u klasi Player se nalazi objekt klase Club.

Poslovni sloj (eng. Bussiness layer) je sloj koji obuhvaća u ovoj aplikaciji Model, Repository i Service projekte. Kao što i sam naziv kaže, u ovom sloju se obavlja mnoštvo operacija kojima se manipulira podacima i povezuje sloj za bazu podataka i prezentacijski sloj koji će biti kasnije objašnjen. Svrha ovog sloja u ovoj aplikaciji je da prima zahtjeve korisnika koji dolaze iz prezentacijskog sloja, da obavlja traženu operaciju te da rezultat operacije proslijedi sloju za bazu podataka. Nakon što operacija bude obavljena prosljeđuje se odgovor iz sloja za bazu podataka, preko ovog sloja sve do sloja za prezentiranje.

Sloj za prezentiranje rezultata (eng. Presentation layer) je sloj koji prezentira tražene podatke korisniku korištenjem WebAPI-ja. To znači da kada korisnik klikne na određenu lokaciju u web aplikaciji dolazi do kreiranja zahtjeva za određenim podacima ili za određenom operacijom

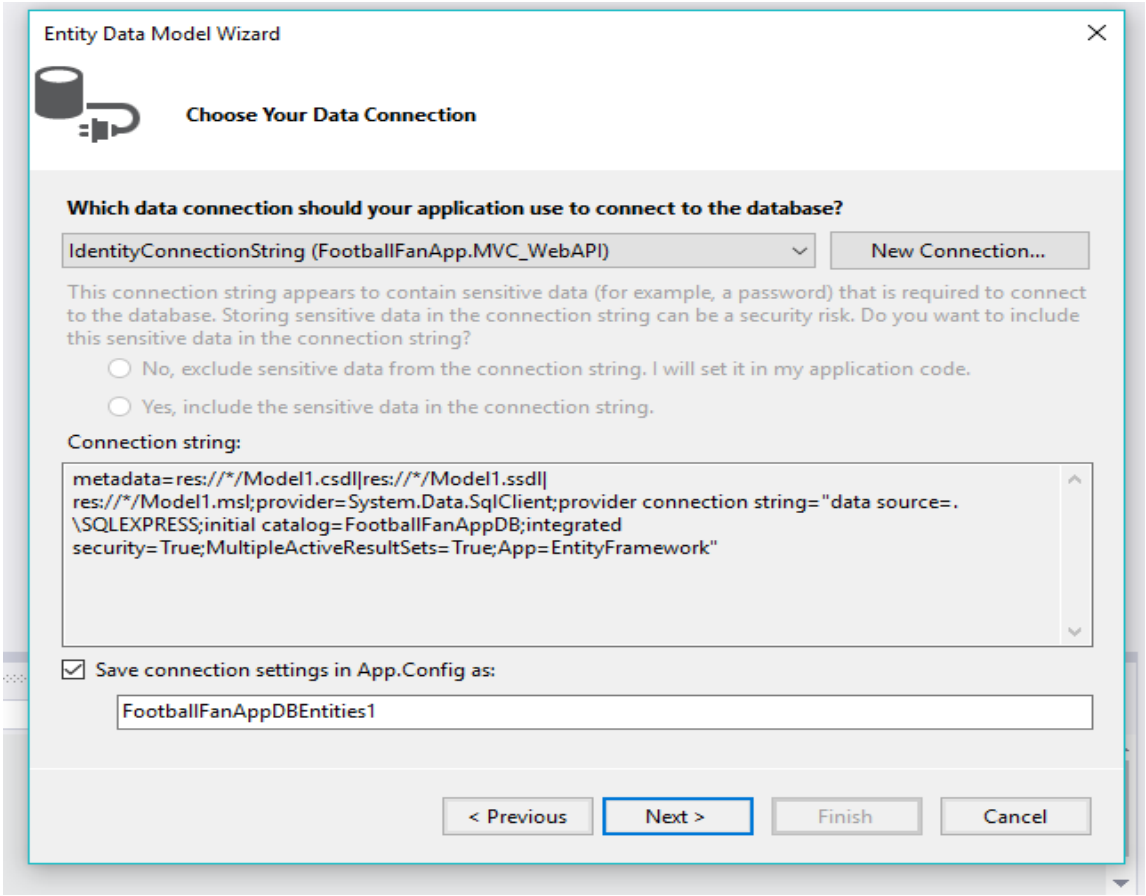
koja se nalazi u pozadini aplikacije (eng. backend). Taj zahtjev se prosljeđuje na poslovni sloj gdje se obrađuje. Bitno je napomenuti da sloj za prezentiranje ne zna točno šta se nalazi u pozadini nego su mu samo poznati zahtjevi koje je moguće koristiti i dobiti određeni odgovor.

Prije podrobnijeg objašnjavanja projekata iz projektne arhitekture potrebno je objasniti glavni razlog korištenja troslojne arhitekture u projektima. Razlog leži u fleksibilnosti programskog koda. Recimo da je potrebno na određenom projektu izvršiti izmjene u bazi podataka, a u projektu nije korištena troslojna arhitektura. Prilikom izmjenjivanja programer bi trebao promijeniti redom svaki dio koda koji je povezan sa starim dizajnom baze podataka što bi bio zahtijevan posao. Ukoliko programer koristi troslojnu arhitekturu u aplikaciji morati će samo izmijeniti sloj za pristup bazi podataka dok će ostali slojevi ostati netaknuti jer ni jedan od slojeva nije povezan direktno na modele iz baze podataka. Također bitno je navesti da troslojna arhitektura omogućava jednostavan rad više osoba na istom projektu jer dok jedna osoba radi bazu podataka, ostali članovi tima mogu raditi na poslovnom sloju ili na prezentacijskom sloju pri čemu neće oštetiti rad na bazi podataka.

Svi ovi razlozi dovode do konačnog zaključka, a to je da se pozadina aplikacije koja koristi ovu arhitekturu može koristiti u bilo kojem okruženju, neovisno o prezentacijskom sloju i tehnologiji koja se koristi za prikazivanje podataka u web pregledniku.

3.2.1. DAL – Data Access Layer

U prethodnom poglavlju je DAL ukratko objašnjen ali da bi u potpunosti bilo jasno o čemu je riječ u tom sloju potrebno ga je detaljno objasniti. Već je spomenuto da se sastoji od klasa koje predstavljaju tablice unutar baze podataka, ali na koji način su zapravo te tablice povezane sa klasama u sloju?. Odgovor na to pitanje je datoteka sa .edmx ekstenzijom. Točnije to je Entity Data Model unutar kojeg se nalazi klasni dijagram koji predstavlja bazu podataka. Već je tijekom uvodnog dijela dokumentacije navedeno da će se u ovom dijelu koristiti Database First pristup realizaciji projekta te je tako i učinjeno.



Slika 3.2.: Prikaz izbornika za spajanje na bazu podataka

Na slici je moguće vidjeti izbornik kojim se odabere veza na bazu podataka te se na temelju odabrane veze kreira ključ za spajanje na bazu koji se uvrštava u konfiguraciju DAL projekta.

Nakon što je uspostavljena veza sa bazom podataka bilo je potrebno navesti naziv ključa za spajanje na bazu u Context klasi koja se generira zajedno sa klasama svih tablica iz baze podataka kako bi komunikacija između baze i sloja za pristup podacima iz baze podataka bila potpuna.

Tijekom razvoja aplikacije nailazilo se na problem dohvaćanja samo određenih podataka iz određene tablice. U tu svrhu su kreirane pomoćne klase koje su kao atribute imale samo one atribute koji su bili potrebni za dohvaćanje točno određenih podataka. Te klase nisu imale tablice u bazi podataka nego su njihovi objekti korišteni u poslovnom sloju pri čemu su

vrijednosti atributa tih objekata bile jednake odgovarajućim atributima iz odgovarajućih klasa čije se tablice nalaze u bazi podataka.

3.2.2. Token

Prije pojašnjavanja pojma „Token“ potrebno je objasniti autentikaciju sustava. Dakle, autentikacija je pojam koji označava proces određivanja identiteta neke osobe (subjekta) pri kojem subjekt daje određene podatke prema kojima sustav može utvrditi da li je taj subjekt stvarno taj kojim se predstavlja u tom trenutku. U ovoj aplikaciji provodi se autentikacija prilikom bilo kojeg zahtjeva koji dolazi od prezentacijskog sloja, a drugi primjer korištenja autentikacije je prilikom upisivanja lozinke na bankomatu.

Nakon što je u detalje razjašnjena autentikacija sustava može se pojasniti i pojam „Token“. Token je niz znakova koji se dodjeljuje svakom korisniku prilikom prijave u sustav. Na temelju tog niza znakova korisnik pristupa svim dostupnim podacima nakon prijave u sustav. Prilikom odabira određene funkcije u sustavu šalje se zahtjev prema poslovnom sloju koji zatim provjerava da li je taj niz znakova (token) ispravan (validan). Nakon provjere, ukoliko je token ispravan, korisniku su dostavljeni traženi podaci. U sklopu tokena je kreiran i standard JSON Web Token (RFC 7519) koji je dosegnuo veliku popularnost zbog svoje veličine. JSON Web Token standard se zbog svoje veličine može lagano prenositi kao niz znakova, kao atribut zaglavlja i unutar tijela zahtjeva za slanjem podataka.

Pojam „Token“ je objašnjen te je sada potrebno navesti razloge zbog kojih se isplati koristiti autentikaciju na temelju tokena. Postoje tri glavna razloga zbog kojih bi svaki sustav koji ima i malu količinu povjerljivih podataka trebao koristiti token, a to su:

- ✓ Tokeni su samoodrživi – svaki token zadrži u sebi niz znakova koji je dovoljan za autentikaciju i na taj način oslobađa server od spremanja podataka prilikom prijave te pamćenja sesija (ukoliko u jednom trenutku broj korisnika prijavljenih u sustavu iznosi preko 1000 to bi jako opterećivalo sustav).
- ✓ Tokeni se mogu generirati bilo gdje – kreiranje tokena se čak može odvijati i na drugom serveru jer su verifikacija tokena i kreiranje tokena totalno odvojeni jer jedina poveznica je niz znakova koji se predaje prilikom verifikacije, a verifikacija ne zna gdje je taj niz znakova kreiran. Također generiranje tokena se može odvijati i u drugoj kompaniji tj. druga kompanija može obavljati to za nas.
- ✓ Navođenje veoma preciznih dozvola – unutar tokena se mogu navesti sva dopuštenja i podaci kojima korisnik smije pristupiti.

Sada je još ostalo pojasniti strukturu JSON Web Tokena. JSON Web Token se sastoji od tri dijela. Ti dijelovi su:

- o zaglavlje
- o sadržaj
- o potpis

Zaglavlje je niz znakova šifriran Base64 sustavom nakon kojeg dolazi točka, a nakon točke slijedi dio za sadržaj. Zaglavlje se sastoji od meta podataka koji uključuju tip tokena i algoritma za kriptiranje koji se koristi za potpisivanje tokena. Nakon zaglavlja dolazi sadržaj, a sadržaj se

sastoji od niza zahtjeva koje token kodira. Dio za potpis se sastoji od niza znakova koji označavaju potpis (tajnu). Potpis je najvažniji dio tokena jer se prema njemu utvrđuje da li korisnik ima pravo pristupiti određenim podacima. Token nije kriptiran nego je potpisan, a to znači da se prilikom autentikacije provjerava točnost potpisa. Da bi se kojim slučajem Token kriptirao došlo bi do velikih problema jer bi tada sustav bio veoma ranjiv. Naime, danas postoji rijetko koji tip kriptiranja da se nije pojavila barem jedna osoba koja zna određeni tip kriptiranja dekriptirati. Zbog toga se potpisom zapravo rješava taj problem na vrlo efikasan način.

Prije završetka ovog poglavlja potrebno je navesti par savjeta vezanih za implementiranje JSON Web Tokena u aplikacijama:

- ✓ Token ostavite tajnim i na taj način i sigurnim – token treba biti poznat samo onim servisima kojima je zaista potreban.
- ✓ Sadržaj bez povjerljivih podataka – nikada ne treba uvrštavati osjetljive podatke (korisničko ime, lozinka, itd.) u sadržaj tokena.
- ✓ Postavite vijek trajanja tokena – nakon što se korisniku prilikom prijave u sustav dodijeli token on ostaje validan zauvijek, osim ako se ne postavi vrijeme trajanja tog tokena.
- ✓ Token samo u HTTP zahtjevima – nikada ne treba slati token zajedno sa zahtjevima koji nisu vezani uz HTTP protokol jer na taj način izlažemo token velikom riziku.
- ✓ Korištenje dodatnih provjera – postoji mogućnost dodavanja sekundarnog sustava za provjeru tokena kako bi se osiguralo da je token generiran unutar određenog sustava.

3.2.3. Dependency Resolver

Ovaj projekt u projektnoj arhitekturi sadrži skup ovisnosti koje se reguliraju korištenjem AutoMapper-a i Ninject-a. Točnije povezuju se klase i njihova sučelja eng. interface) i na taj način se stvara njihova međuovisnost korištenjem Ninject programske biblioteke, dok se AutoMapper biblioteka koristi kako bi se objekti klasa kojima se manipulira kroz slojeve aplikacije mogli povezivati.

Naime, u aplikaciji svaki sloj koristi drugi skup modela (klasa) tablica iz baze podataka. Poslovni sloj koristi modele iz FootballFanApp.Model projekta iz projektne arhitekture zajedno sa pripadajućim sučeljima iz FootballFanApp.Model.Common projekta, dok prezentacijski sloj koristi prezentacijske modele (eng. view model) koji se nalaze unutar Web API projekta iz projektne arhitekture.

3.2.4. Repository

Repository je dio projektne arhitekture koji se zasniva na Repository Pattern sustavu, a pripada poslovnom sloju aplikacije. Razlog korištenja Repository Pattern sustava je taj da se na taj način modeli baze podataka sakrivaju od viših razina aplikacije (poslovni sloj pristupa sadržaju ovog dijela projektne arhitekture, a ne nižem sloju tj. bazi podataka). Također bitno je napomenuti da ovakva arhitektura programskog koda olakšava testerima testiranje aplikacije kada je aplikacija završena i spremna za testiranje.

Ideja repository dijela projekta je da se kreira repozitorij za svaki model iz baze podataka koji će se koristiti u višim razinama projekta te da se kreira jedna klasa koja će sadržavati generičke metode za operacije koje će se provoditi u svim repozitorijima unutar projekta. Na taj način je programski kod dobro strukturiran jer za svaki entitet (model iz baze podataka) postoji repozitorij unutar kojih se nalaze metode koje se koriste za manipuliranje podacima iz baze podataka, a pozivaju se iz viših razina aplikacije. Još jedan od bitnih razloga zbog kojih se koristi Repository Pattern je što se programer osigurava da neće kreirati metode za entitete koje se neće uopće koristiti.

Razlog kreiranja generičkog repozitorija je taj što na taj način programer posjeduje jedinstvenu metodu za bilo koji entitet. Dakle, ukoliko i dođe do rekonstrukcije baze podataka i potrebno je kreirati još jednu ili nekoliko tablica te ih implementirati i u programskom kodu, programer ne mora pisati iste metode samo za te nove entitete nego samo poziva metode iz generičkog repozitorija. Još je bitno spomenuti da za svaki repozitorij postoji i sučelje kao što je navedeno ranije u ovoj dokumentaciji za modele u poslovnom sloju.

3.2.5. Service

Servisi (eng. service) su standardni dio ovakve arhitekture u aplikacijama i služe kao poveznica između repozitorija i kontrolera unutar web API dijela aplikacije. Unutar servisa se nalaze klase gdje postoji jedna klasa za svaki entitet iz baze podataka. Unutar svake klase se nalaze iste metode kao i u svakoj od klasa koje predstavljaju repozitorije, međutim, postoji razlika u sadržaju tih metoda. Naime, u klasama unutar Repository dijela aplikacije pozivale su se metode generičkog repozitorija, dok se u klasama koje predstavljaju servise u metodama koriste tj. pozivaju metode iz repozitorija kako bi se manipuliralo podacima.

3.2.6. Web API

Web API dio projektne arhitekture je projekt koji spada u prezentacijski sloj aplikacije. Unutar Web API dijela projektne arhitekture nalaze se kontroleri, HTML datoteke i ostali dijelovi koji međudjeluju kao cjelina i koriste se za prikazivanje konačnog rezultata korisniku prilikom interakcije korisnika i aplikacije u web pregledniku. Jedina poveznica Web API dijela aplikacije i poslovnog sloja je kontroler. Kontroler je dio prezentacijskog sloja koji je koristi kako bi se nakon poziva bilo kojeg zahtjeva od strane korisnika ti zahtjevi isporučili poslovnom sloju koji ih obrađuje i nakon toga vraća rezultat natrag kontroleru koji prosljeđuje dobiveni rezultat prema prikazu, a konačni rezultat je prikaz podataka korisniku. Unutar kontrolera se nalaze metode unutar kojih se pozivaju metode od pripadajućeg servisa, a povratni tip tih metoda je jednak podacima koji se dobiju kao rezultat iz metode pripadajućeg servisa. Bitno je objasniti i na koji način korisnički zahtjevi i kontroleri komuniciraju da bi korisnik dobio podatke koje zahtijeva. Svaki kontroler sadrži u programskom kodu prefiks putanje koja vodi do podataka za određeni entitet. Također svaka metoda unutar kontrolera sadrži putanju koja predstavlja točno određeni zahtjev unutar kontrolera. Dakle, kada korisnik zahtjeva određene podatke, korisnički zahtjev dolazi do kontrolera čiji prefiks putanje odgovara prefiksu putanje iz zahtjeva koji je korisnik uputio. Kada se pronađe odgovarajući kontroler, program pronalazi

metodu čija putanja odgovara putanji u zahtjevu. Nakon toga započinje obrada zahtjeva u kontroleru, nakon toga i u poslovnom sloju, a do tada prezentacijski sloj čeka na povratni tip tj. rezultate poslanog zahtjeva. Unutar ovog dijela projektne arhitekture nalaze se također i prezentacijski modeli za svaki od entiteta iz baze podataka te okruženje koje se koristi kao alat za prikaz rezultata dobivenih iz kontrolera. Nešto više o okruženju koje se koristilo kao alat za prikaz rezultata u ovoj aplikaciji biti će napisano u idućem poglavlju.

3.3. Izrada Angular dijela projekta u svrhu povezivanja prikaza i backend dijela aplikacije

U ovom poglavlju će biti detaljno objašnjeno Angular okruženje koje se primjenjivalo u ovoj aplikaciji u svrhu povezivanja prezentacije rezultata i kontrolera u Web API dijelu projektne arhitekture.

Arhitektura ovog dijela se sastoji od više mapa pri čemu svaka mapa predstavlja jedan ili više entiteta vezanih za bazu podataka. Unutar svake mape se nalaze još tri podmape, a te podmape su:

- komponente
- servisi
- prikazi

Podmapa u kojoj se nalaze komponente sadrži typescript datoteke čiji naziv se sastoji od opisa svrhe te datoteke i ključne riječi component pri čemu su ta dva dijela naziva datoteke odvojena točkom.

Podmapa u kojoj se nalaze servisi sadrži typescript datoteke unutar kojih se nalaze metode u kojima se pozivaju HTTP zahtjevi kojima se pozivaju metode iz kontrolera te metode koje su potrebne u komponentama za obradu podataka.

Podmapa u kojoj se nalaze prikazi sadrži datoteke sa html ekstenzijom. Svaka od tih html datoteka je usko povezana sa pripadajućom komponentom na način da oni podaci koji se nalaze u komponenti mogu se uvrstiti u html datoteku te prikazati korisniku u web pregledniku.

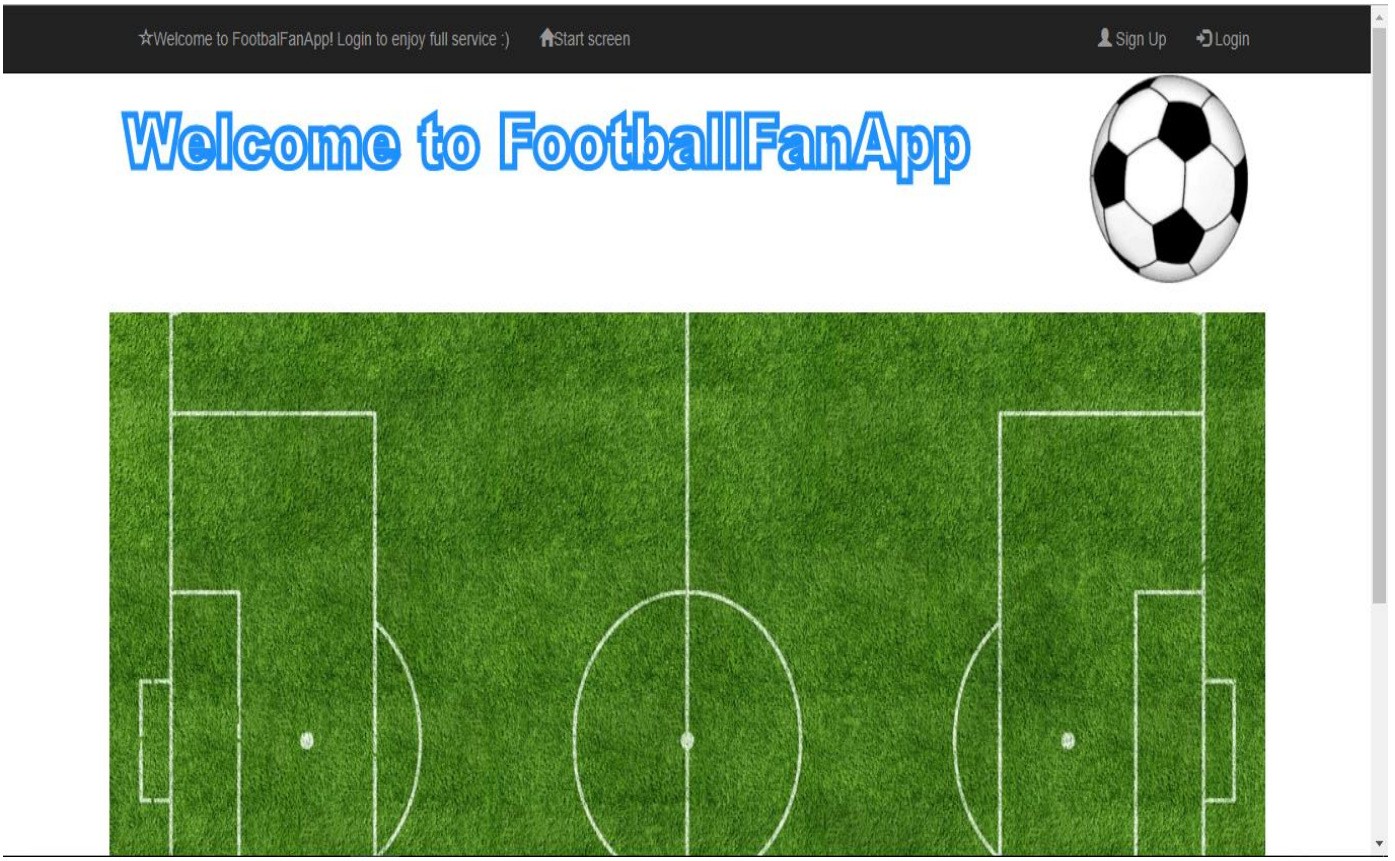
Tijekom izrade ovog dijela aplikacije bilo je potrebno koristiti i poneke eksterne biblioteke kako bi se olakšala implementacija određenih ideja vezanih za prezentiranje podataka korisnicima.

Službena arhitektura Angular okruženja nije primjenjivana u ovoj aplikaciji te bi postojeću arhitekturu trebalo preurediti u slučaju da je potrebno poštivati službenu arhitekturu.

3.4. Prikaz rezultata

Nakon što su ukratko objašnjeni svi dijelovi programskog koda, odnosno aplikacije, potrebno je prikazati rezultate koje bilo koji korisnik može vidjeti prilikom pokretanja i korištenja ove aplikacije.

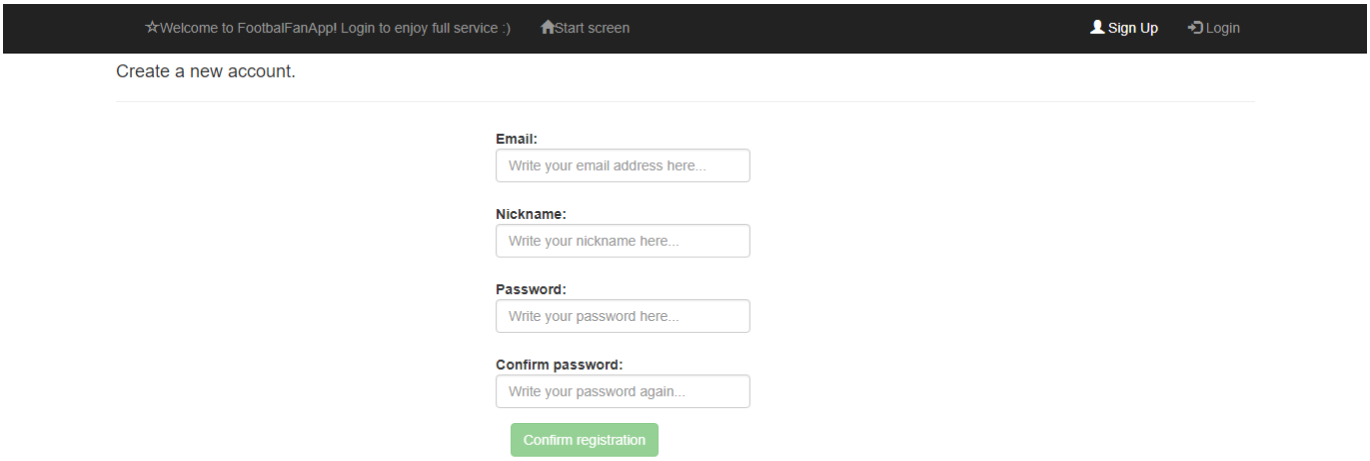
Nakon što korisnik pokrene aplikaciju u zadanom web pregledniku otvara se početna stranica aplikacije koja izgleda ovako:



Slika 3.3.: Naslovna stranica aplikacije

Kao što je i vidljivo na slici, početna stranica se sastoji od navigacijske trake pri vrhu na kojoj se nalaze poruka dobrodošlice, poveznica na početnu stranicu ukoliko korisnik otvori poveznice koje se nalaze s desne strane stranice pri vrhu te se želi vratiti natrag na početnu stranicu i poveznica za prijavu u sustav i za registraciju korisnika. Nogometna lopta koja je vidljiva na slici je datoteka sa .gif ekstenzijom i prikazuje animaciju kretanja lopte prilikom njezine rotacije. Još jedan od segmenata početne stranice je i slika nogometnog terena čija je svrha da, da zajedno sa nogometnom loptom, dočara korisnicima o kakvoj je aplikaciji riječ.

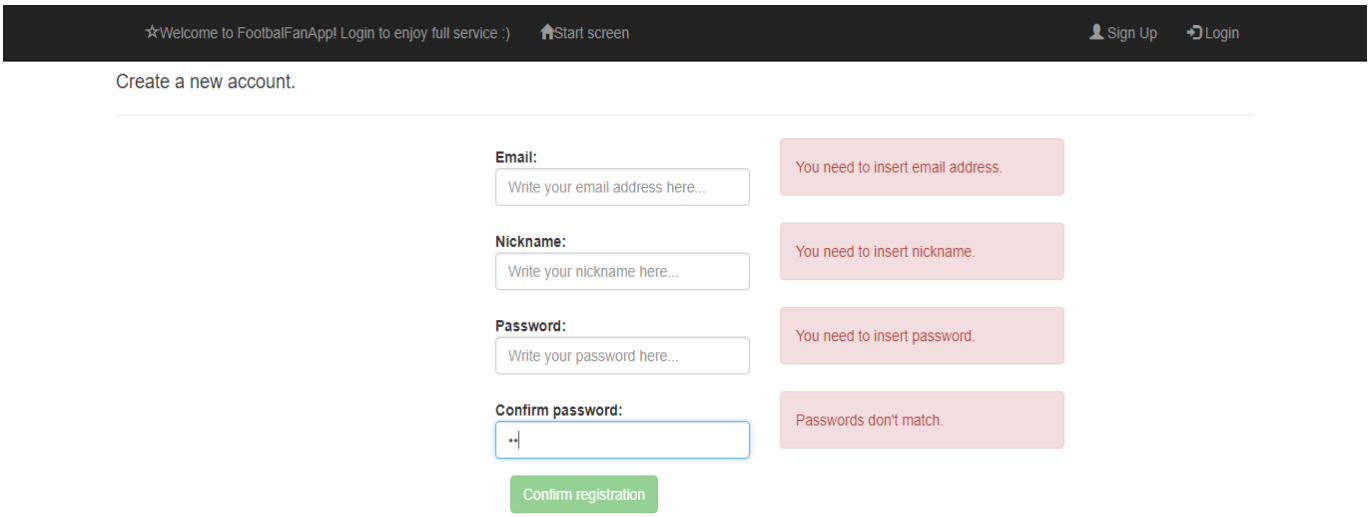
Prikaz forme za unos podataka prilikom registracije korisnika izgleda ovako:



Slika 3.4.: Prikaz stranice za registraciju

Kao što se može vidjeti, na slici se nalazi jednostavna forma za unos e-mail adrese, nadimka, lozinke te polje za ponovni unos (potvrdu) lozinke. Cijela forma je poravnata stilskim sredstvima kako bi prikaz registracijske forme bio što uredniji.

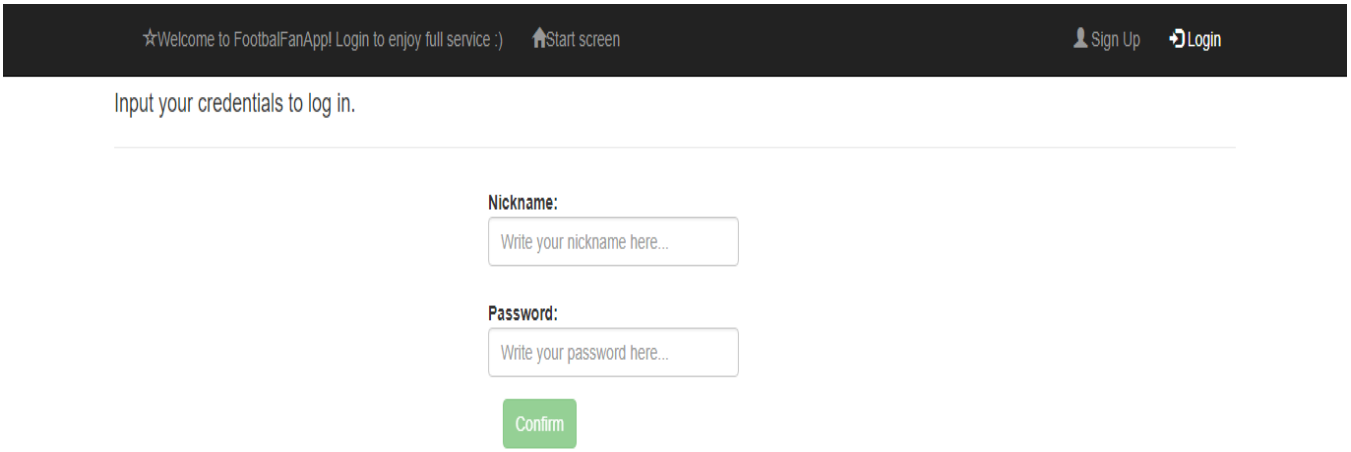
Ukoliko korisnik klikne na bilo koje od polja za unos iz forme i ne unese ni jedan podatak, a zatim i napusti polje za unos pojaviti će se poruka upozorenja. Prikaz registracijske forme u tom slučaju izgleda ovako:



Slika 3.5.: Prikaz stranice za registraciju - upozorenja

Ukoliko je korisnik zabunom kliknuo na poveznicu koja vodi do forme za registraciju može se vratiti na naslovnu stranicu klikom na poveznicu „*Start screen*“. Također ukoliko je korisnik već registriran, a zabunom je kliknuo na poveznicu za registracijsku formu može kliknuti na poveznicu „*Login*“ kako bi ga aplikacija preusmjerila na formu za prijavu u sustav.

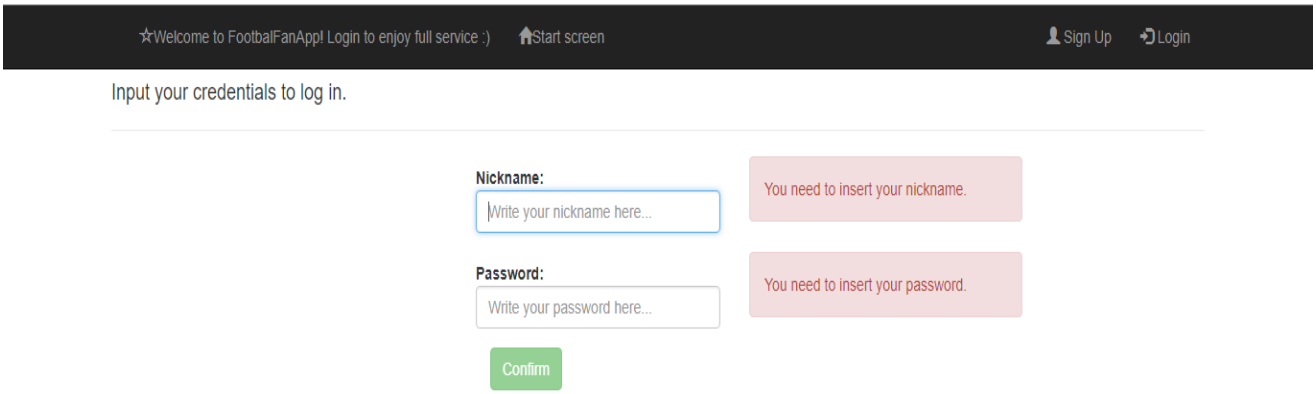
Stranica za prijavu u sustav izgleda ovako:



Slika 3.6.: Prikaz stranice za prijavu

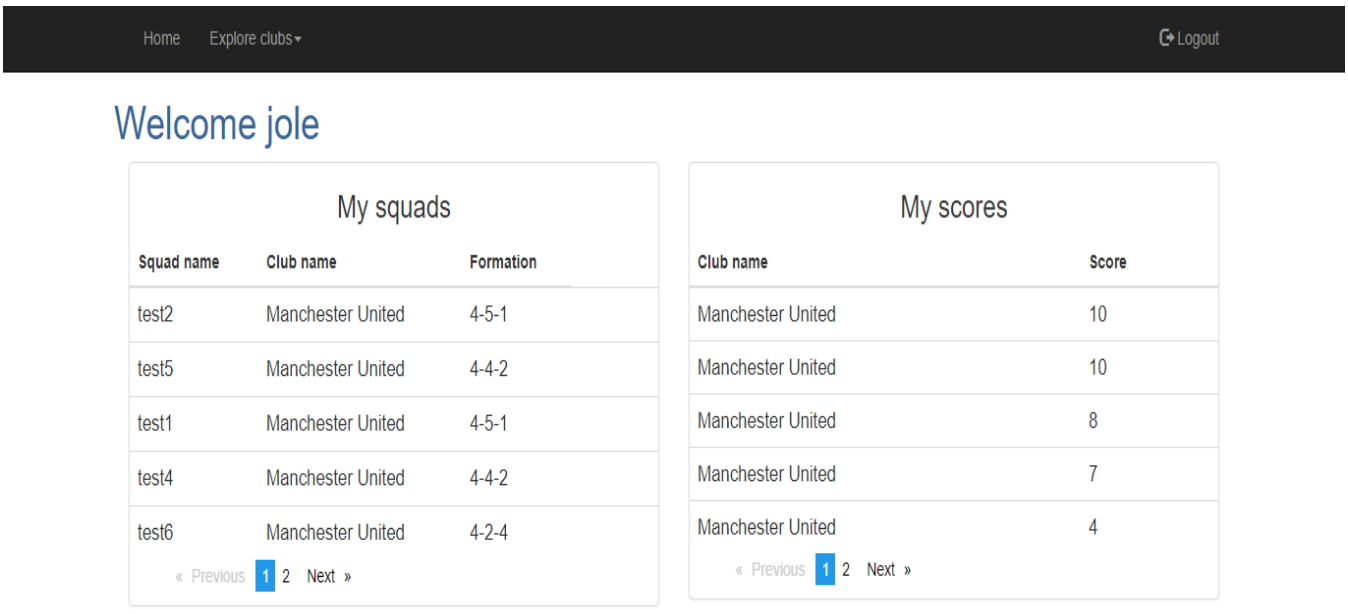
Dakle, sastoji se od forme za upis nadimka i lozinke koju je korisnik unio prilikom registracije u sustav. Kao i registracijska forma, forma za prijavu u sustav stilski je poravnata radi urednosti.

Kao i unutar registracijske forme, forma za prijavu u sustav ima određene provjere unešenih podataka te provjere da li su podaci uopće uneseni. Ukoliko korisnik pokuša unijeti određene podatke u formi, ali napusti polje za unos bez da je unio bilo kakav podatak aktivira se poruka koja upozorava korisnika da mora unijeti podatke unutar polja za unos. Prikaz sa porukom upozorenja izgleda ovako:



Slika 3.7.: Prikaz stranice za prijavu - upozorenja

Nakon što se korisnik prijavi u sustav otvara mu se korisnička naslovna stranica.

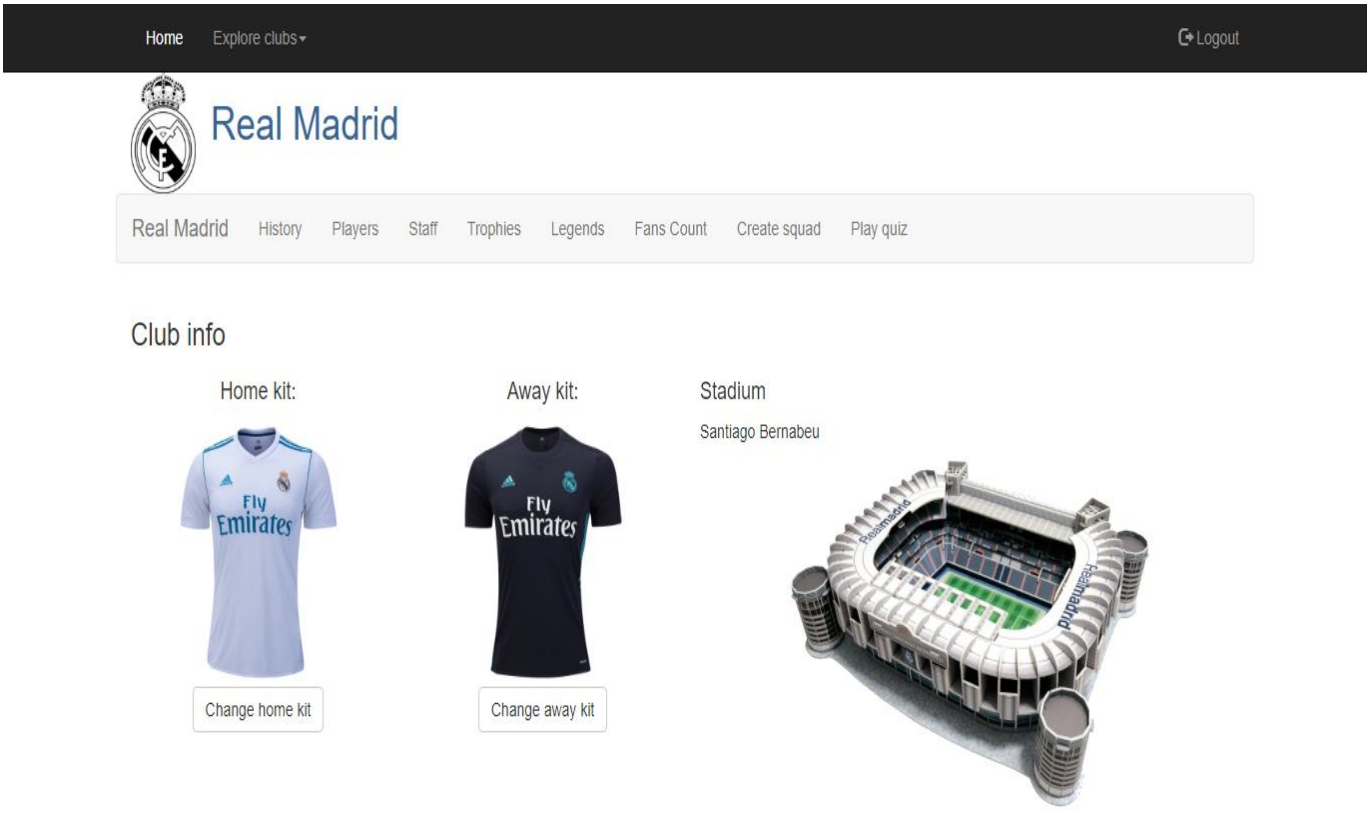


Slika 3.8.: Prikaz korisničke početne stranice

Na korisničkoj naslovnoj stranici nalazi se poruka dobrodošlice te dvije tablice, jedna prikazuje sve momčadi koje je korisnik kreirao dok druga pokazuje sve rezultate kviza koje je korisnik ostvario. Tablica koja prikazuje sve momčadi koje je korisnik kreirao sastoji se od tri stupca, a to su: naziv momčadi, naziv kluba i formacija koju je korisnik odabrao prilikom kreiranja momčadi. Tablica koja prikazuje sve rezultate kviza koje je korisnik ostvario sastoji se od dva stupca, a ta dva stupca su: naziv kluba i rezultat u kvizu. Obje tablice posjeduju i raspodjelu kreiranih momčadi, odnosno postignutih rezultata, po stranicama pri čemu je 5 momčadi, odnosno postignutih rezultata, vidljivo po stranici.

Također, bitno je napomenuti da je došlo do promjene u navigacijskoj traci na kojoj se sada nalazi poveznica koja vodi do korisničke naslovne stranice, padajući izbornik sa 5 klubova koji se smatraju najpopularnijima među nogometnim navijačima te poveznice koja prilikom klika korisnika na nju provodi odjavu korisnika iz sustava i preusmjerava korisnika na početnu stranicu aplikacije.

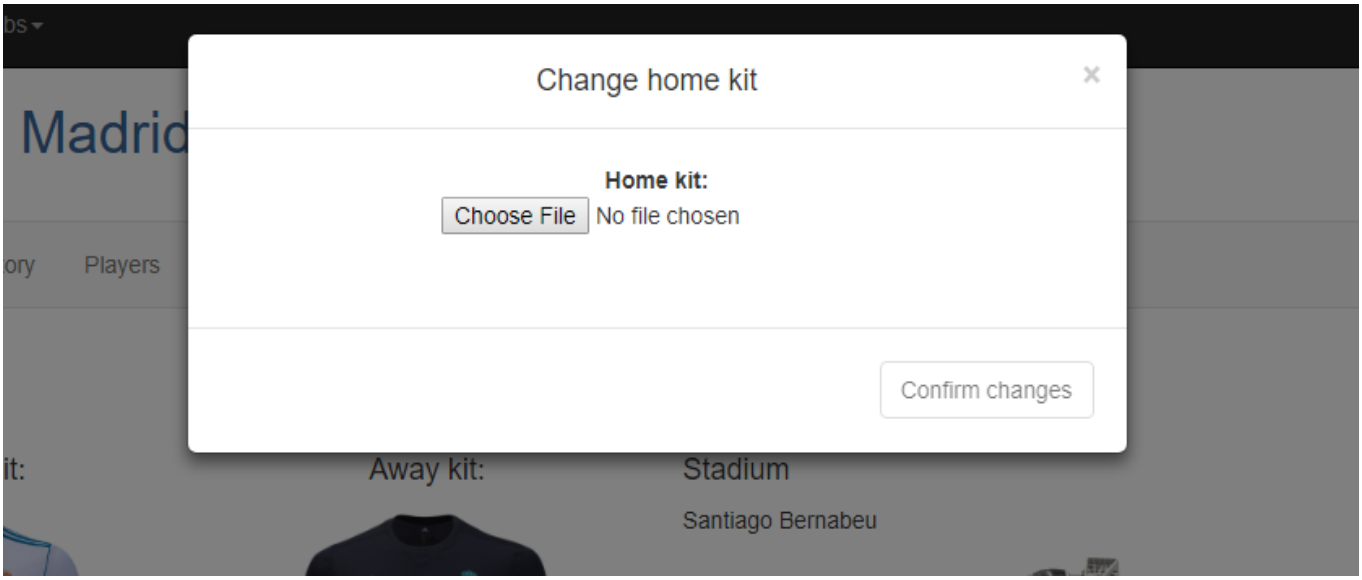
Nakon klika na padajući izbornik prikažu se nazivi 5 najpopularnijih klubova te korisnik može otvoriti klubsku naslovnu stranicu klikom na jedan od klubova unutar padajućeg izbornika.



Slika 3.9.: Prikaz klubske naslovne stranice

Klubska naslovna stranica sadrži veći broj poveznica koje vode na razne podgrupe vezane za odabrani klub. Te podgrupe su: povijest kluba, igrači, osoblje, riznica trofeja, legende kluba, broj obožavatelja, kreiranje momčadi te klubski kviz. Ostali sadržaj klubske naslovne stranice je skup informacija vezanih za klub, a to su garnitura dresova za domaće i gostujuće utakmice te transparentna slika stadiona. Također, moguće je promijeniti dresove kada istekne trenutna

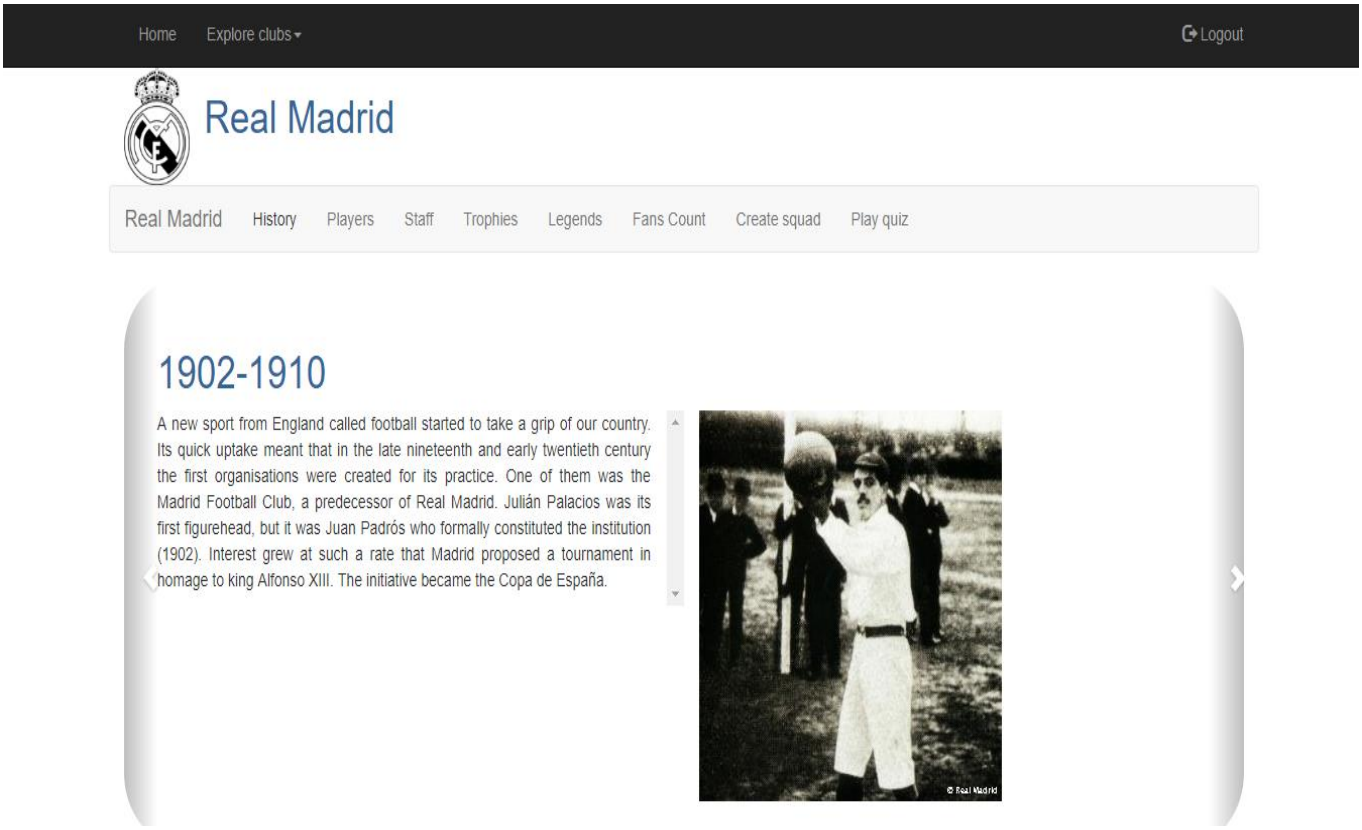
nogometna sezona i postanu poznate garniture dresova za iduću sezonu klikom na gumb „*Change home kit*“ ili klikom na gumb „*Change away kit*“.



Slika 3.10.: Prikaz skočnog prozora za izmjenu dresa

Skočni prozor za promjenu garniture dresova se sastoji od jednostavnog sustava za odabir datoteke .png ekstenzije, tipke za potvrdu promjena te ikone u desnom kutu skočnog prozora za zatvaranje otvorenog skočnog prozora.

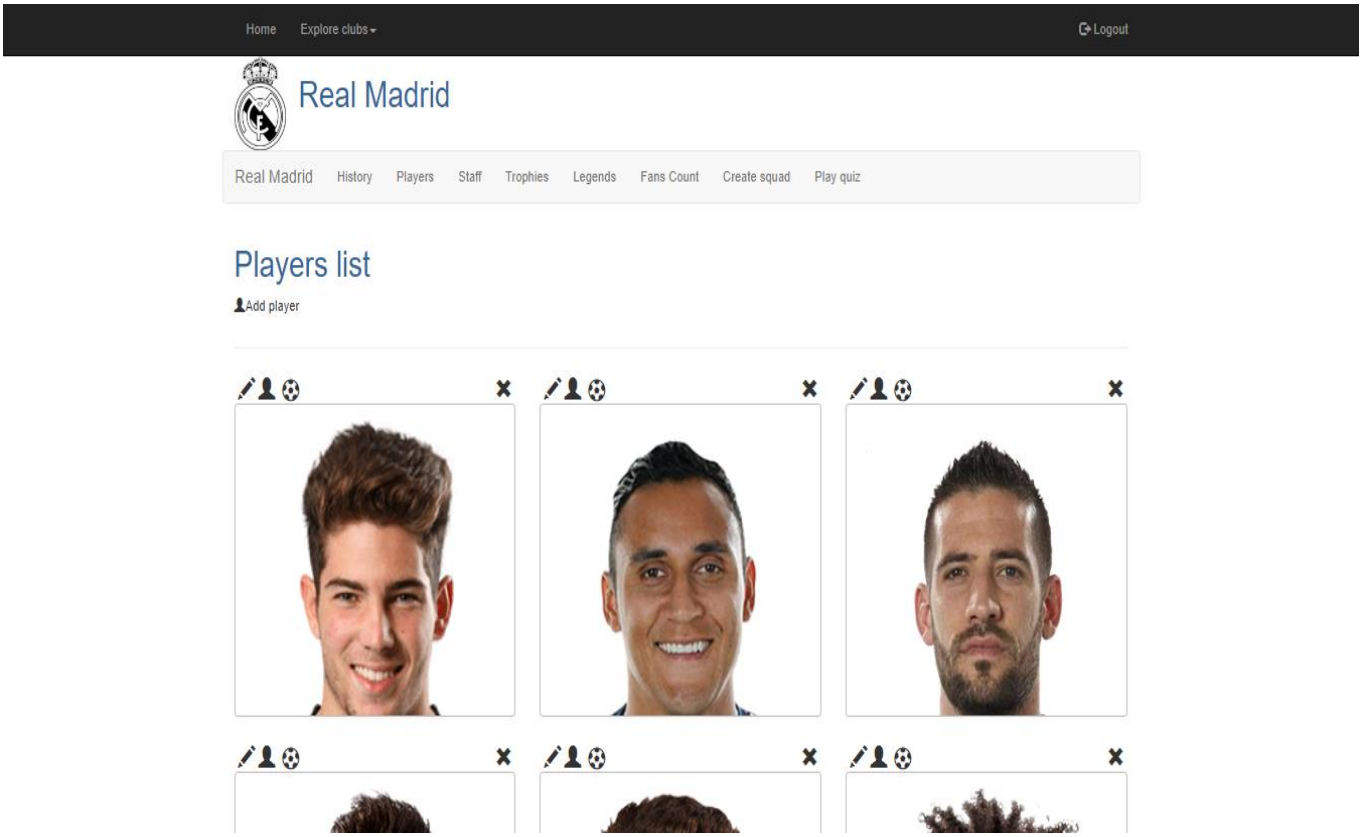
Sada je na redu prikaz povijesti kluba, a prikaz povijesti kluba je dizajniran na sljedeći način:



Slika 3.11.: Prikaz stranice za povijest kluba

Prikaz povijesti kluba sastoji se od karusela unutar kojeg se nalazi vremensko razdoblje, kratka povijest tijekom naznačenog vremenskog razdoblja te slika koja je obilježila to razdoblje.

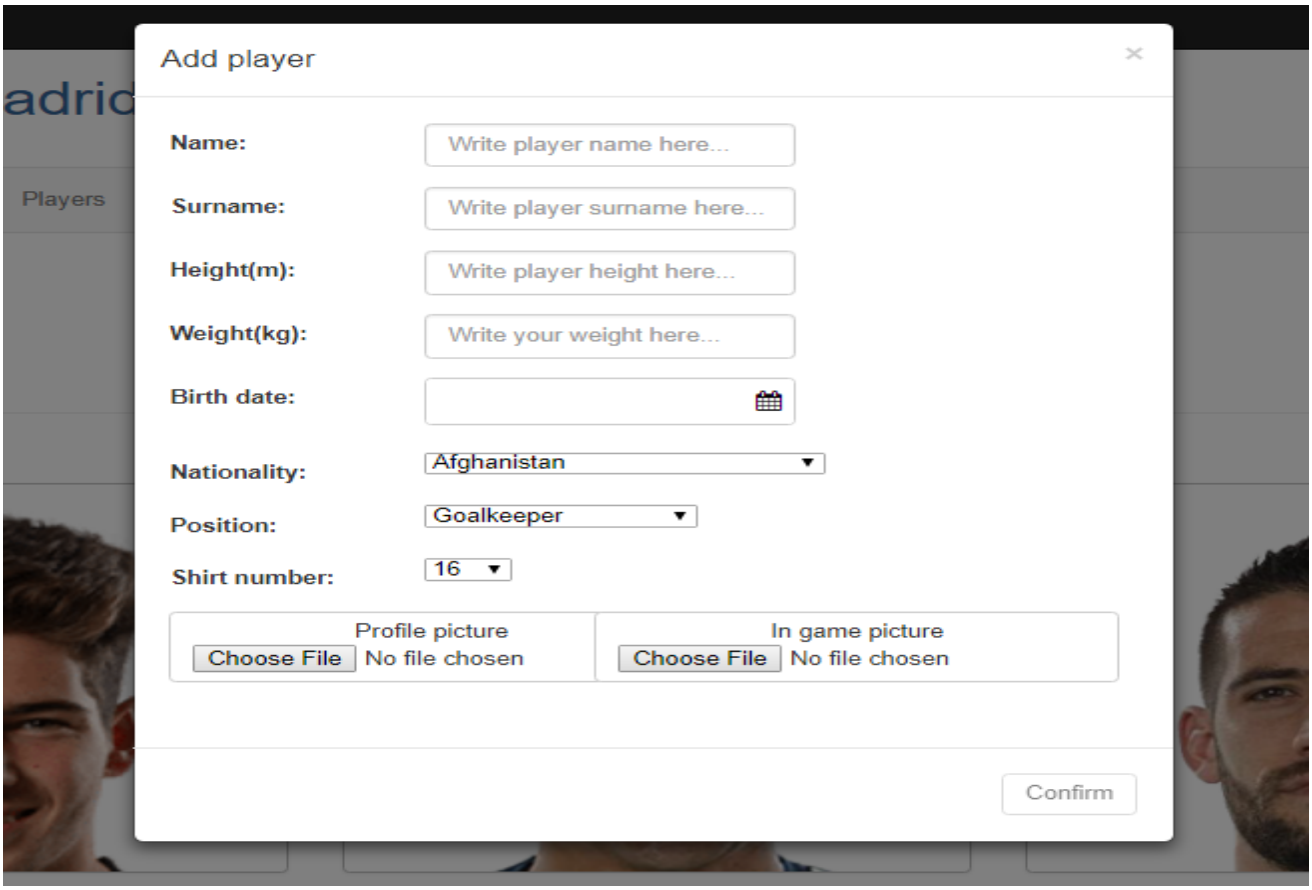
Idući prikaz koji je na redu je prikaz popisa igrača koji se trenutno nalaze u seniorskoj kategoriji kluba.



Slika 3.12.: Prikaz stranice za prikaz igrača

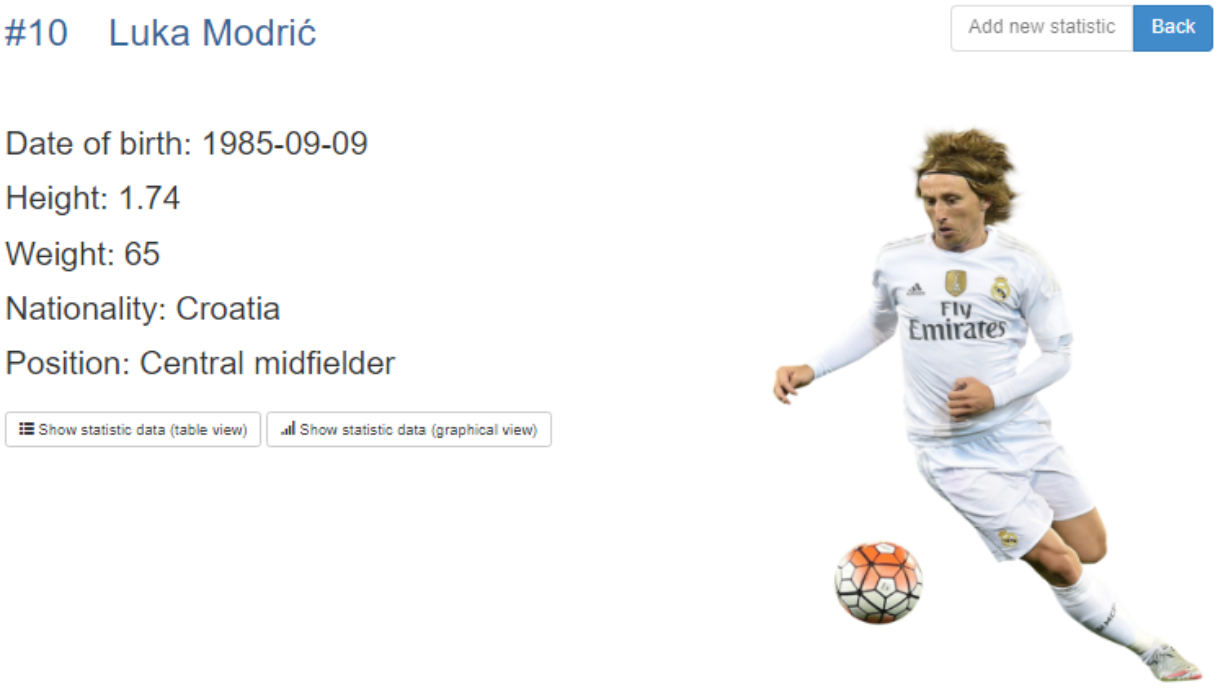
Na gore navedenom prikazu može se vidjeti lista igrača pri čemu za svakog igrača postoji ploča unutar koje se nalazi slika profila igrača te funkcijske ikone za uređivanje podataka o igraču, promjenu slike profila, promjenu slike u igri te za brisanje igrača. Također iznad same liste igrača nalazi se i poveznica koja, prilikom korisničkog klika na nju, otvara skočni prozor u kojem se nalazi forma za dodavanje novog igrača.

Unutar forme za dodavanje igrača nalaze se polja za unos svih podataka koje posjeduje određeni igrač tj. koje posjeduje tablica iz baze podataka koja se odnosi na igrača. Nakon što korisnik unese sve potrebne podatke mora potvrditi unos klikom na tipku na dnu skočnog prozora. Nakon što je proces dodavanja igrača dovršen, novi igrač se pridodaje na listu svih igrača i vidljiv je korisnicima.



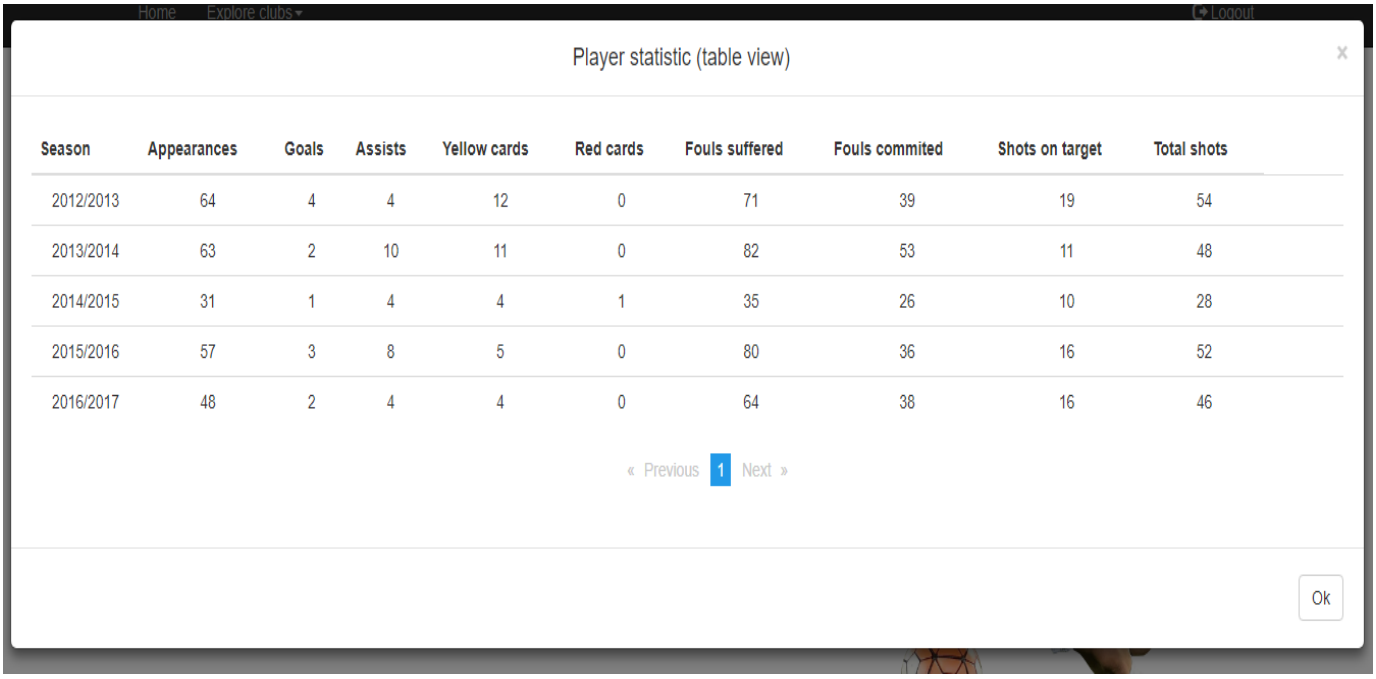
Slika 3.13.: Prikaz skočnog prozora za dodavanje igrača

Ukoliko korisnik klikne na bilo kojeg od igrača otvara se prikaz sa detaljnim informacijama o igraču kojeg je korisnik odabrao. Detaljni prikaz podataka odabranog igrača izgleda ovako:



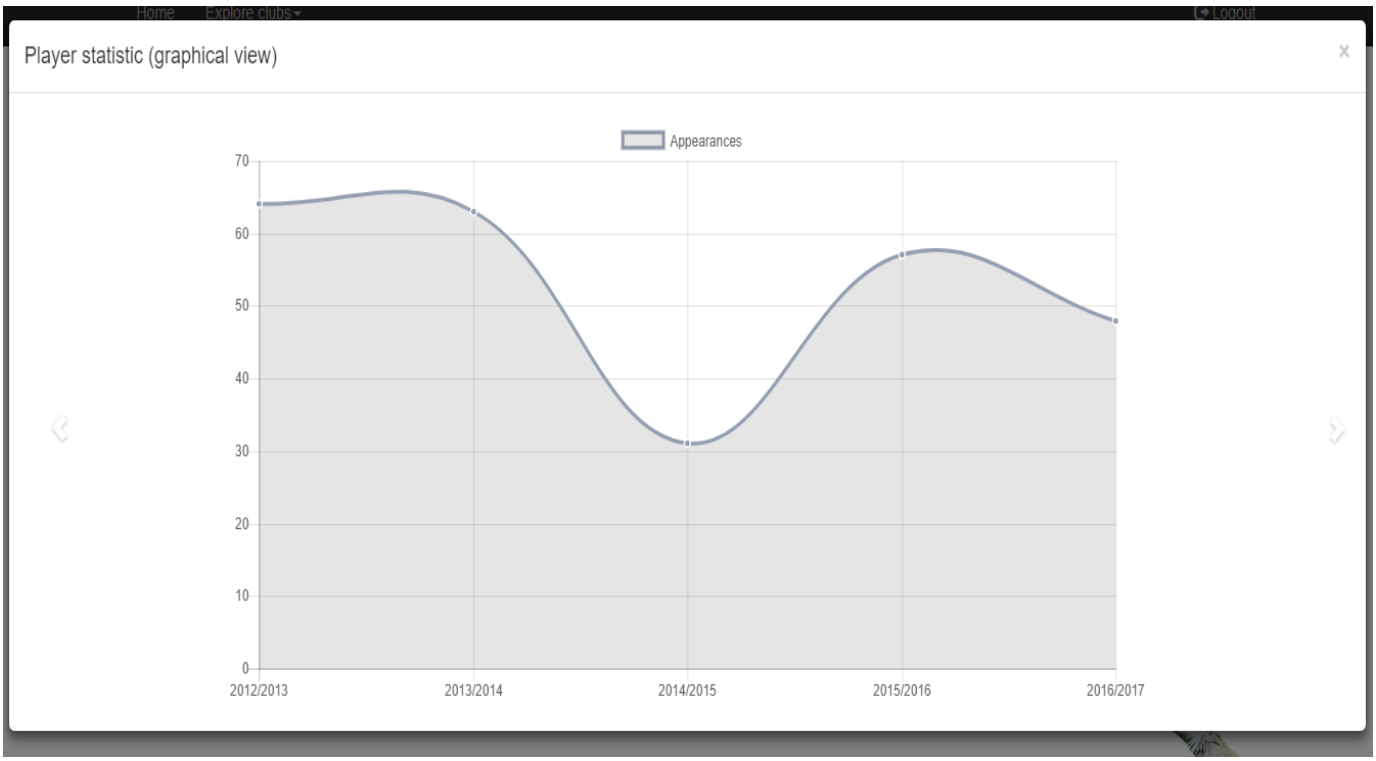
Slika 3.14.: Prikaz stranice za detaljni prikaz igrača

Ukoliko korisnik odabere gumb za prikaz statistike u tabličnom obliku onda će otvoriti sljedeći skočni prozor.



Slika 3.15.: Prikaz skočnog prozora za tablični prikaz statistike

Ukoliko korisnik odabere gumb za prikaz statistike u grafičkom obliku onda će se otvoriti sljedeći skočni prozor.

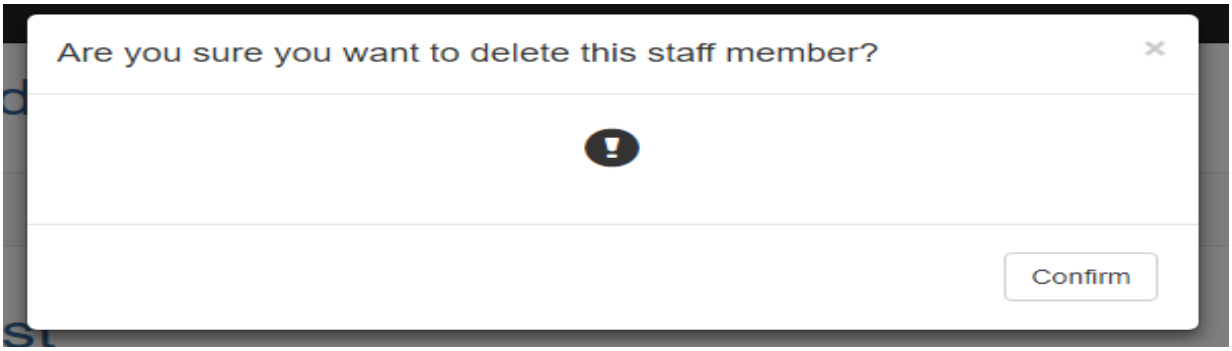


Slika 3.16.: Prikaz skočnog prozora za grafički prikaz statistike

Idući prikaz je prikaz osoblja kluba. Prikaz osoblja kluba je raspoređen na dva dijela kao što je moguće vidjeti na slici ispod. Prvi dio se odnosi na glavnog trenera u klubu pri čemu se mogu uređivati podaci o članu osoblja, mijenjati slika profila člana osoblja i moguće je obrisati određenog člana osoblja.

Slika 3.17.: Prikaz stranice za prikaz osoblja

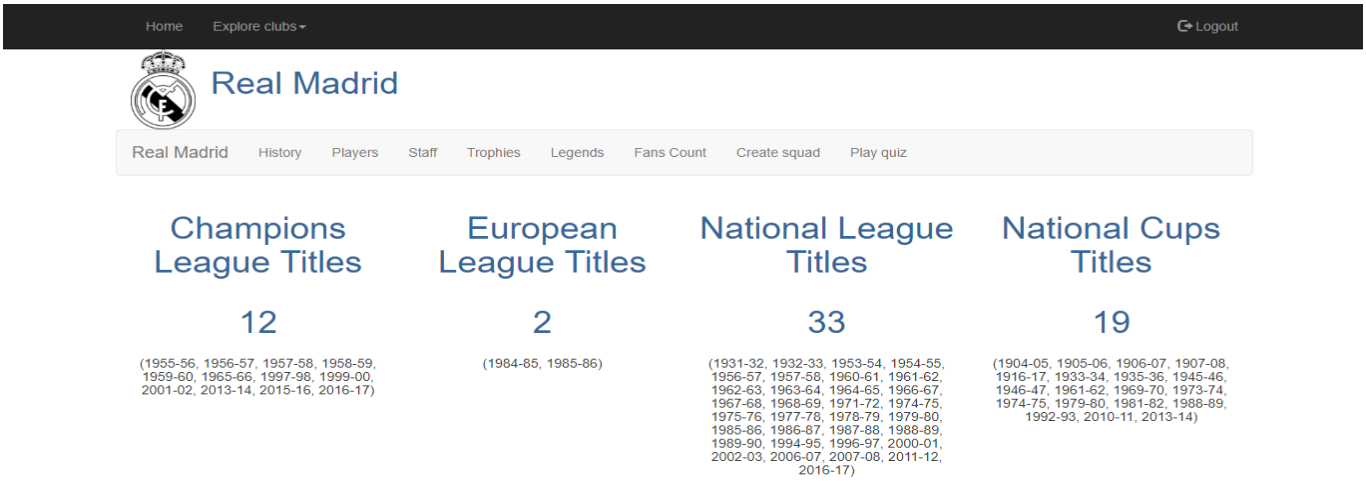
Ukoliko korisnik želi obrisati određenog člana osoblja otvara mu se sljedeći skočni prozor.



Slika 3.18.: Prikaz skočnog prozora za brisanje člana osoblja

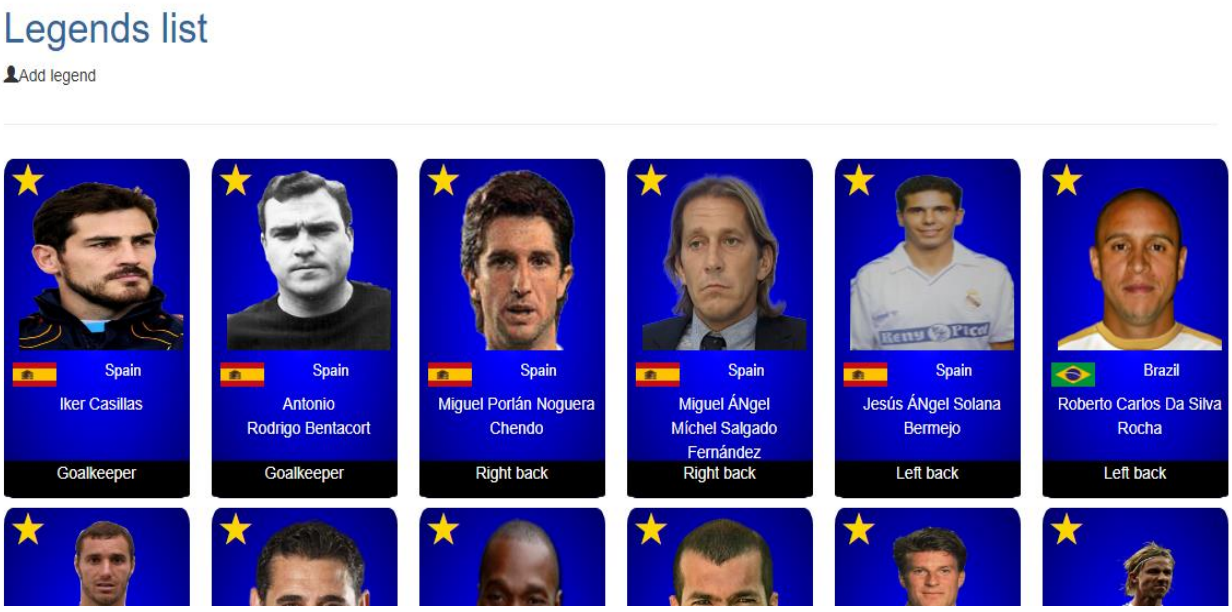
Također kao i kod prikaza igrača određenog kluba, i kod prikaza osoblja moguće je dodati novog člana osoblja. Međutim, ukoliko u klubu već postoji član osoblja sa funkcijom menadžera tada korisnik prilikom dodavanja novog člana osoblja ne može odabrati funkciju menadžera.

Idući prikaz je prikaz riznice trofeja. Riznica trofeja se sastoji od naziva natjecanja, broja osvojenih naslova određenog natjecanja i popisa godina kada je određeni klub osvojio naslov određenog natjecanja.



Slika 3.19.: Prikaz stranice za riznicu trofeja

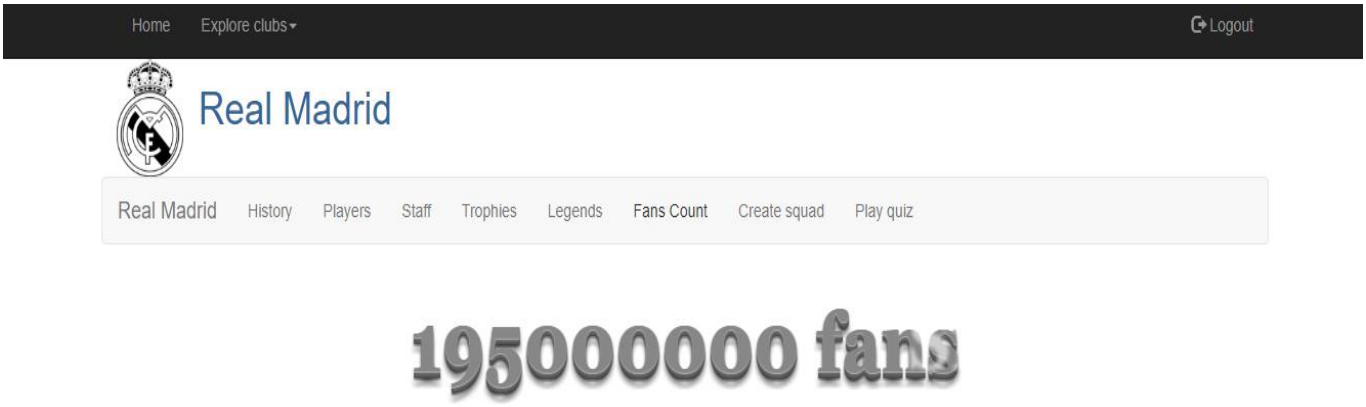
Poslije prikaza riznice trofeja slijedi prikaz legendi nogometnog kluba.



Slika 3.20.: Prikaz stranice za prikaz legendi kluba

Prikaz legendi se sastoji od jedne kartice za svaku legendu kluba. Unutar kartice se nalazi slika profila legende, ikona zvijezde kako bi se na taj način označila kartica i kako bi se dalo do znanja korisnicima da je riječ o legendi kluba. Ostali dijelovi kartice su državljanstvo legende, ime i prezime te pozicija koju je legenda kluba igrao za vrijeme svoje nogometne karijere na samom dnu kartice.

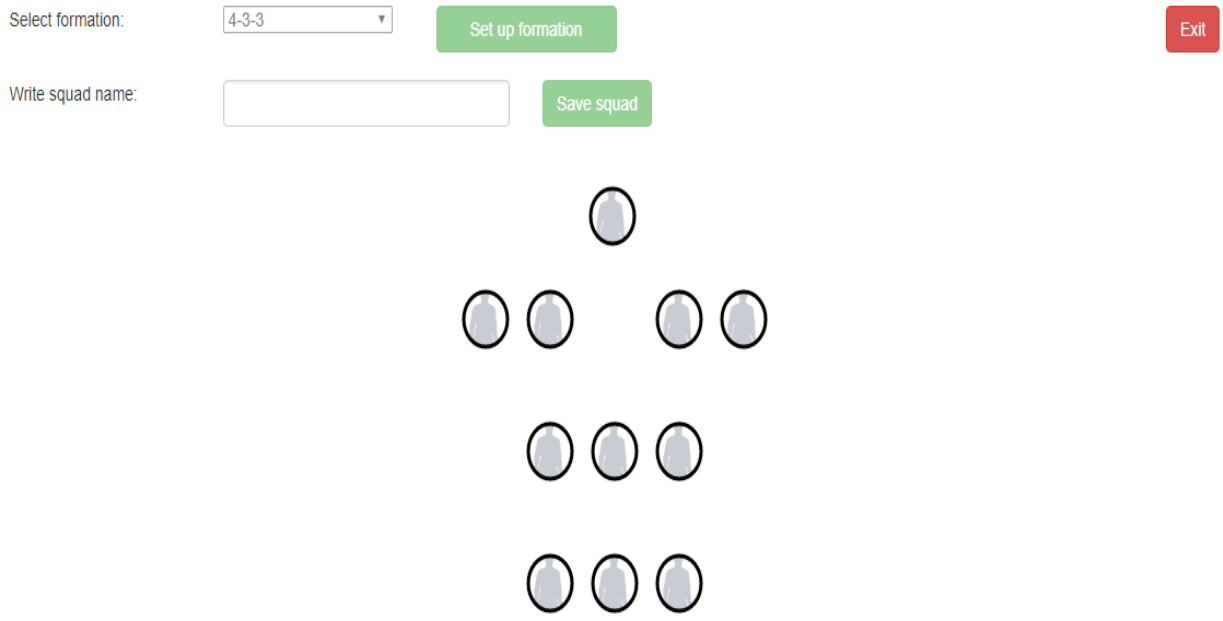
Nakon popisa legendi slijedi prikaz ukupnog broja obožavatelja diljem svijeta. Taj prikaz se sastoji od stilski uređenog brojača koji kreće od 0 i broji sve do trenutnog broja obožavatelja.



Slika 3.21.: Prikaz stranice za prikaz broja obožavatelja

Nakon prikaza broja obožavatelja određenog kluba slijedi interaktivni dio web aplikacije, a to je kreiranje momčadi od legendi i igrača koji su trenutno u klubu te igranje kviza u kojem se nalaze pitanja vezana za klub koji je korisnik bio odabrao.

Klikom na poveznicu „*Create squad*“ korisnika aplikacija preusmjerava na prikaz unutar kojeg korisnik treba odabrati željenu formaciju te potvrditi odabir kako bi mu se prikazao grafički prikaz formacije koju je odabrao.



Slika 3.22.: Prikaz stranice za kreiranje momčadi

Nakon što je korisnik odabrao željenu formaciju i potvrdio odabir formacije te se prikazala formacija, može započeti sa popunjavanjem pozicija u momčadi klikom na bilo koji od krugova u formaciji. Svaki krug unutar formacije predstavlja jednu poziciju. Klikom na određeni krug

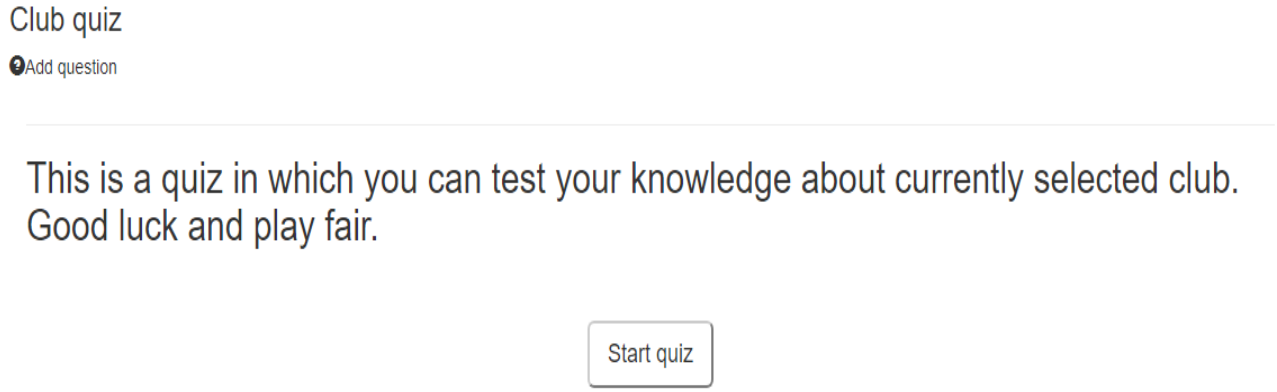
ispod formacije se prikaže tablica sa trenutnim igračima i legendama čija pozicija odgovara poziciji na kojoj se nalazi krug. Nakon odabira određenog igrača, njegova slika profila se pojavi unutar kruga gdje je taj igrač odabran.

Name	Surname	Position	Nationality
Iker	Casillas	Goalkeeper	Spain
Antonio Rodrigo	Bentacort	Goalkeeper	Spain
Francisco Kiko	Casilla Cortés	Goalkeeper	Spain
Luca	Zidane Fernández	Goalkeeper	France
Keylor	Navas	Goalkeeper	Costarica

Slika 3.23.: Prikaz tablice sa dostupnim igračima za momčad

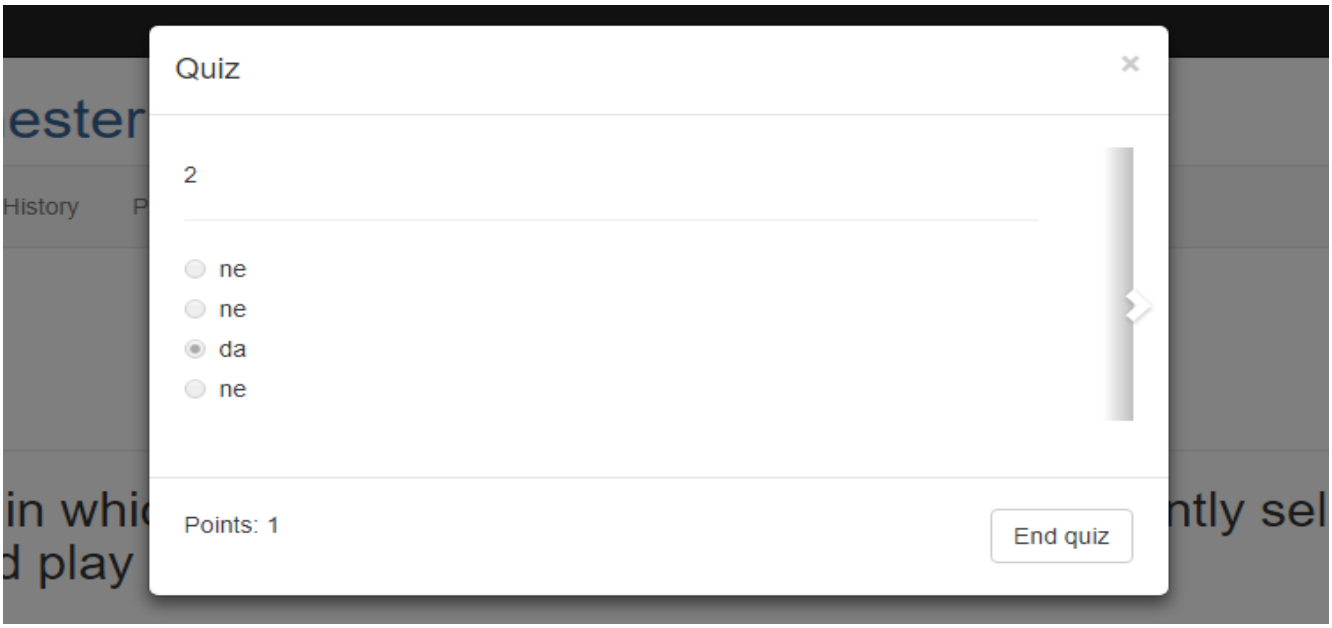
Prije nego korisnik može spremiti momčad u bazu podataka mora popuniti cijelu momčad (odabrati 11 igrača) i upisati u polje za unos naziv momčadi. Nakon što su popunjeni svi potrebni podaci korisnik može spremiti momčad u bazu podataka klikom na gumb „*Save squad*“.

Drugi interaktivni sadržaj aplikacije je kviz vezan za klub koji je korisnik odabrao. Prikaz unutar kojeg se pokreće kviz sastoji se od opisa kviza i gumba pomoću kojeg se pokreće kviz.



Slika 3.24.: Prikaz stranice za igranje kviza

Nakon klika na gumb „*Start quiz*“ otvara se skočni prozor unutar kojeg se igra kviz.



Slika 3.25.: Prikaz skočnog prozora za kviz

Prikaz unutar kojeg se igra kviz se sastoji od pitanja, odgovora, strelice za prebacivanje pitanja, bodova i gumba kojim se u bilo kojem trenutku može prekinuti igranje kviza. Nakon klika na gumb trenutno bodovno stanje se sprema u bazu podataka.

4. Zaključak

Ovaj diplomski rad može poslužiti kao veoma kvalitetna vježba za sve osobe koje imaju namjeru istraživati i baviti se ovim područjem programiranja kao i tehnologijama i okruženjima korištenim tijekom izrade ovog diplomskog rada. Za vrijeme izrade ovog diplomskog rada bilo je poteškoća koje su savladane u kratkom roku, a bitno je napomenuti da je svladavanje tih poteškoća pridonijelo usavršavanju u primjenjenom području programiranja. Također stečena su nova znanja u samom procesu razvoja ove aplikacije koja će pridonijeti u kasnijem istraživanju aplikacija ovog tipa. Za sam razvoj aplikacije ne može se reći da je bio lagan jer je arhitektura aplikacije dosta složena i zahtijeva dosta truda i uloženog vremena kako bi se došlo do konačnih rezultata koji su prikazani u ovoj dokumentaciji. Također potrebno je zahvaliti se mentoru na susretljivosti i ažurnosti kada je u pitanju odgovaranje na bilo kakva pitanja vezana uz ovu aplikaciju i općenito uz diplomski rad. Nakon završetka izrade ove aplikacije programer posjeduje znanja koja mu pružaju veoma kvalitetnu podlogu za zaposlenje nakon završenog studija.

Literatura

[1] C# Guide | Microsoft Docs, https://docs.microsoft.com/en-us/dotnet/csharp/, pristup ostvaren 12.11.2017.

[2] What is C#? - Definition from WhatIs.com, http://searchwindevelopment.techtarget.com/definition/C, pristup ostvaren 12.11.2017.

[3] All about the C# Programming Language, https://www.thoughtco.com/all-about-the-c-programming-language-958330, pristup ostvaren 12.11.2017.

[4] c# tutorial for beginners - YouTube, https://www.youtube.com/playlist?list=PLAC325451207E3105, pristup ostvaren 28.6.2017.

[5] Object Oriented Programming Using C# .NET, http://www.c-sharpcorner.com/UploadFile/84c85b/object-oriented-programming-using-C-Sharp-net/, pristup ostvaren 12.11.2017.

[6] C# Object Oriented Programming Basic to Advance, https://www.youtube.com/watch?v=e7Yj6vLyYOI, pristup ostvaren 4.7.2017.

[7] The Dependency Injection Design Pattern, https://msdn.microsoft.com/en-us/library/hh323705(v=vs.100).aspx, pristup ostvaren 13.11.2017.

[8] AutoMapper, http://automapper.org/, pristup ostvaren 14.11.207.

[9] What is AutoMapper and How to map two objects using AutoMapper?, http://www.kunal-chowdhury.com/2013/01/what-is-automapper-and-how-to-map-objects.html#Y9637PZb4c5sy8mU.97, pristup ostvaren 14.11.2017.

[10] Ninject - Wiki, http://www.ninject.org/wiki.html, pristup ostvaren 14.11.2017.

[11] Understanding Multilayered Architecture in .NET, http://www.c-sharpcorner.com/UploadFile/1492b1/understanding-multilayered-architecture-in-net/, pristup ostvaren 14.11.2017.

[12] Token Based Authentication Made Easy - Auth0, https://auth0.com/learn/token-based-authentication-made-easy/, pristup ostvaren 21.11.2017.

[13] ASP.NET | The ASP.NET Site, https://www.asp.net/, pristup ostvaren 13.11.2017.

[14] ASP.NET MVC | The ASP.NET Site, https://www.asp.net/mvc, pristup ostvaren 15.11.2017.

[15] Entity Framework | Microsoft Docs, https://docs.microsoft.com/en-us/aspnet/entity-framework, pristup ostvaren 15.11.2017.

[16] Getting Started with Entity Framework 6 Code First using MVC 5 | Microsoft Docs, https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application, pristup ostvaren 16.11.2017.

[17] An Introduction to Entity Framework with Rowan Miller | Seth Juarez | Channel 9, https://channel9.msdn.com/Blogs/Seth-Juarez/An-Introduction-to-Entity-Framework-with-Rowan-Miller, pristup ostvaren 16.11.2017.

[18] Entity Framework Database First, https://msdn.microsoft.com/en-us/library/jj206878(v=vs.113).aspx, pristup ostvaren 16.11.2017.

[19] Entity Framework Database First Approach, https://www.tutorialspoint.com/entity_framework/entity_database_first_approach.htm, pristup ostvaren 17.11.2017.

[20] Repository Pattern | DevIQ, http://deviq.com/repository-pattern/, pristup ostvaren 17.11.2017.

[21] The Repository Pattern, https://msdn.microsoft.com/en-us/library/ff649690.aspx, pristup ostvaren 18.11.2017.

[22] CRUD Operations Using the Generic Repository Pattern and Dependency Injection in MVC, http://www.codeproject.com/Articles/838097/CRUD-Operations-Using-the-Generic-Repository-Pat, pristup ostvaren 18.11.2017.

[23] Angular Docs, https://angular.io/, pristup ostvaren 02.09.2017.

[24] Understanding Angular 2 Components for AngularJS Devs, https://teamgaslight.com/blog/understanding-angular-2-components-for-angularjs-devs, pristup ostvaren 19.11.2017.

[25] Angular 2 tutorial for beginners - YouTube, https://www.youtube.com/playlist?list=PL6n9fhu94yhWqGD8BuKuX-VTKqlNBj-m6, pristup ostvaren 05.09.2017.

[26] Angular 2 Tutorials - YouTube, https://www.youtube.com/playlist?list=PLC3y8-rFHvwg5gEu2KF4sbGvpUqMRSBSW, pristup ostvaren 10.09.2017.

Sažetak rada

Tema diplomskog rada je „Interaktivna aplikacija za obožavatelje nogometnih klubova“. Aplikacija se sastoji od desetak projekata koji zajedno međudjeluju kao cjelina. Svaki od projekata ima određenu zadaću koju mora u cjelosti točno obavljati. Ukoliko bilo koji od projekata ne obavlja svoju zadaću kako treba, aplikacija ili neće točno prikazivati podatke tj. neće ispravno raditi ili neće uopće biti mogućnosti da ju korisnik pokrene. Nakon pokretanja korisnik se mora prijaviti u sustav ukoliko je registriran, a ako korisnik nije registriran u sustavu mora se prvo registrirati kako bi mogao pristupiti sadržaju. Nakon prijave korisnik ima mogućnost odabrati neki od ponuđenih 5 klubova. Nakon odabira kluba korisnik može pregledati povijest kluba, trenutnu listu seniorskih igrača u klubu, detalje o odabranom igraču, osoblje kluba, legende kluba, broj obožavatelja te kreirati svoj početni sastav i odigrati kviz vezan za klub koji je odabrao. Također korisnik može pregledati svoje momčadi koje je kreirao i vidjeti rezultate koje je postignuo u igranju kviza na korisničkoj stranici koja se automatski prikazuje nakon prijave korisnika u sustav. Na početku izrade ove aplikacije prvo je kreirana baza podataka, a nakon toga projektna arhitektura unutar Visual Studio razvojnog okruženja. Nakon toga je povezana baza podataka sa projektnom arhitekturom te je započeta izrada serverskog dijela aplikacije. Izrada serverskog dijela je potrajala zbog složenosti same aplikacije, a nakon što je završen serverski dio aplikacije započeta je izrada prezentacijskog dijela aplikacije korištenjem Angular razvojnog okruženja. Serverski dio i prezentacijski dio su povezani i djeluju kao cjelina korištenjem HTTP protokola, točnije, korištenjem HTTP zahtjeva. Nakon što su u cjelosti završeni i serverski dio i prezentacijski dio aplikacije, aplikacija u potpunosti obavlja sve funkcionalnosti koje su navedene u ovoj dokumentaciji.

KLJUČNE RIJEČI: C#, klasa, sučelje, projekt, angular, dependency injection, automapper, višeslojno, sloj, ninject, .NET, okruženje, API, simbol, repozitorij, servis, komponenta, baza podataka

Abstract

Interactive application for football club fans

Topic of this graduate paper is „Interactive application for football club fans“. Application is constited of dozen projects which work together as a unit. Each project has specific task which he needs to perform fully correct. If any project doesn't perform task as it should, application will not show data correctly i.e. won't work correctly or the user will not be able to use it. After running the application user needs to sign in if he is registered into the system, if the user isn't registered into the system he needs to register in order to access the content. After signing, user can select one of five suggested clubs. After he selects one club he can see club history, current players in the club, details about selected player, staff team, club legends, number of fans, create his squad and play quiz related to selected club. Also, user can see squads that he made and scores that he achieved playing quiz on user profile page on which user is automatically redirected after signing in. At the begging of the making this application, database was created first and after that project architecture was created inside Visual Studio framework. After that database was connected with project architecture and application server side creation started. Application server side creation took a lot of time because of the complexity of the application and after server side was finished, creation of presentation part of the application has started using Angular framework. Server side and presentation part are connected and work as a unit using HTTP protocol, to be precise, using HTTP requests. After parts of application are fully finished, application fully performs all functionalities that are mentioned in this graduate paper.

KEY WORDS: C#, class, interface, project, angular, dependency injection, automapper, multi-layer, layer, ninject, .NET, framework, API, token, repository, service, component, database

Životopis

Josip Tomaić je rođen 26.12.1993. u Našicama. Osnovnu školu je upisao 2000. godine u školi Josipa Jurja Strossmayera u Đurđenovcu koju je završio 2008. godine. Nakon završene osnovne upisao je srednju ekonomsku školu u srednjoj školi Isidora Kršnjavog u Našicama iste godine koju je završio 2012. godine. Nakon završene srednje škole upisao je smjer računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku u 2012. godini. 2015. godine je završio trogodišnji smjer na upisanom fakultetu te je upisao diplomski studij koji traje dvije godine.Član je DVD-a Našičko Novo selo od 2010. godine te je 2015. godine položio ispit za vatrogasca. Sa spomenutim društvom osvojio je brojne nagrade zbog postignutih rezultata na vatrogasnim natjecanjima seniorskih ekipa. Od 2014. godine član je udruge pod nazivom „Udruga mladih Đurđenovac“ u Đurđenovcu i zajedno sa ostalim članovima obavlja volonterske aktivnosti za koje je udruga dobila nekoliko priznanja.

JOSIP TOMAIĆ

Prilozi

Na DVD-u koji se prilaže uz dokumentaciju nalaze se:

Dokumenti:

FootballFanApp_dokumentacija_Tomaić_Josip.docx

FootballFanApp_dokumentacija_Tomaić_Josip.pdf

Datoteke:

Slike

Izvorni kod aplikacije

Cjelokupni projekt