Web aplikacija za upravljanje teretanom

Šitina, Ivan

Master's thesis / Diplomski rad

2019

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:396621

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Sveučilišni studij

WEB APLIKACIJA ZA UPRAVLJANJE TERETANOM

Diplomski rad

Ivan Šitina

Osijek, 2019

SADRŽAJ

1. UVOD 1
   1.1. Zadatak diplomskog rada 1
2. TEHNOLOGIJE I PROGRAMSKI JEZICI 2
   2.1. Razvojno oruženje 2
   2.2. T-SQL 2
   2.3. C# 3
   2.4. HTML 4
      2.4.1. Struktura HTML-a 4
   2.5. CSS 6
      2.5.1. Struktura CSS-a 7
   2.6. Bootstrap 7
   2.7. JavaScript i JQuery 8
3. MVC RAZVOJ APLIKACIJE 10
   3.1. ASP.NET 10
   3.2. MVC arhitektura 10
      3.2.1. Pojam MVC 10
      3.2.2. Dijelovi MVC arhitekture 11
   3.3. Kontroleri i akcije 12
   3.4. Modeli 14
   3.5. Pogledi 14
      3.5.1. Sintaksa u Razor pogledu 15
      3.5.2. Layout pogledi 16
      3.5.3. Sekcije 16
      3.5.4. Parcijalni pogledi (eng. Partial views) 17
   3.6. Pomoćne (helper) metode 17
   3.7. Validacije 18
   3.8. ADO.NET i Entity Framework 19
      3.8.1. ADO.NET 19
      3.8.2. Entity Framework 20
      3.8.3. LINQ 22
   3.9. WEB Api 23
      3.9.1. GET 25
      3.9.2. POST 25
      3.9.3. PUT 25

3.9.4. DELETE 25
3.10. URL usmjeravanje 27
3.11. Sigurnost aplikacije 28
4. WEB APLIKACIJA 30
4.1. Uvod u aplikaciju 30
4.1.1. Registracija u sustav 30
4.1.2. Prijava u sustav 31
4.2. Admin rola 32
4.3. Trener rola 39
4.4. Rola korisnika 42
4.5. Dodatno 45
4.5.1. Profil 45
4.5.2. Ocjenjivanje trenera 46
5. ZAKLJUČAK 48
LITERATURA 49
SAŽETAK 50
ABSTRACT 51
ŽIVOTOPIS 52

1. UVOD

Glavni zadatak rada je izrada web aplikacije koja omogućuje evidenciju i prikaz svih članova teretane, kao i kreiranje, uređivanje i brisanje istih, ali i implementaciju sustava kojega koristi više krajnjih korisnika s različitim pravima pristupa.

Rad je podijeljen u četiri poglavlja. U prvom poglavlju opisan je zadatak diplomskoga rada. Drugo poglavlje donosi pregled i objašnjenje korištenih tehnologija pri izradi zadane aplikacije. Korišten je programski jezik C#. Za izradu baze podataka korišten je MS SQL Server, dok su na korisničkom sučelju korištene tehnologije HTML, CSS, Razor, Javascript i JQuery.

U trećem poglavlju je prikazan razvoj aplikacije u MVC arhitekturi, dok zadnje poglavlje donosi opis rada i mogućnosti izvedene web aplikacije.

1.1. Zadatak diplomskog rada

Zadatak diplomskog rada je dizajnirati i napraviti web aplikaciju za upravljanje teretanom koja nudi različite programe i usluge.

2. TEHNOLOGIJE I PROGRAMSKI JEZICI

U ovom poglavlju opisane su tehnologije i programski jezici korišteni pri izradi web aplikacije.

2.1. Razvojno oruženje

Prilikom razvijanja rada korišten je Microsoft Visual Studio 2017 koji podržava razne programske jezike. Uz to, nudi i široki spektar komponenata koje pojednostavljuju razvijanje aplikacija poput *Intellisense* komponente koja predlaže ostatak kôda kao i refaktoriranje kojim se poboljšava kôd pa ga je lakše za održavati. Visual Studio sadrži još i alate poput dizajnera sheme baza podataka, dizajnera klasa, web dizajnera. Također, prihvaća proširenja koja unaprjeđuju funkcionalnosti na skoro svim područjima, dodajući podršku sistema za manipulaciju izvornim kodom i skupinu novih alata.

Osim Microsoft Visual Studia, korišten je i Microsoft SQL Server 2017 kao relacijska baza podataka kojemu je primarni jezik za upite Transact SQL (T-SQL), što znači da osim osnovnih i klasičnih operacija nudi i složenije stvari poput mijenjanja programskog toka i slično.

2.2. T-SQL

Transact-SQL (T-SQL) proizvod je tvrtke Microsoft i Sysbase koji se koristi pri interakciji s relacijskim bazama podataka. Upotpunjava SQL (eng. Structured Query Language) standard uključenjem nekoliko proširenja koja su razvijena na njemu. Neka od proširenja kojima T-SQL čini SQL moćnijim su mogućnost deklariranja varijabli, kontrola transakcija, upravljanje pogreškama i iznimkama, operacija nad nizovima, obrada datuma i vremena i niz ugrađenih matematičkih funkcija.

T-SQL koristi Microsoft SQL Server, a sve aplikacije koje komuniciraju s instancom SQL servera čine to slanjem T-SQL naredbi serveru, neovisno o korisničkom sučelju aplikacije.

```
--Creating stored Procedure with parameter and executing it
CREATE PROCEDURE usp_SalesbyCountry_with_out
(
@Country  varchar(20),
@sum_amount int OUT
)
AS
BEGIN
SELECT @sum_amount=SUM(SalesAmount)
FROM tbl_Sales2
WHERE Country=@Country
GROUP BY Country
END

DECLARE @sum_amount int
EXEC usp_SalesbyCountry_with_out 'USA',@sum_amount=@sum_amount OUT
SELECT @sum_amount AS Total_saleamount_USA
```

Sl.2.1. *Primjer korištenja Storanih procedura*

2.3. C#

C# je objektno orijentirani programski jezik kojeg je razvio Microsoft 2002. godine s verzijom 1.0 Microsoftovog .NET okvira. Danas je jedan od najpopularnijih programskih jezika za izradu Windows programa i web aplikacija. Izveden je od programskog jezika C, a sličan je programskom jeziku C++. Upotrebljava iste osnovne operatore poput C++ programskog jezika, objektno je orijentiran, osjetljiv na mala i velika slova, a ima i vrlo sličnu sintaksu. Ono što je najvažnije, C# je posebno dizajniran za Microsoftov .NET okvir što programerima omogućuje iskorištavanje svih značajki koje nudi .NET API. [1]

```
class Program
{
    static void Main(string[] args)
    {
        int ppid = 0;
        if (args != null && args.Length > 0)
        {
            int.TryParse(args[0], out ppid);
        }
        ProtectProcess.AntiDebug.DebugSelf(ppid);
        Console.WriteLine("Self-Debugging...");
        Console.ReadLine();
    }
}
```

Sl.2.2. *Primjer C# koda*

2.4. HTML

HTML (eng. HyperText Markup Language) je osnovni alat za izradu web stranica, koji preglednicima (Mozilla Firefox, Google Chrome, Internet Explorer, itd.) govori sve o izgledu i sadržaju web stranice. Međutim, postoje razni alati koji omogućavaju izradu web stranica i bez poznavanja HTM-a. Takvi alati automatski generiraju kôd po uputama korisnika, preko manje ili više jednostavnog korisničkog sučelja. Riječ je o tzv. vizualnom uređivanju gdje korisnik, odabirom određenih alata, uređuje web stranicu i prati promjene u realnom vremenu koje će se kasnije prikazati u nekom od preglednika.

Vizualno uređivanje ima nekoliko prednosti. Osim što korisnik odmah vidi sadržaj vlastite web stranice, program će sam generirati potreban kôd, zbog čega će moguće pojave grešaka biti svedene na minimum. HTML nije programski jezik jer njime ne možemo izvršiti nikakvu logičku operaciju. Svrha mu je opis hipertekstualnih dokumenata.

2.4.1. Struktura HTML-a

Serilizacija je proces prevođenja objekata u sekvencu bitova kako bi objekt mogao biti sačuvan u datoteci ili prenijet preko mreže, radi mogućnosti vraćanja u prvobitni oblik, vjerojatno najrašireniji način za serijalizaciju podataka danas je jezik tagova pa gotovo da ne postoji sustav koji informacije ne dobiva iz jedne ovakve strukture. Isti način komunikacije upotrebljava i HTML, zbog čega je potrebno upoznati se s načinom njegovog funkcioniranja. Svaki HTML kôd sastoji se od sadržaja zatvorenog u tagove. Tagovi su elementi HTML kôda. Prvi element je *<html>* i on je obavezan na svakoj stranici jer preglednik na taj način zna da se radi o početku

HTML dokumenta, dok element *</html>* označava njegov kraj. Dio dokumenta koji je smješten između elemenata *<head>* i *</head>* naziva se zaglavlje dokumenta i on se ne prikazuje u pregledniku, a svrha mu je detaljniji opis stranice koja se prikazuje.

Tekst koji se nalazi između elemenata *<title>* i *</title>* predstavlja naslov stranice i ispisuje se u naslovnoj stranici preglednika, a njegov način zapisa se, također, vidi u kôdu iznad. Unutar elemenata *<body>* i *</body>* nalazi se dio dokumenta koji preglednik prikazuje. Unutar tih elemenata moguće je dodavati i neke druge elemente kako bi se formirao izgled stranice, što će se opisati u nastavku teksta. Svi navedeni elementi su obavezni i predstavljaju minimalan zahtjev koji svaka stranica mora zadovoljiti. Kada je riječ o interaktivnosti korisnika s bazom podataka, primjenjuju se HTML obrasci. Za označavanje obrasca u HTML-u se koristi element za početak obrasca i za kraj. Struktura obrasca je veoma slična klasičnim obrascima na papiru. Korisnik s traženim podacima ispunjava polja koja se nalaze unutar obrasca. Kada popuni sva tražena polja, šalje ih na obradu pomoću skripte koja se nalazi na poslužitelju. Kada poslužitelj obradi sadržaj obrasca, kao rezultat obrade izvršava određenu akciju.

Obrazac predstavlja zasebni dio HTML stranice. Unutar njega se mogu pojaviti različiti tipovi polja; polja za unos teksta, polja za unos lozinke, tekstualni okvir, padajući izbornik, kružni izbornik, kontrolni kvadratić i gumb. Na jednoj stranici se može naći više obrazaca pri čemu za svaki mora biti naznačen njegov početak i kraj. Atributi koje obrazac ima su: identifikator (*id*), linijski stil (*style*), ime (*name*), ruta do funkcije za obradu podataka iz obrasca (*action*), HTTP (eng. Hyper Text Transfer Protocol) metoda koja šalje podatke do funkcije koja ih obrađuje (*method*, može biti POST ili GET), tip podatka koji se šalje (*enctype*, nije potrebno navoditi, osim kada se radi o slanju datoteka, tada se navodi kao *multipart/form-data*), detalji o okviru u koji se podaci šalju (*target*).

Različiti tipovi polja u obrascu se definiraju pomoću elementa *<input>* i vrijednosti atributa. Kao i obrazac, i element *<input>* ima svoje atribute, a oni su: identifikator (*id*), linijski stil (*style*), ime polja (*name*), tip polja (*type*), podatak koji se šalje na obradu (*value*), onemogućavanje unosa vrijednosti (*disabled*), označavanje polja predviđeno samo za čitanje (*readonly*), broj znakova u polju (*size*), maksimalan broj znakova za unos (*maxlength*), putanje do slike (*src*), redni broj polja u obrascu (*tabindex*).

S obzirom na vrijednost atributa, razlikuju se sljedeći tipovi polja: jednostavno tekstualno polje (*text*), polje za unos lozinke (*password*), kontrolni kvadratić (*checkbox*), kružni izbornik (*radio*), gumb za prihvat podataka (*submit*), polje za odabir dokumenta (*file*), gumb za čišćenje obrasca (*reset*), nevidljivo polje (*hidden*), slika (*image*), multifunkcionalni gumb (*button*). Ne vrijede svi atributi za sve tipove polja. Atributi *readonly*, *size*, *maxlength* vrijede samo za polja *text* i

password, atribut *checked* vrijedi za polja *radio* i *checkbox*, a atribut *src* vrijedi samo za tip polja *img*. Jednostavni tekstualni element je element od jednog reda, a može biti i tekstualno polje za unos lozinke. Ukoliko je potrebno tekstualno polje, tada se atributu *type* pridjeljuje vrijednost *text*, a ako je potrebno polje za unos lozinke, tada atribut *type* ima vrijednost *password*. [2]

```
<!DOCTYPE html>
<html>
<title>HTML Tutorial</title>
<body>

<h1>This is a heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Sl. 2.3. *Primjer HTML strukture*

2.5. CSS

CSS (eng. Cascading Style Sheet) je tehnologija koja služi za dizajniranje HTML dokumenata. Nije bila dostupna od početka HTML-a i do tada se dizajn stranica predstavljao pomoću tablica. Svojom pojavom CSS je unio revoluciju u izradi HTML stranica, prvenstveno zato što nudi niz prednosti u formiranju stranica, u odnosu na način formiranja stranica pomoću tablica. Mogućnosti formatiranja sadržaja upotrebom HTML-a su poprilično ograničene. Upotreba CSS-a omogućava razdvajanje prezentacije podataka i dizajn od same strukture podataka. Rezultat toga razdvajanja je manji HTML kôd, koji je ujedno i pregledniji te se olakšava manipuliranje njime. Osim toga, CSS je omogućio veliki broj novosti za prikaz podataka koji ne postoje u HTML-u, a vremenom su se razvile tehnike koje su skraćivale vrijeme izrade HTML stranica. Tri su načina pisanja stilova:

1) Inline - piše se unutar HTML datoteke.
2) Interni - piše se, također, unutar HTML-a.
3) Eksterni – piše se u vanjskoj datoteci koju povezujemo s HTML dokumentom.

Najčešći način pisanja CSS stilova je odvajanje u posebnu datoteku koja se povezuje s HTML dokumentom. Prednost u odnosu na ostala dva načina je odvojenost HTML-a i CSS-a što omogućava uključivanje jednoga CSS stila na više stranica. Ako je CSS pisan u vanjskoj datoteci, potrebno ga je uključiti u HTML datoteku. [2]

2.5.1. Struktura CSS-a

Sintaksa CSS-a ima oblik: selektor *{ svojstvo: vrijednost; }*. Selektor označava HTML element na koji se stil primjenjuje. Svojstvo određuje šta mijenjano određenom selektoru, a vrijednost označava koja vrijednost se dodjeljuje određenom svojstvu. Vrijednost može biti boja, veličina slova, font slova i sl. Postoji više tipova selektora, a nekoliko najkorištenijih su:

1) Jednostavni selektori
2) Klasni selektori
3) Id selektori

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

Sl. 2.4. *Primjer CSS strukture*

2.6. Bootstrap

Bootstrap je jedan od često korištenih okvira za izradu responzivnih web stranica. Sadrži HTML i CSS temeljne predloške za zajedničke komponente korisničkog sučelja, poput gumbova, padajućih izbornika, obrazaca, tablica, navigacije, upozorenja, modalnih prozora i slično.

Može se koristiti u sklopu bilo koje tehnologije na strani poslužitelja i na bilo kojoj platformi. Neke od njegovih prednosti su podržavanje odgovarajućeg dizajna i omogućavanje bržeg i lakšeg stvaranja odgovarajuće web aplikacije. Jedna od bitnijih značajki je responzivni dizajn koji omogućuje aplikaciji automatsko prilagođavanje različitim veličinama zaslona na kojima se web aplikacija koristi. [3]

```
<table class="table table-dark">
  <thead>
    <tr>
      <th scope="col">#</th>
      <th scope="col">First</th>
      <th scope="col">Last</th>
      <th scope="col">Handle</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <th scope="row">1</th>
      <td>Mark</td>
      <td>Otto</td>
      <td>@mdo</td>
    </tr>
    <tr>
      <th scope="row">2</th>
      <td>Jacob</td>
      <td>Thornton</td>
      <td>@fat</td>
    </tr>
    <tr>
      <th scope="row">3</th>
      <td>Larry</td>
      <td>the Bird</td>
      <td>@twitter</td>
    </tr>
  </tbody>
</table>
```

Sl. 2.5. *Primjer Bootstrap tablice*

2.7. JavaScript i JQuery

JavaScript je programski jezik za skriptiranje koji se koristi za izradu interaktivnih web stranica. Riječ je o trećem sloju standardnih web tehnologija uz HTML i CSS, opisanih u prethodnim poglavljima. Također, postoje i naprednije verzije JavaScripta na poslužiteljskoj strani koje omogućuju dodavanje više funkcionalnosti na web stranicu, poznatiji kao JavaScript okviri (eng. framework). JavaScript ima mogućnost povezivanja s objektima svoje okoline u svrhu osiguranja programskog nadzora nad njima unutar web preglednika. Sadrži i standardnu biblioteku objekata i matematičkih funkcija te osnovni skup jezičnih elemenata poput operatora, kontrolnih struktura i slično.

Jezgra JavaScript-a ima mogućnost proširenja u različite svrhe i manipulaciju nad objektima poput postavljanja u elemente HTML obrazaca i odgovaranja na događaje korisnika kao što su klikovi miša, unos obrazaca te navigacija pogleda.

```
<button onclick="myFunction()">Click Me!</button>

<script>
function myFunction() {
  var x = document.getElementById("demo");
  x.style.fontSize = "25px";
  x.style.color = "red";
}
</script>
```

Sl. 2.6. *Primjer JavaScript koda*

S druge strane JQuery je jedan od brojnih JavaScript biblioteka razvijen u svrhu nadogradnje i poboljšanja primarnog JavaScript-a. Pojednostavljuje sintaksu JavaScript-a i omogućuje efikasniju interakciju s drugim programskim jezicima namijenjenih za izgradnju web aplikacija. Također, omogućuje jednostavan pristup DOM-u (eng. Document Object Model) što omogućuje izradu animacija i dinamičkih AJAX (eng. Asynchronous JavaScript And XML) web komponenti na jednostavan način za razliku od klasičnog JavaScript-a. Uporabom AJAX funkcionalnosti JQuery biblioteke postoji mogućnost izrade web aplikacija koje se u brojnim stvarima ponašaju kao *Windows Desktop* aplikacije. U klasičnim web aplikacijama, kod promjene samo jednog parametra na obrascu, potrebno je ponovno učitavanje cijele stranice sa svim njezinim podacima. Međutim, s ovom tehnologijom taj problem je riješen, a rezultat je brže izvršavanje aplikacije. Još neke od prednosti JQuery-a su: slično ponašanje u različitim preglednicima, jednostavnost uporabe, veliki broj dodatnih biblioteka i proširenja, podrška za AJAX i slično. [4]

Na sljedećoj slici je primjer JQuery koda.

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
```

Sl. 2.7. *Primjer JQuery koda*

3. MVC RAZVOJ APLIKACIJE

3.1. ASP.NET

ASP.NET je Microsoftova tehnologija koja se koristi za razvoj i izvršavanje web aplikacija i servisa. Njezina prva verzija nastala je početkom 2002. godine. Riječ je o naprednijem sljedbeniku starije tehnologije ASP (eng. Active Server Pages). Zasniva se na .Net okviru i CLR-u (eng. Common Language Runtime), koji omogućuju pisanje programskog koda u bilo kojem od podržanih .Net jezika. Tradicionalni razvoj web aplikacija u ASP.NET-u baziran je na web obrascima koji obuhvaćaju mnogobrojne korisničke kontrole koje se običnom tehnikom povuci i ispusti (eng. Drag and drop) mogu ugraditi na web stranicu. U početnim fazama razvoja ASP.NET-a većina programera je usavršavala *Desktop*, tj. *Windows* aplikacije bazirane na programiranim događajima (eng. Event driven). Taj način programiranja gotovo je u potpunosti preslikan u web obrascima što je mnogobrojnim razvojnim inženjerima omogućilo lakši prijelaz prema razvoju i unaprjeđenju web aplikacija.

U tradicionalnim *Desktop* aplikacijama vrijednost varijable automatski se sprema u memoriju računala na kojoj se izvršava aplikacija, gdje se čuva dok pojedina varijabla ili objekt trebaju korisniku. Kod web aplikacija postoji problem pohrane podataka za svaki zahtjev (eng. Request) prema web poslužitelju. Nakon što se web stranica generira i kao odgovor (eng. Response) pošalje nazad prema klijentu, podaci ne bivaju sačuvani, nego bi se trebali osmisliti neki drugi načini pohrane podataka koji su nužni za buduće korištenje aplikacije. Ovaj problem čuvanja stanja poznat je stručnjacima kao *state problem*, a rješava se pomoću različitih ugrađenih mehanizama. Neki od važnijih mehanizama za rješavanje *state problema* kod ASP.NET-a su: skrivena polja na stranici, *query string*, *session state*, *application state* i *cookies*. [5]

3.2. MVC arhitektura

3.2.1. Pojam MVC

MVC je kratica za *Model-View-Controller*, a označava *softwaresku* arhitekturu za razvoj web aplikacija. U svrhu organizacije ranih GUI (eng. Graphical User Interface) aplikacija, sama je arhitektura prvi put upotrijebljena 70-ih godina dvadesetog stoljeća. Popularnost MVC arhitekture rasla je razmjerno s evolucijom Interneta i porastom korištenja web aplikacija. Upravo zbog svoje usklađenosti s primarnim principima funkcioniranja web aplikacija, ponovno je postala aktualna. Osim tog ključnog razloga, MCV ima mogućnost razdvajanja podataka,

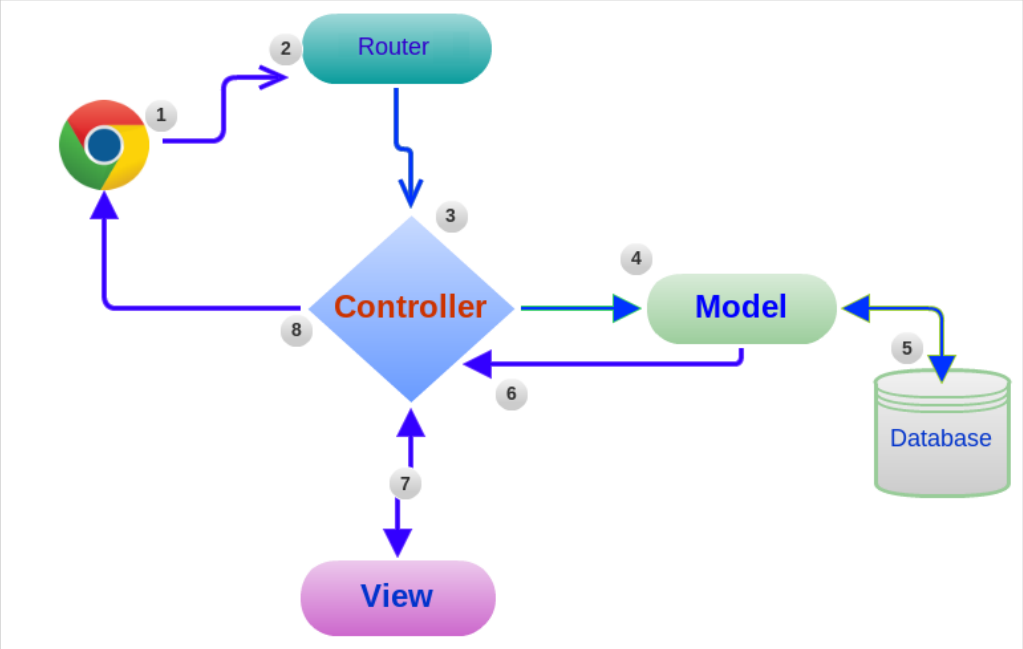
programsku logiku i korisničko sučelje što pridonosi olakšavanju održavanja aplikacije kao i njihovog testiranja. Upravo sve navedene karakteristike i prednosti, izdvajaju danas MVC kao osnovnu arhitekturu za izradu web aplikacija. [5]

3.2.2. Dijelovi MVC arhitekture

Kao što je u uvodnom dijelu navedeno, MVC je sastavljen od tri osnovna dijela odnosno komponente:

1) Model – sadrži podatke s kojima korisnik radi i koje program koristi
2) Pogled (eng. View) – generirani rezultat, odnosno web stranica koja sadrži podatke modela
3) Kontroler (eng. Controller) – prima sve korisničke zahtjeve (eng. Request), izvršava razne operacije na modelu, odabire i šalje rezultat, odnosno pogled koji se generira prema korisniku. [5]

Implementacija arhitekture MVC u ASP.NET-u prikazana je prema slici 3.1.



Sl. 3.1. *Arhitektura MVC*

Kao što je jasno iz shematskog prikaza implementacije arhitekture MVC u ASP.NET prikazano na slici 3.1., kontroleru je omogućeno upravljanje podacima modela, a svi zahtjevi od strane korisnika, dolaze kontroleru. Promjene stanja modela mogu se trajno zapisati u npr. relacijskoj bazi podataka. Kontroler s podacima modela poziva odgovarajući pogled, koji se generira od strane tzv. *View enginea* i šalje kao rezultat korisniku. U prvim verzijama MVC-a kao pregledi

koristili su se pogledi *aspx* stranice, ali od verzije MVC3 pogledi se kreiraju korištenjem sintakse *Razor*, dok datoteka *Viewa* ima dodatak *cshtml*, ako se upotrebljava C# kao programski jezik, ili *vbhtml* ako se upotrebljava jezik Visual Basic. [5]

3.3. Kontroleri i akcije

Kontroleri (eng. Controller), u svojstvu jednog od glavnih dijelova MVC-a, zaprimaju svaki zahtjev prema aplikaciji i upravljaju tim zahtjevom pomoću metoda tzv. akcija (eng. Action methods). Ključna karakteristika kontrolera je da ne pohranjuju i ne sadrže podatke. Također, ne generiraju korisničko sučelje direktno, nego prvenstveno procesiraju dolazne zahtjeve. Najbanalniji način implementacije kontrolera je nasljeđivanje klase *Controller* iz imenskog prostora *System*. Akcijske metode su *Web.Mvc*. metode unutar klase kontrolera koje zaprimaju zahtjeve. Svaka od tih metoda poziva se posebnim URL-om (eng. Uniform Resource Locator), te upravlja i usmjerava rezultate poziva. Upravo zbog toga, akcijske metode oblikuju ponašanje kontrolera.

Pri pozivu akcijske metode u kontorleru, u većini je slučajeva potrebno prenijeti i pročitati određene podatke koji dolaze s zahtjevom, bilo da se radi o podacima iz URL-a ili onima s obrasca na web stranici. Tri su osnovna načina preuzimanja i čitanja tih podataka:

1) Podaci se preuzimaju unutar kontekstnih objekata (eng. Context objects)
2) Podaci se preuzimaju kao vrijednosti parametara kojima se zovu metode kontrolera
3) Korištenjem povezivanja modela (eng. Model binding)

Kontekstni objekti su oni objekti koji sadrže podatke i informacije vezane uz zahtjev kao što je primjerice naziv klijentskog računala, IP (eng. Internet Protocol) adresa istog, korisničko ime korisnika aplikacije i mnoštvo drugih korisnih informacija. Podaci se mogu dohvatiti preko svojstava kontekstnih objekata, dostupnih u klasi *Controller*. Neka od bitnijih svojstava prikazana su u tablici 3.1. u nastavku.

Tablica 3.1. *Svojstva kontekstnih objekata*

Svojstvo	Tip	Opis
Request.QueryString	NameValueCollection	Varijable poslane preko URL-a GET request-om
Request.Form	NameValueCollection	Varijable poslane s obrasca u POST request-u

Request.Cookies	string	Cookies poslani zahtjevom s preglednika
Request.Url	Uri	Zahtjevana URL adresa
Request.UserHostAddress	string	IP adresa s koje je poslan zahtjev
RouteData.Values	RouteValueDictionary	Kolekcija route parametara preuzetih iz URL-a
HttpContext.Application	HttpApplicationStateBase	Aplikacijski spremnik
HttpCOntext.Session	HttpSessionStateBase	Session (korisnički) spremnik
User	Iprincipal	Autentikacijski podaci o prijavljenom korisniku
TempData	TempDataDictionary	Privremeni spremnik podataka za korisnika
Server	HttpServerUtilityBase	Podaci poslužitelja

U svrhu generiranja izlaza, kao rezultata akcija u kontroleru, klasa *Controller* sadrži niz metoda, koje kao rezultat daju objekte naslijeđene od klase *ActionResult*. Ti objekti generiraju odgovarajući izlaz prema klijentu preko objekta *Response*. „Rezultat se može generirati preko *viewa*, redirekcijom prema nekom drugom *viewu* ili URL-u metodom *Redirect* ili redirekcijom na neku drugu metodu metodom *RedirectToAction*.“ [5]

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace Tutorial8.Controllers
{
    public class HomeController : Controller
    {

        public ActionResult Index(string id)
        {
            return Content(id);
        }
                                  Country "India"   state "Maharashtra"
        public ActionResult Find(string Country, string state)
        {
            string result = "Country :- " + Country + " State :- " + state;

            return Content(result);
        }


    }
}
```

Sl. 3.2. *Primjer Controllera i akcija*

3.4. Modeli

Povezivanje modela označava općenito proces stvaranja .NET objekata pri slanju podataka od strane klijenta. Na primjer, ako korisnik popuni obrazac podacima te ih, pomoću gumba *submit*, pošalje akcijskoj metodi koja sadrži parametar ili parametre koji se povezuju sa poslanim podacima, MVC *framework* obavi povezivanje navedenim postupkom povezivanja modela (eng. Model binding). Na taj se način poslani podaci mogu obraditi u samoj metodi i zatim pozvati neke druge akcije. [5]

```
3 references
public class DataSourceRequest
{
    2 references
    public int PageSize { get; set; }
    1 reference
    public int Skip { get; set; }
    2 references
    public int Page { get; set; }

    1 reference
    public int Take { get; set; }

    0 references
    public List<Sort> Sort { get; set; }
}

1 reference
public class Sort
{
    0 references
    public string Field { get; set; }
    0 references
    public string Dir { get; set; }
}
```

Sl. 3.3. *Primjer Modela*

3.5. Pogledi

U kontroleru je jedan od najčešćih rezultata akcijskih metoda pogled (eng. View). On se kao odgovor šalje korisniku u web preglednik. Sintaksa koja omogućava stvaranje dinamičkog pogleda na poslužitelju dio je MVC-a koji se zove *Razor*. Stoga, *Razor* praktički predstavlja *markup* jezik poslužitelja koji dopušta da se na određen način procesira, ugradi dinamički sadržaj i proizvede HTML stranica koja će se poslati korisniku. [5]

```
@model Gym.Models.Course

@{
    ViewBag.Title = "Create";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<h2>Kreiraj novo</h2>

@using (Html.BeginForm("Save", "Courses"))
{
    <div class="form-group">
        @Html.LabelFor(m => m.CourseName)
        @Html.TextBoxFor(m => m.CourseName, new { @class = "form-control", @autocomplete = "off" })
    </div>
    <div class="form-group">
        @Html.LabelFor(m => m.Price)
        @Html.TextBoxFor(m => m.Price, new { @class = "form-control", @autocomplete = "off" })
    </div>

    @Html.HiddenFor(m => m.Id)
    @Html.AntiForgeryToken()
    <button type="submit" class="btn btn-success">Spremi</button>
    <div>
        @Html.Partial("_BackToListButtonPartial")
    </div>
}
```

Sl. 3.4. *Primjer Viewa*

3.5.1. Sintaksa u Razor pogledu

Razor je *markup* jezik s poslužiteljske strane. Dvije su vrste sadržaja web stranica koje koriste ovu sintaksu: korisnički sadržaj i poslužiteljski kôd. Na poslužitelju se temelji kôd koji može oblikovati dinamički web sadržaj, dok je web stranica napisana u pregledniku. Prilikom pozivanja web stranice, poslužitelj izvršava kôd, temeljen na poslužitelju unutar stranice, prije nego stranicu vrati u preglednik. Pokretanjem na poslužitelju, kôd može obavljati složene zadatke, poput pristupa bazama podataka.

Postoji razlika korištenja *@model <klasa/tip>* i *@Model.<svojstvo>*. Primjer *@model* koristimo kod prijenosa modela iz kontrolera u pogled, dok se *@Model* koristi kod prikaza vrijednosti svojstva modela u pogledu. [5]

Razor je baziran na ASP.NET dizajnu i stvoren je za stvaranje web aplikacija. Sintaksa mu je vrlo slična klasičnom ASP-u i programskom jeziku PHP.

```
<ul>
@for (int i = 0; i < 10; i++) {
<li>@i</li>
}
</ul>
```

Sl. 3.5. *Razor sintaksa*

3.5.2. Layout pogledi

Pri projektiranju aplikacije, česta je pojava, u praksi, ponavljanje pojedinih dijelova stranice u većini aplikacija poput stranice izbornika, loga u zaglavlju stranice, podnožja s podacima itd. Da bi se dozvolio ispis zajedničkih dijelova stranica na jednom mjestu i da bi se promjene mogle raditi na jednom mjestu, standardno se koriste *Layout* pogledi, kojima je to osnovna funkcija.

```
<!DOCTYPE html>

<html>
<head>
    <title>@View.Title</title>
    <link href="/Content/Site.css" rel="stylesheet" type="text/css">
</head>
<body>

    <div id="header">
        <a href="/">Home</a>
        <a href="/Products">Products</a>
    </div>

    <div>
        @RenderBody()
    </div>

</body>
</html>
```

Sl. 3.6. *Primjer Layout poleda*

Za razliku od standardnih pogleda *Razor* metoda *@RenderBody*() omogućuje ugrađivanje sadržaja pogleda koji je povezan s ovim *layoutom* preko svojstva *Layout*. [5]

3.5.3. Sekcije

Jedan od načina prikazivanja dinamičkog sadržaja pogleda na posebnim mjestima u odgovarajućim *layout* pogledima je kreiranje sekcija na *layout* pogledu koje se dodaju pomoću *Razor* metode *@RenderSection(„<naziv sekcije>“).*[5]

3.5.4. Parcijalni pogledi (eng. Partial views)

U praksi je učestala situacija postojanje više pogleda koji upotrebljavaju iste dijelove HTML-a i/ili *Razor* naredbi. Pri takvoj je pojavi najbolje zajednički dio pogleda implementirati i izdvojiti na jednom mjestu kao poseban parcijalni pogled. Parcijalni se pogledi najčešće pohranjuju u mapu *~/Views/Shared* jer su često korišteni od strane više pogleda i pozivaju se preko *Razor* metode *@Html.Partial(„<naziv parcijalnog pogleda>“)*. [5]

```
<!DOCTYPE html>
<html>
<head>
    <link rel="shortcut icon" href="~/favicon.ico" type="image/x-icon"/>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - Gym</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")

</head>
<body>

    @Html.Partial("_NavBar")

    <div class="container body-content">
        @RenderBody()
        @*<hr />
        <footer>
            <p>&copy; @DateTime.Now.Year Gym</p>
        </footer>*@
    </div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

Sl. 3.7. *Primjer Partial viewa*

3.6. Pomoćne (helper) metode

„*Helper* metode generiraju HTML elelmente tzv. HTML *helper* metode. Premda se HTML elementi (npr. HTML elementi input) mogu direktno upisivati u pogled koristeći HTML sintaksu, MVC *framework* nudi čitav niz metoda koje generiraju završne HTML elemente, čime olakšava način pisanja i uređivanja u pogledima.“ [5]

Primarne HTML *helper* metode za generiranje koje se upotrebljavaju na web stranicama za interakciju s korisnicima, navedene su u tablici 3.2. koja slijedi u nastavku teksta.

Tablica 3.2. *Osnovne HTML helper metode*

Helper metoda	HTML element	Atribut	Opis
@Html.CheckBox	Input	type="checkbox"	Element za označavanje opcija
@Html.RadioButton	Input	type="radio"	Element za izbor jedne od opcija u grupi
@Html.Hidden	Input	type="hidden"	Tekst koji je skriven
@Html.Password	Input	type="password"	Teks sa prikazanim karakterima za lozinku
@Html.TextBox	Input	type="text"	Tekst u jednoj liniji
@Html.TextArea	textarea	-	Tekst u više linija
@Html.DropDownList	select	-	Padajuća lista za odabir
@Html.ListBox	select	type="multiple"	Lista sa mogućnosti višestrukog odabira

3.7. Validacije

Prilikom kreiranja aplikacije važno je osigurati točnost, preciznost i konzistentnost podataka, a to nastojanje naziva se validacije podataka. S obzirom da su podaci u aplikaciji MVC opisani modelom koji povezujemo (eng. Bind) u metodama kontrolera i/ili prosljeđujemo u poglede, taj bi se pojam validacije mogao preciznije definirati kao validacija modela. Validacija podataka sadrži različite provjere podataka, od jednostavnih, poput ispravnosti formata unesenog podatka, pa do složenih poslovnih pravila koja su definirana kroz više podataka, međusobno povezanih u nekoj matematičkoj relaciji. Primjer za jednostavnu validaciju su: tekst koji ne smije imati više od n znakova, datum koji ne smije biti u prošlosti i slično. Validacija se može implementirati na poslužiteljskoj i na klijentskoj strani. Direktna validacija modela podrazumijeva način validacije podataka unutar samog kontrolera na temelju povezanog modela i to tako da se kodira svako validacijsko pravilo.

Iako poslužiteljska validacija čini glavni i značajan način provjere podataka, svako slanje podataka prema poslužitelju iziskuje određeni protok podataka po mreži u smislu vremena,

količine podataka i obrade podataka na poslužitelju. Stoga, ako se primjerice radi o aplikaciji koja se nalazi na internetu i ima veći broj korisnika koji istovremeno upotrebljavaju aplikaciju, nikako nije zanemariv utrošak resursa koji su potrebni za izvršavanje validacije. U navedenom slučaju bi bilo dobro da se validacija napravi u okviru samog preglednika, korištenjem klijentskih tehnika programiranja, odnosno upotrebom funkcija JavaScript i JQuery. [5]

```
public class LoginViewModel
{
    [Required]
    [Display(Name = "Email")]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Lozinka")]
    public string Password { get; set; }

    [Display(Name = "Zapamti me?")]
    public bool RememberMe { get; set; }
}
```

Sl. 3.8. *Primjer validacije*

3.8. ADO.NET i Entity Framework

U novije doba, većina aplikacija upotrebljava neki oblik pohrane podataka. Neke od renomiranih relacijskih baza podataka su danas, najčešće korišteni, tip skladišta podataka. Njihova je glavna funkcija trajna pohrana podataka koje je unio korisnik aplikacije ili podataka uvezenih iz nekog vanjskog sustava. „U kontekstu web aplikacija, baze podataka služe za pohranu podataka o anketama, proizvodima, korisnicima sustava, kupcima itd. drugim riječima, bez skladišta podataka nemoguće je napraviti ozbiljnu poslovnu aplikaciju.“ [5]

3.8.1. ADO.NET

Microsoft u .NET *frameworku* ima uključen set klasa koje omogućuju programerima konzumaciju podatkovnih servisa od strane baza podataka u svrhu omogućavanja aplikaciji komunikacije s bazama podataka. ADO.NET objekti model nudi pregršt mogućnosti za stvaranje distribuiranih aplikacija koje razmjenjuju gotovo svaku vrstu podataka. ADO.NET omogućava dosljedan pristup skladištima podataka kao što su MS SQL Server, SQL Server Express i XML

datoteke. Uz to, jedan od glavnih obaveza objektnog modela je odvajanje pristupa od manipulacije, pomoću diskretnih sastavnica koje se mogu upotrebljavati odvojeno ili u tandemu. [5]

3.8.2. Entity Framework

Kod Entity Frameworka, klasa u domeni naziva se entitet i uključena je poput svojstva tipa *DbSet<Tentity>* u izvedenom kontekstnom razredu. Entity Framework API mapira sve pojedine entitete u tablicu, a svako pojedino svojstvo predstavlja stupac u tablici baze. Na primjeru slika 3.9., 3.10., i 3.11. u nastavku, *Student*, *StudentAddress* i *Grade* su klase domene u aplikaciji.

```
public class Student
{
    public int StudentID { get; set; }
    public string StudentName { get; set; }
    public DateTime? DateOfBirth { get; set; }
    public byte[]  Photo { get; set; }
    public decimal Height { get; set; }
    public float Weight { get; set; }

    public StudentAddress StudentAddress { get; set; }
    public Grade Grade { get; set; }
}
```

Slika 3.9. *Klasa Student*

```
public partial class StudentAddress
{
    public int StudentID { get; set; }
    public string Address1 { get; set; }
    public string Address2 { get; set; }
    public string City { get; set; }
    public string State { get; set; }

    public Student Student { get; set; }
}
```

Slika 3.10. *Klasa StudentAddress*

```
public class Grade
{
    public int GradeId { get; set; }
    public string GradeName { get; set; }
    public string Section { get; set; }

    public ICollection<Student> Students { get; set; }
}
```

Slika 3.11. *Klasa Grade*

Prethodno navedene klase postaju entiteti kad se uključe kao *DbSet<Tentity>* svojstva u kontekstnoj klasi, koja proizlazi iz *DbContext*, kao što je prikazano na slici 3.12.

```
public class SchoolContext : DbContext
{
    public SchoolContext()
    {

    }

    public DbSet<Student> Students { get; set; }
    public DbSet<StudentAddress> StudentAddresses { get; set; }
    public DbSet<Grade> Grades { get; set; }
}
```

Slika 3.12. *SchoolContext*

U gore navedenoj kontekstnoj klasi *Student*, *StudentAddress* i *Grade* su svojstva tipa *DbSet<Tentity>* i nazivaju se skup entiteta. Entitet može uključivati dvije vrste svojstva, a to su skalarno i svojstvo navigacije. [6]

3.8.2.1. Skalarno svojstvo

Primitivna svojstva tipa nazivaju se skalarna svojstva. Skalarno svojstvo pohranjuje stvarne podatke i mapira se u jedinstveni stupac u tablici baze podataka. [6]

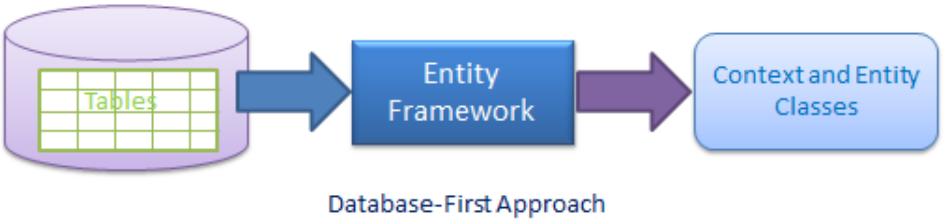
3.8.2.2. Navigacijsko svojstvo

Svojstvo navigacije predstavlja odnos prema drugom entitetu. Dvije su vrste navigacijskih svojstava, referentna navigacija i navigacija kolekcije.

U slučaju kada entitet uključuje svojstvo entitetskog tipa, naziva se referentni objekt za navigaciju i predstavlja vezu više na prema jedan. Ukoliko entitet sadrži svojstvo kolekcijskog tipa, naziva se navigacija kolekcije i predstavlja vezu više na prema više.

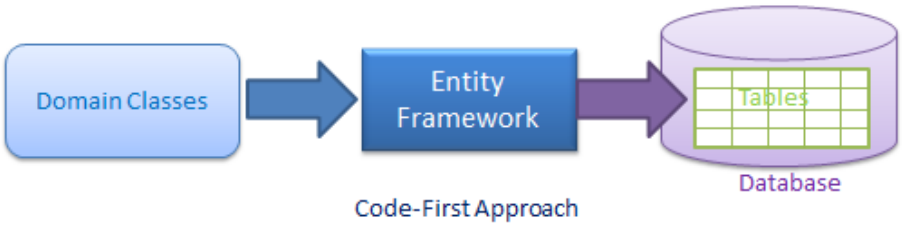
Upotrebom Entity Frameworka, tri su različita pristupa koja se koriste pri izradi web aplikacije, a to su *Database-first*, *Code-first* i *Model-first.*

U *Database-first* pristupu baze podataka generira se kontekst i entiteti postojeće baze podataka, pomoću *EDM wizard-a* koji je integriran unutar Visual Studia za izvršavanje Entity Framework naredbi kao što prikazuje i slika 3.13. u nastavku.



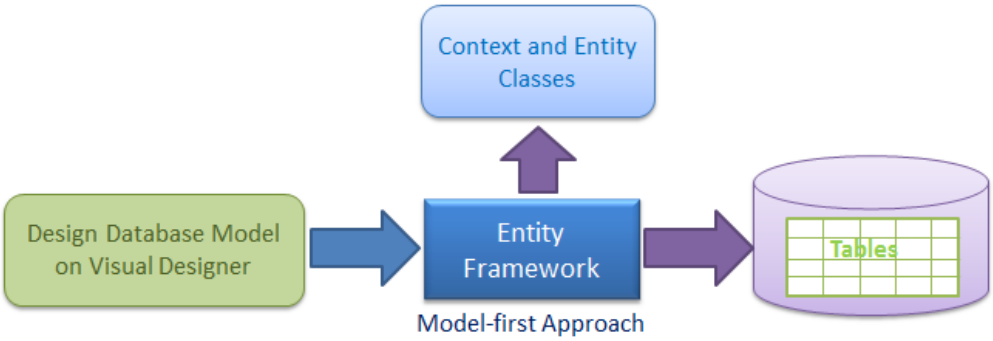
Slika 3.13. *Database-first pristup*

U slučaju nepostojanja postojeće baze podataka, preporučljivo je koristiti *Code-first* pristup za generiranje baze, stvarajući entitete i kontekstne klase iz kojih se generira baza podataka na temelju stvorenih klasa, pomoću naredbi za migraciju.



Slika 3.14. *Code-first pristup*

U *Model-first* pristupu stvaraju se entiteti, relacije i nasljedne hijerarhije direktno na vizualnom dizajneru koji je ugrađen unutar Visual Studia. Nakon toga se generiraju entiteti, kontekstne klase i skripta baze podataka iz vizualnog modela. [6]



Slika 3.15. *Model-first pristup*

3.8.3. LINQ

„LINQ (eng. Language Integrated Query) uveden je u Visual Studio 2008. godine s ciljem da omogući upite prema kolekcijama objekata koji su mapirani na neki objekt u relacijskoj bazi podataka pomoću Entity Frameworka. Svaka tablica baze podataka je predstavljena zasebnom klasom unutar konceptualnog modela te se sva komunikacija između klase i stvarne tablice

odvija u potpunosti automatizirano, u pozadini. LINQ *to* SQL u pozadini prevodi LINQ upit u Transact SQL te ga šalje bazi podataka na izvršavanje. Kada baza podataka vrati rezultat, LNIQ *to* SQL taj isti rezultat prevodi natrag u objekt aplikacije.“ [5]

Postoje dvije vrste LINQ sintakse, a to su *Lamda* i *Query* sintaksa prikazane na slici 3.16. u nastavku.

```
var vendorQuery = from v in vendors
                  where v.CompanyName.Contains("Toy")
                  orderby v.CompanyName
                  select v;


var vendorQuery = vendors
              .Where(v => v.CompanyName.Contains("Toy"))
              .OrderBy(v=> v.CompanyName);
```

Sl. 3.16. *Prikaz Query i Lamda sintakse*

3.9. WEB Api

ASP.NET web API (eng. Application Programming Interface) je okvir za izradu HTTP servisa koji mogu biti korišteni od strane velikog broja različitih klijenata kao što su preglednici, pametni telefoni, tableti i sl. U principu, sve uređaje koji podržavaju komunikaciju putem protokola HTTP. Web servise i web API-je možemo gledati kao web aplikacije koje nemaju vlastito korisničko sučelje (eng. User Intefrace), nego sadrže jedino logiku za dohvat i manipulaciju nad podacima. U tom je slučaju kreiranje prepušteno klijentskim aplikacijama. Rezultat takvog načina je modularni sustav koji ima bolje mogućnosti skaliranja te se može obuhvatiti više različitih, istom programskom logikom. Koristeći web servis i web API može se stvoriti zajednička komponenta za dohvat i manipulaciju nad podacima te se za svaku vrstu uređaja može uraditi samo posebno korisničko sučelje, umjesto da se za svaku vrstu uređaja posebno piše kompletna aplikacija s podacima i prilagođenim sučeljem.

Web API je *open source framework* za izgradnju servisa RESTful (eng. REpresentational State Transfer) pomoću tehnologija .NET. [5]

REST je arhitekturni stil koji omogućuje set standarda između sustava na webu što im olakšava međusobnu komunikaciju. REST sustavi često nazivani i RESTful sustavi, okarakterizirani su kao sustavi koji ne čuvaju stanje (eng. Stateless) što je i temeljni princip REST-a jer server nikada ne bi trebao čuvati podatke o klijentu.

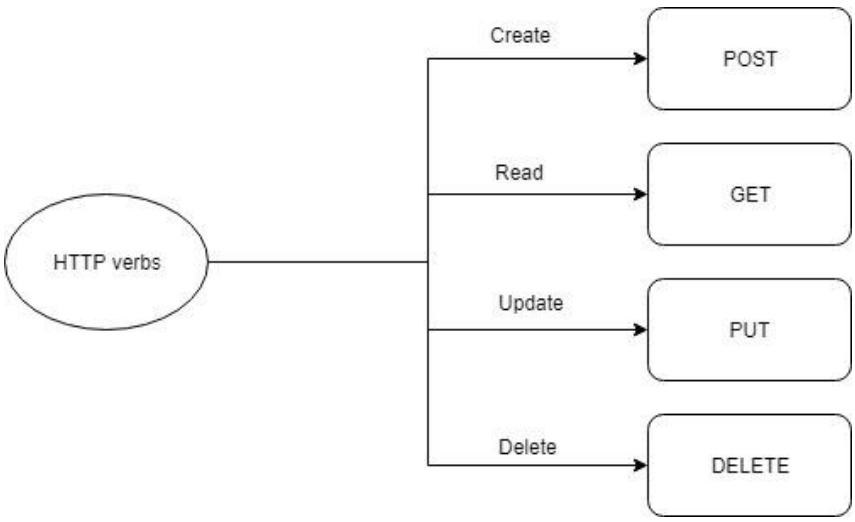
Implementacije klijenta i poslužitelja mogu se obavljati samostalno bez da jedan zna bilo što o drugome i obratno što znači da se kôd na strani poslužitelja može promijeniti u bilo kojem trenutku bez utjecaja na rad poslužitelja, dok se kôd na strani poslužitelja može se promijeniti bez utjecaj na rad klijenta.

Sve dok su obje strane upoznate s vrstom formata poruke koju si međusobno šalju, mogu se čuvati modularno i odvojeno. Takvo razdvajanje korisničkog sučelja od pohrane podataka uvelike poboljšava fleksibilnost sučelja preko platformi i njegovu skalabilnost. Jedna od prednosti je i mogućnost odvojenog razvijanja.

S obzirom da su REST sustavi *stateless,* poslužitelj i klijent mogu razumjeti svaku primljenu poruku, čak i bez vidljivih prethodnih poruka. Takvo ograničenje se provodi korištenjem resursa umjesto naredbi. Resursi predstavljaju informacije koje će trebati pohraniti ili slati drugim uslugama.

Budući da REST sustavi komuniciraju putem standardnih operacija nad resursima, ne oslanjaju se na implementaciju sučelja. Takva ograničenja pomažu prijelaznim aplikacijama da postignu pouzdanost, brzu izvedbu i skalabilnost kao komponente kojima se može upravljati, ažurirati i ponovno koristiti bez utjecaja na sustav u cijelosti, čak i za vrijeme rada sustava.

Kod REST-a korisnici šalju zahtjeve za preuzimanjem ili izmjenom resursa, a poslužitelji šalju odgovore na te zahtjeve. Standardne načine za slanje zahtjeva i primanje odgovora čine HTTP ključne riječi u kombinaciji s URI-em (eng. Uniform Resource Identifier).



Sl. 3.17. *CRUD akcije*

3.9.1. GET

Ključnu riječ GET upotrebljavamo pri dohvatu podataka i to u JSON (eng. JavaScript Object Notation) obliku. Ovakvi zahtjevi su sigurni jer ne mijenjaju resurse. Ukoliko ovakav tip zahtjeva prođe bez greške, server će odgovoriti s HTTP statusom 200 *„OK“*. Ukoliko URI potražuje resurs koji ne postoji, server će vratiti HTTP status *„Not found“*. Ukoliko status nije dobar, server vraća status 400 *„Bad request“*.

3.9.2. POST

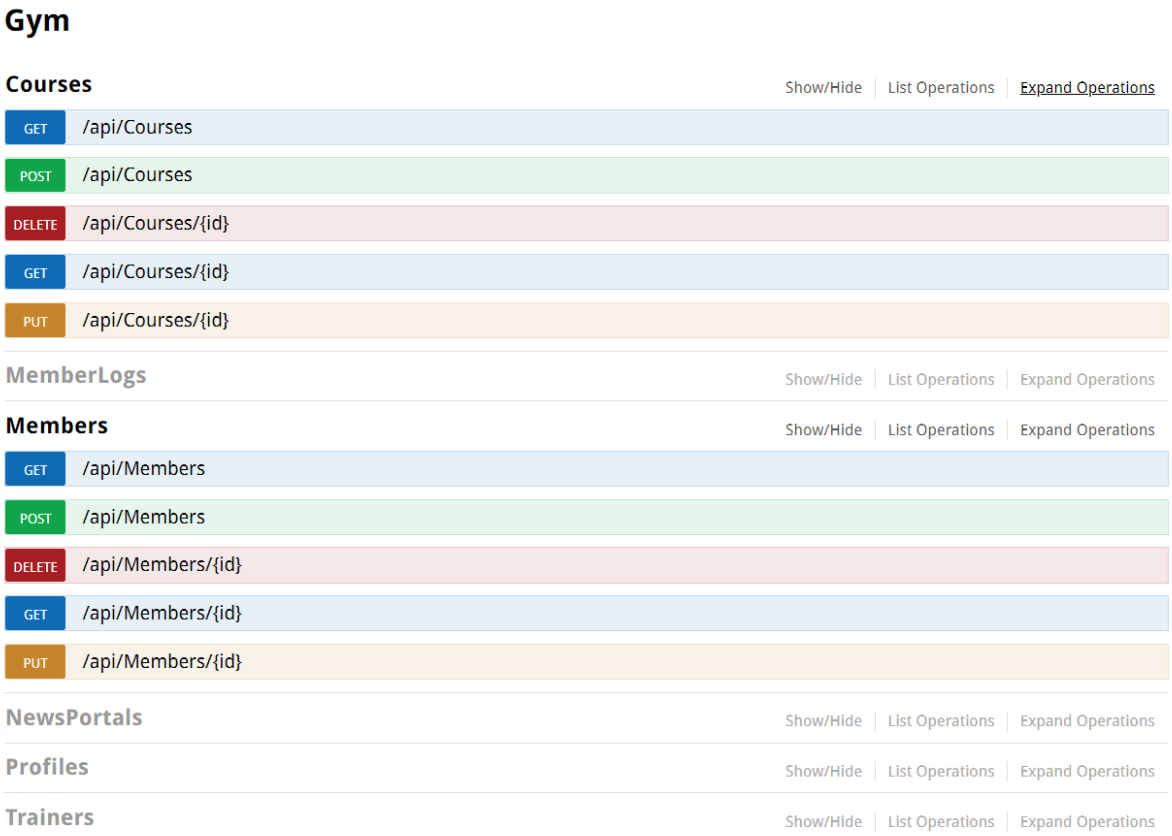
Zahtjev sa ključnom riječi POST upotrebljavamo za kreiranje novog resusrsa. Podaci novog resusra se šalju na server i ako je sve u redu, server odgovara statusom 201 *„CREATED“*.

3.9.3. PUT

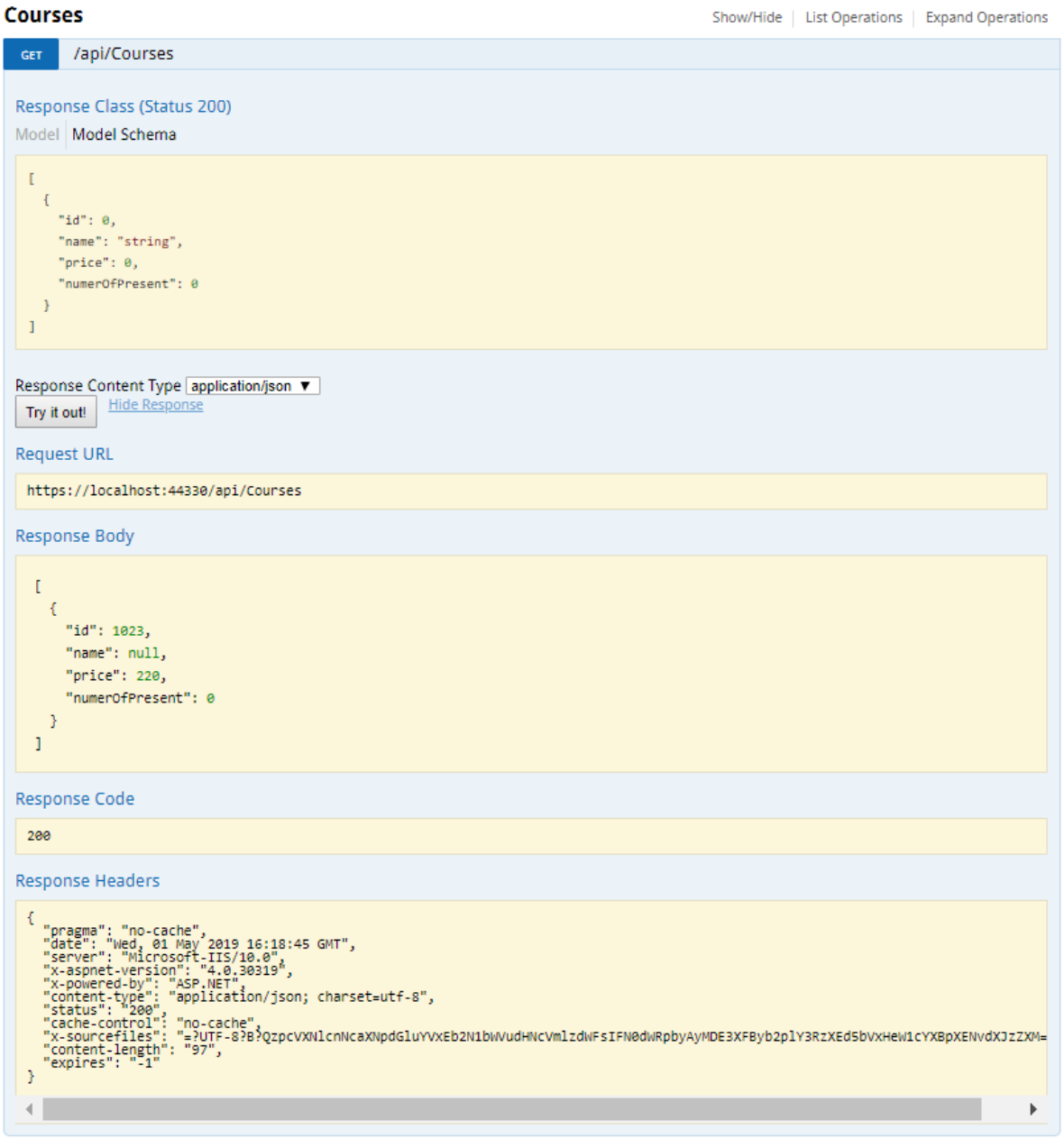
PUT se upotrebljava prilikom izmjene resursa. URI označava resurs koji je potrebno izmijeniti, a tijelo zahtjeva nove vrijednosti resursa. Mogući odgovori koje server vraća prema klijentu su 200 *„OK“* ili 204 *„No content“*.

3.9.4. DELETE

DELETE je prilično lako za razumjeti, a upotrebljava se za brisanje resursa identificiranog pomoću URI-a. Nakon uspješnog brisanja, server vraća HTTP status *„OK“* zajedno s tijelom odgovora ili HTTP status 204 *„No content“* kada odgovor ne sadrži traženi resurs.



Sl. 3.18. *Primjer CRUD akcija koristeći Swagger*



Sl. 3.19. *Primjer Uspješnog dohvata resursa u JSON formatu*

3.10. URL usmjeravanje

Zahtjev od preglednika prvo dolazi do komponente usmjeravanja, koja je ugrađena u sam MVC *framework* i implementirana u obliku klase *UrlRoutingModule*, prije nego dođe do samog kontrolera unutar aplikacije. Zadaća tog modula je potraživanje prve zadovoljavajuće rute s popisa iz zahtjeva koji je došao od preglednika prema aplikaciji. Pod rutom se podrazumijeva URL uzorak koji je mapiran na određeni kontroler i odgovarajuću akciju u njemu. Odluku koji URL odgovara kojem kontroleru i kojoj akcijskoj metodi, modul donosi na temelju unaprijed

definiranih pravila. Prilikom pokretanja MVC aplikacije, izvršava se metoda *Application_Start()* definirana u datoteci *Global.asax* unutar koje se poziva statička metoda za registraciju (izgradnju) liste svih raspoloživih ruta unutar aplikacije. Ta statička metoda se zove *RegisterRoutes()* i nalazi se u klasi *RouteConfig*. Metoda se pokreće samo jednom i to prilikom pokretanja aplikacije. Registracija svih ruta unutar aplikacije se izvršava pozivom metode *MapRoute()* kao što je prikazano na slici 3.20. u nastavku.

```
public class MvcApplication : System.Web.HttpApplication
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            "Default",                                          // Route name
            "{controller}/{action}/{id}",                       // URL with parameters
            new { controller = "Home", action = "Index", id = "" } // Parameter defaults
        );

    }

    protected void Application_Start()
    {
        RegisterRoutes(RouteTable.Routes);
    }
}
```

Sl. 3.20. *Metoda za registraciju ruta*

Metoda sadrži tri osnovna pravila:

1) Naziv (eng. Name) – podatak nije obavezan, ali ako je naveden mora biti jedinstven na razini aplikacije
2) URL obrazac (URL) – definicija rute (eng. URL pattern)
3) Pretpostavljena vrijednost (eng. Default) – ako URL ne sadrži vrijednosti za kontroler i akciju, MVC izvršavanje preusmjerava na pretpostavljenu vrijednost. [5]

3.11. Sigurnost aplikacije

Osiguravanje web aplikacija za web programera predstavlja vrlo složen i zahtjevan posao. Samo postavljanje sigurnosnih postavki jedne web aplikacije zahtjeva pomno planiranje za koje je nužan preduvjet dobro poznavanje mogućnosti i načina na koje je web aplikacije moguće osigurati. Sigurnost podataka je važan čimbenik u svim vrstama aplikacija, ali je kod web aplikacija dodatno naglašen jer su one najčešće dostupne svima. Sigurnosni model ASP.NET MVC oslanja se na već ugrađene mogućnosti .NET Frameworka i web poslužitelja IIS-a (eng. Internet Information Services). Da bi se ispravno zaštitilo MVC aplikaciju, potrebno je obaviti

dvije funkcije na kojima se temelji sigurnosni model svake aplikacije. Te važne dvije funkcije su:

1) Autentikacija – potvrda korisničkog identiteta
2) Autorizacija – proces utvrđivanja ovlasti prijavljenog korisnika. Kada se utvrdi identitet korisnika, mogu mu se dodijeliti određene ovlasti za rad u aplikaciji.

Kao što se u stvarnom životu identitet dokazuje osobnom iskaznicom ili putovnicom, u slučaju aplikacija on se dokazuje upisom točnog korisničkog imena i lozinke. Unutar ASP.NET MVC aplikacija postoje tri modela autentikacije korisnika, a to su:

1) Autentikacija Forms
2) Autentikacija OpenID/Oauth
3) Windows autentikacija

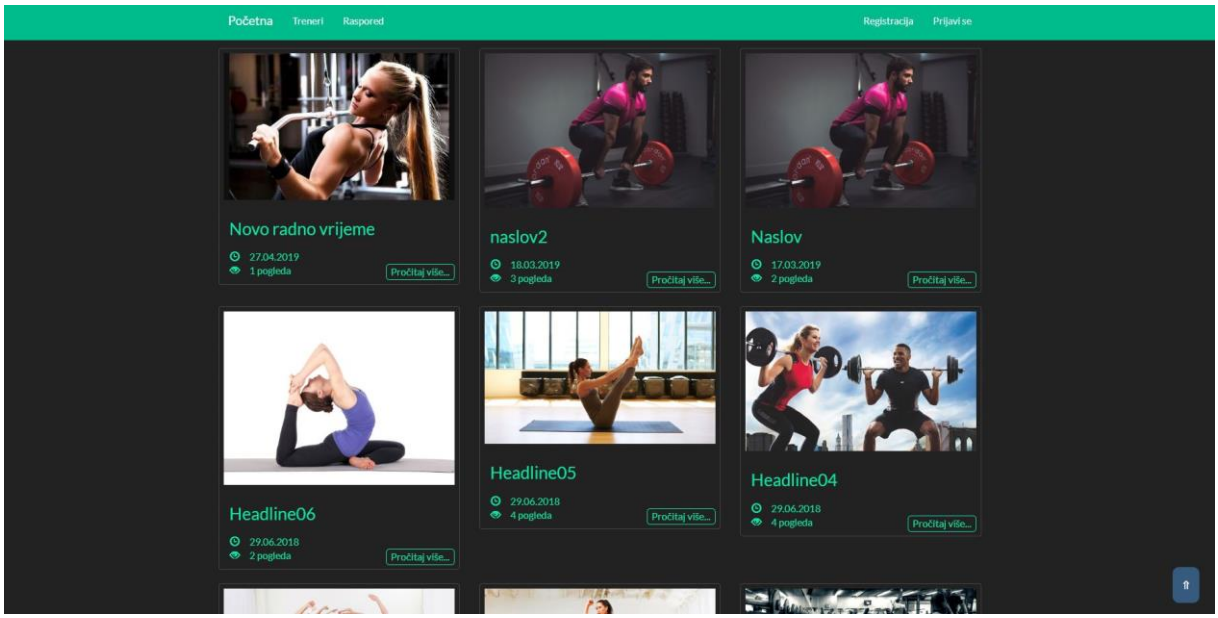
Danas je još uvijek najčešće korišteni mehanizam autentikacije *Forms* autentikacija, upravo zbog dugogodišnje prisutnosti i jednostavnosti korištenja. [5]

4. WEB APLIKACIJA

U nastavku će se opisati korištenje web aplikacije i njezine mogućnosti. Struktura web aplikacije može se koristiti kroz tri role rada: admin, trener i korisnik, a u nastavku rada slijedi objašnjenje funkcionalnosti svake role.

4.1. Uvod u aplikaciju

Prilikom pokretanja aplikacije otvara se početna stranica na kojoj su vidljive objave od strane admina web aplikacije. One predstavljaju jedan oblik komunikacije sa svim korisnicima web aplikacije.

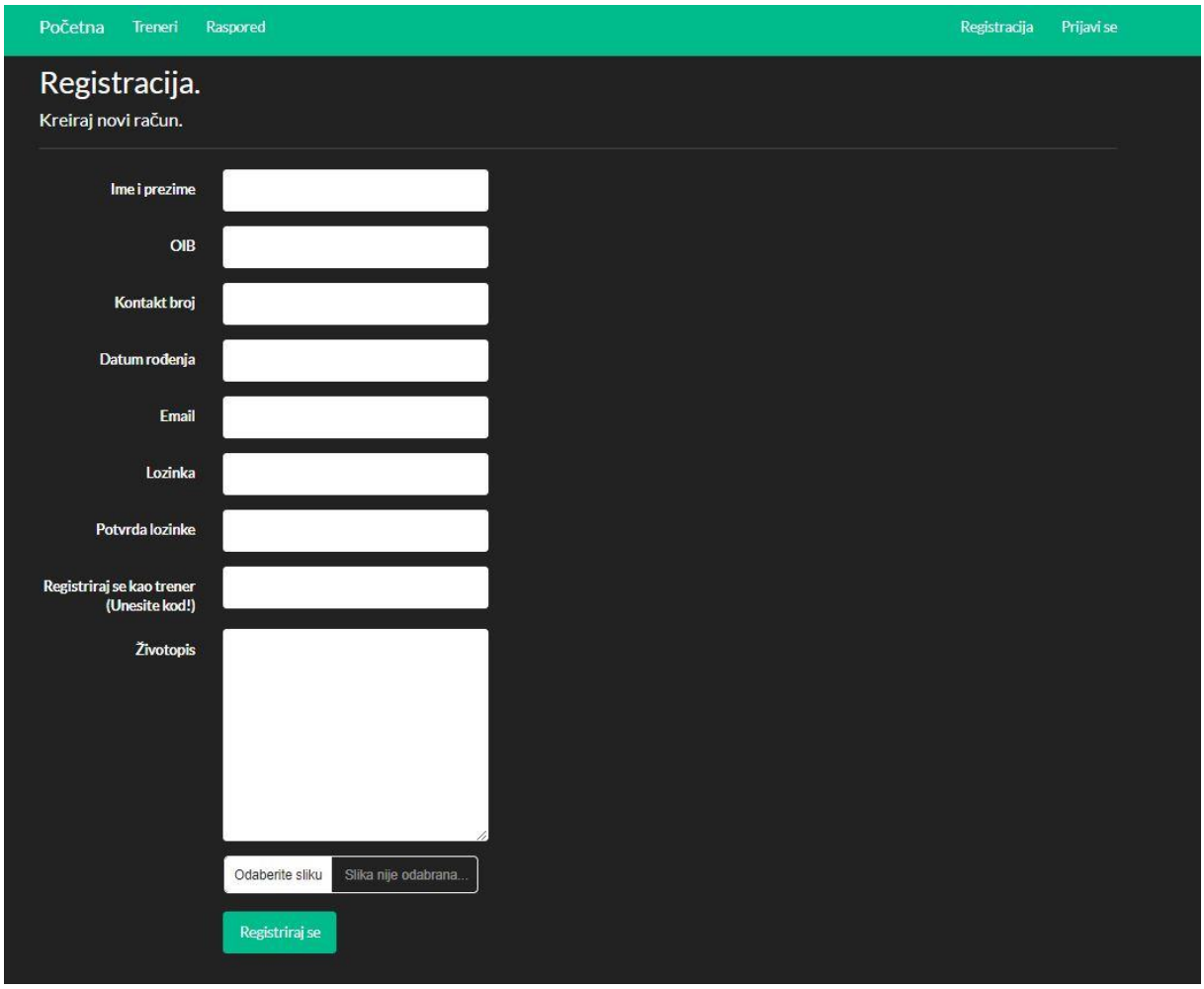


Sl.4.1. *početna stranica web aplikacije*

Prije nego li se korisnik web aplikacije registrira ili prijavi u istu, vidljive su mu novosti, treneri i raspored te mogućnosti registracije ili prijave u web aplikaciju.

4.1.1. Registracija u sustav

Registracija u sustav odvija se pomoću forme za registraciju gdje se pomoću POST metode upisani podaci šalju u bazu podataka. Navedeno je prikazano na slici 4.2. u nastavku, na kojoj su predočeni i podaci koje korisnik upisuje prilikom registracije.

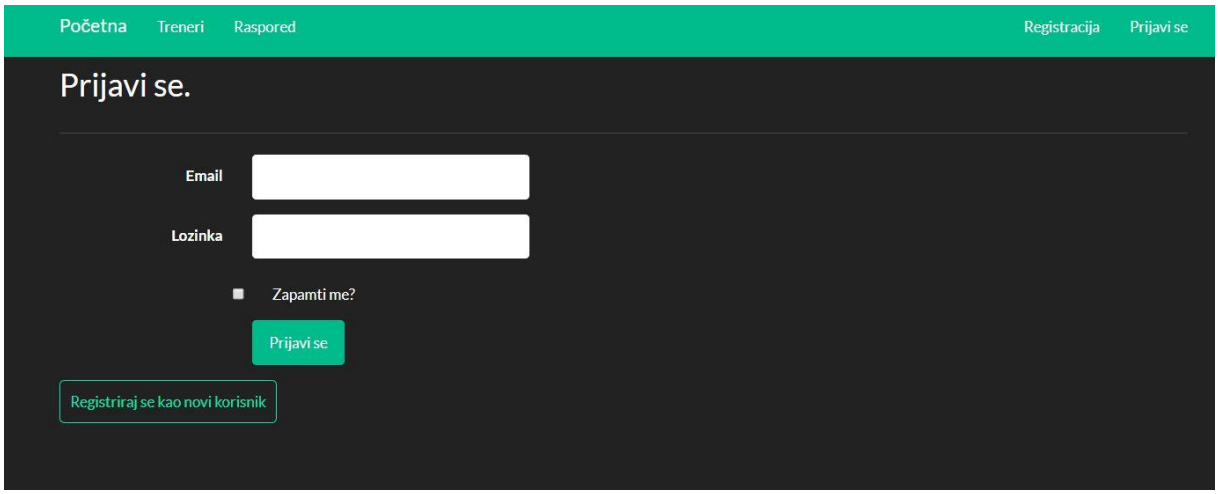


Sl. 4.2. *Forma za registraciju*

Prilikom registracije u sustav, na nekim su poljima postavljeni uvjeti koji se moraju zadovoljiti kako bi se budući korisnik uspješno registrirao. Neki od tih uvjeta su: zahtjev da OIB mora sadržavati točno 11 numeričkih znakova, email mora biti u prihvatljivom obliku, lozinka mora sadržavati minimalno jedno veliko slovo, jedan numerički znak i jedan specijalni znak. Ako neki od uvjeta nije zadovoljen, sustav će, nakon klika na gumb za registraciju, ispisati koja polja nisu pravilno unesena. Neka su polja obavezna, dok neka i nisu. Polja „Registriraj se kao trener“, „Životopis“ i polje za unos profilne slike nisu obavezni. Polje „Registriraj se kao trener“ predviđeno je za popunjavanje samo u slučaju da budući korisnik sustava želi biti trener. U tom slučaju treba upisati određeni kôd u polje kako bi sustav prepoznao koje ovlasti mu treba dodijeliti.

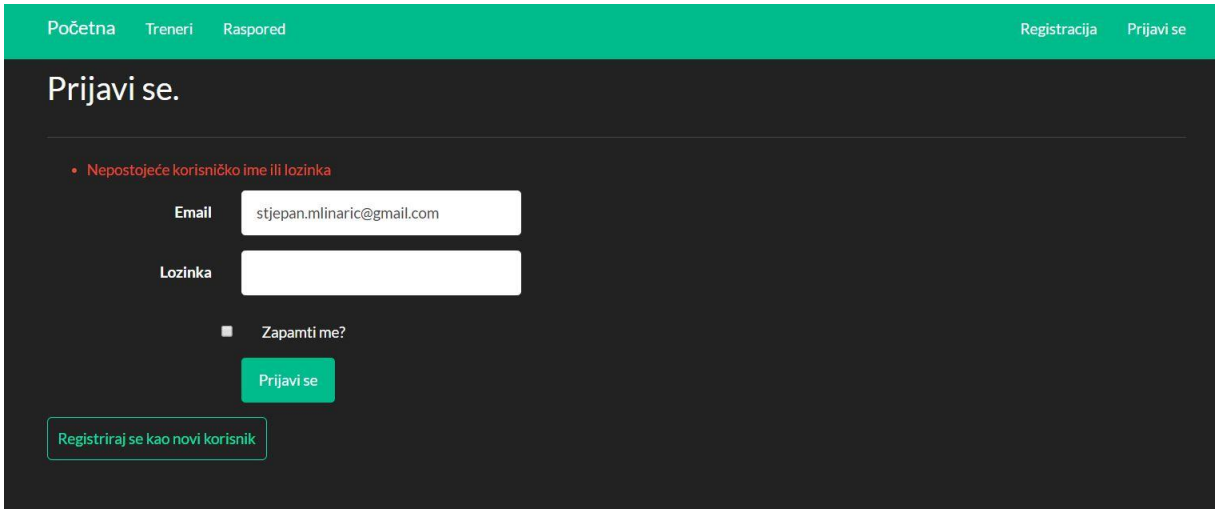
4.1.2. Prijava u sustav

Prijava u sustav odvija se pomoću forme za prijavu koja se sastoji od četiri elementa, a to su korisničko ime, lozinka, gumb za registraciju i gumb za pokretanje akcije za prijavu u sustav.



Sl. 4.3. *Forma za prijavu u sustav*

Nakon klika na gumb za prijavu, u sustavu se provjerava postoji li član s parametrima koji su poslani iz forme za prijavu. Ako se poslani podaci s forme za prijavu ne poklapaju s onima u sustavu, sustav će ispisati poruku kao na slici 4.4.



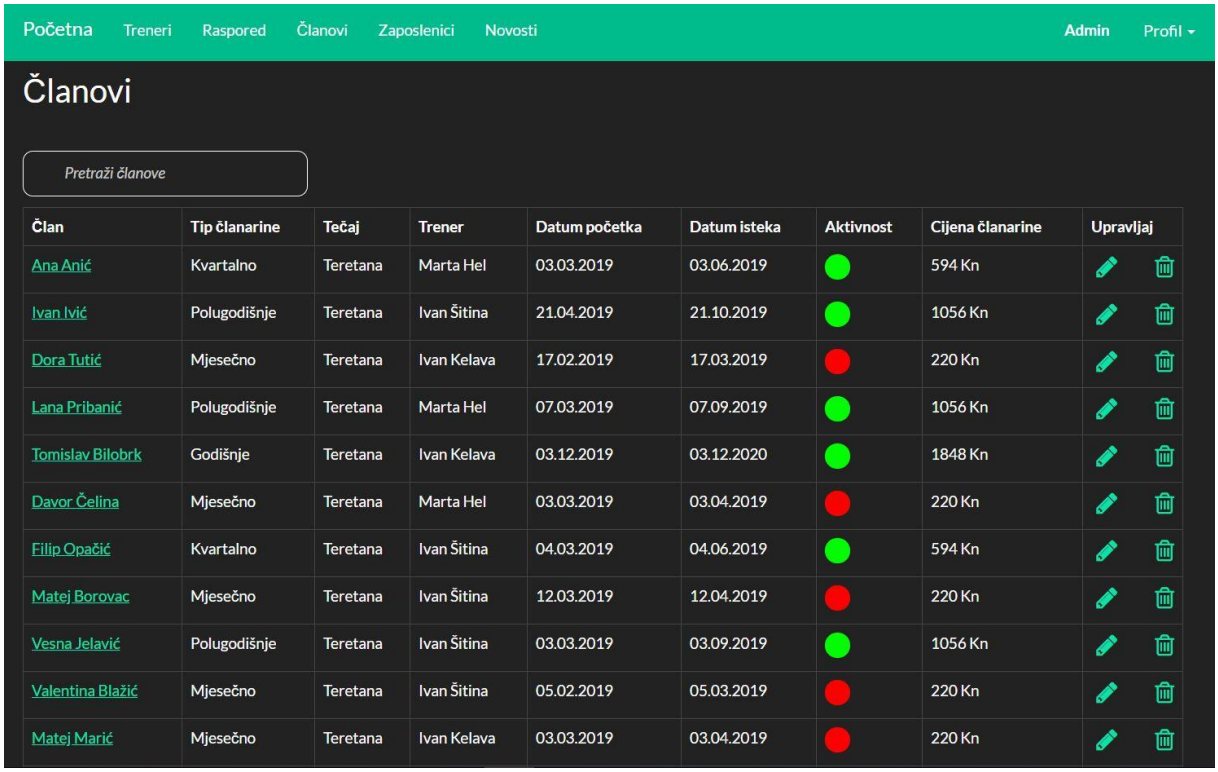
Sl. 4.4. *Neuspješna prijava u sustav*

Ako se poslani podaci s forme za prijavu poklapaju s jednim redom u tablici gdje su korisnici sustava, korisnik ulazi u aplikaciju s određenim pravima koja su mu dodijeljena.

4.2. Admin rola

Nakon prijave u sustav kao admin, otvara se početna stranica, ali i dodatne funkcionalnosti za koje samo admin sustava ima pravo gledanja i upravljanja. Adminove funkcije su: uvid u

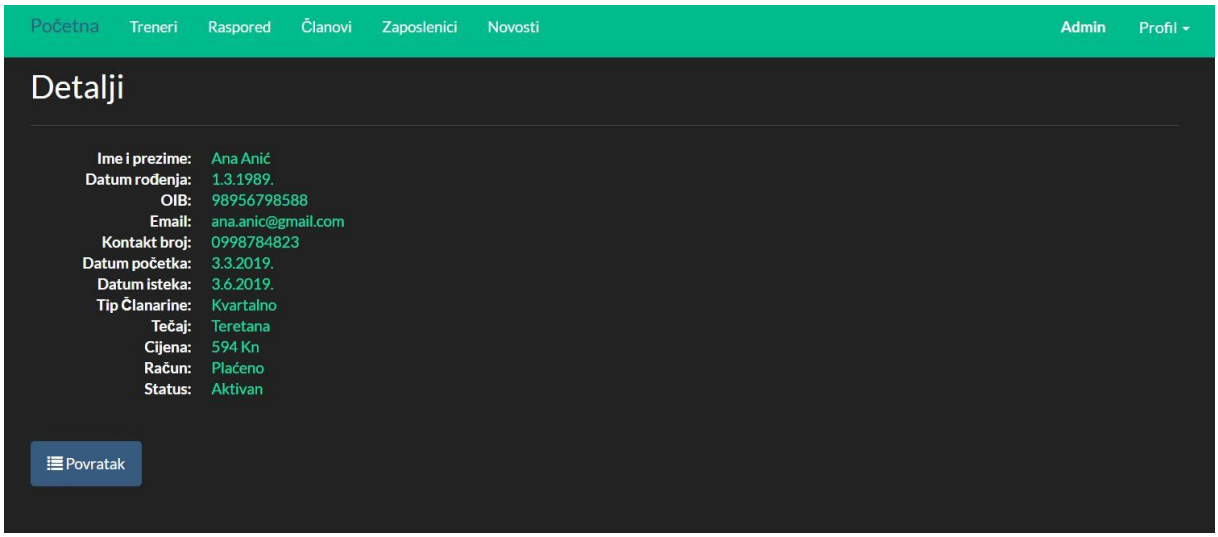
članove, zaposlenike, raspored i novosti kao i manipulacija istima. Na slici 4.5. prikazana je lista članova koje vidi samo admin.



Član	Tip članarine	Tečaj	Trener	Datum početka	Datum isteka	Aktivnost	Cijena članarine	Upravljaj
Ana Anić	Kvartalno	Teretana	Marta Hel	03.03.2019	03.06.2019		594 Kn	
Ivan Ivić	Polugodišnje	Teretana	Ivan Šitina	21.04.2019	21.10.2019		1056 Kn	
Dora Tutić	Mjesečno	Teretana	Ivan Kelava	17.02.2019	17.03.2019		220 Kn	
Lana Pribanić	Polugodišnje	Teretana	Marta Hel	07.03.2019	07.09.2019		1056 Kn	
Tomislav Bilobrk	Godišnje	Teretana	Ivan Kelava	03.12.2019	03.12.2020		1848 Kn	
Davor Čelina	Mjesečno	Teretana	Marta Hel	03.03.2019	03.04.2019		220 Kn	
Filip Opačić	Kvartalno	Teretana	Ivan Šitina	04.03.2019	04.06.2019		594 Kn	
Matej Borovac	Mjesečno	Teretana	Ivan Šitina	12.03.2019	12.04.2019		220 Kn	
Vesna Jelavić	Polugodišnje	Teretana	Ivan Šitina	03.03.2019	03.09.2019		1056 Kn	
Valentina Blažić	Mjesečno	Teretana	Ivan Šitina	05.02.2019	05.03.2019		220 Kn	
Matej Marić	Mjesečno	Teretana	Ivan Kelava	03.03.2019	03.04.2019		220 Kn	

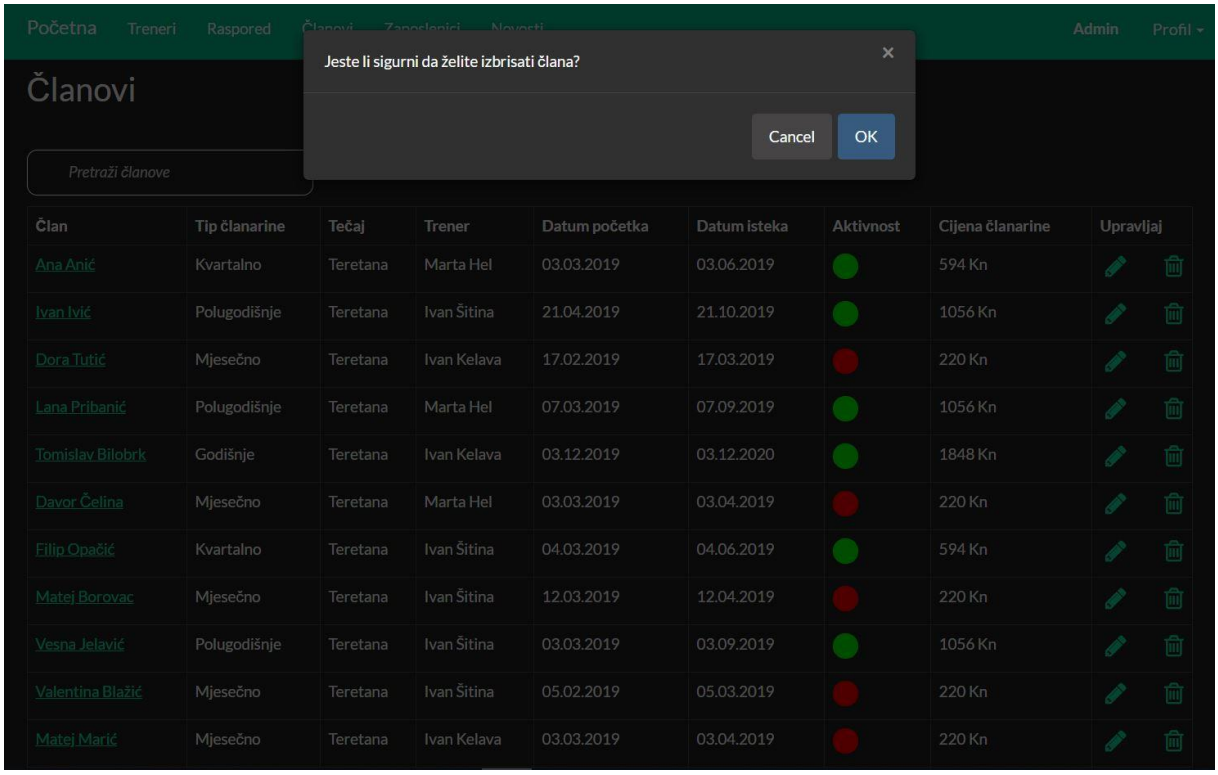
Sl. 4.5. *Lista članova teretane*

Kako se očekuje da lista, predočena u prethodnoj slici, ima više članova ugrađena je funkcionalnost pretrage članova radi brzog pristupa određenom članu. U tablici su prikazane bitne informacije o članu, poput dodijeljenog trenera, datuma plaćene članarine i isteka iste, ukupna cijena s obzirom na tip članarine i druge reference. Sve detalje o članu admin može vidjeti klikom na ime željenog člana i dobit će pregled kakav je prikazan na slici 4.6. u nastavku.



Sl. 4.6. *Detalji o članu*

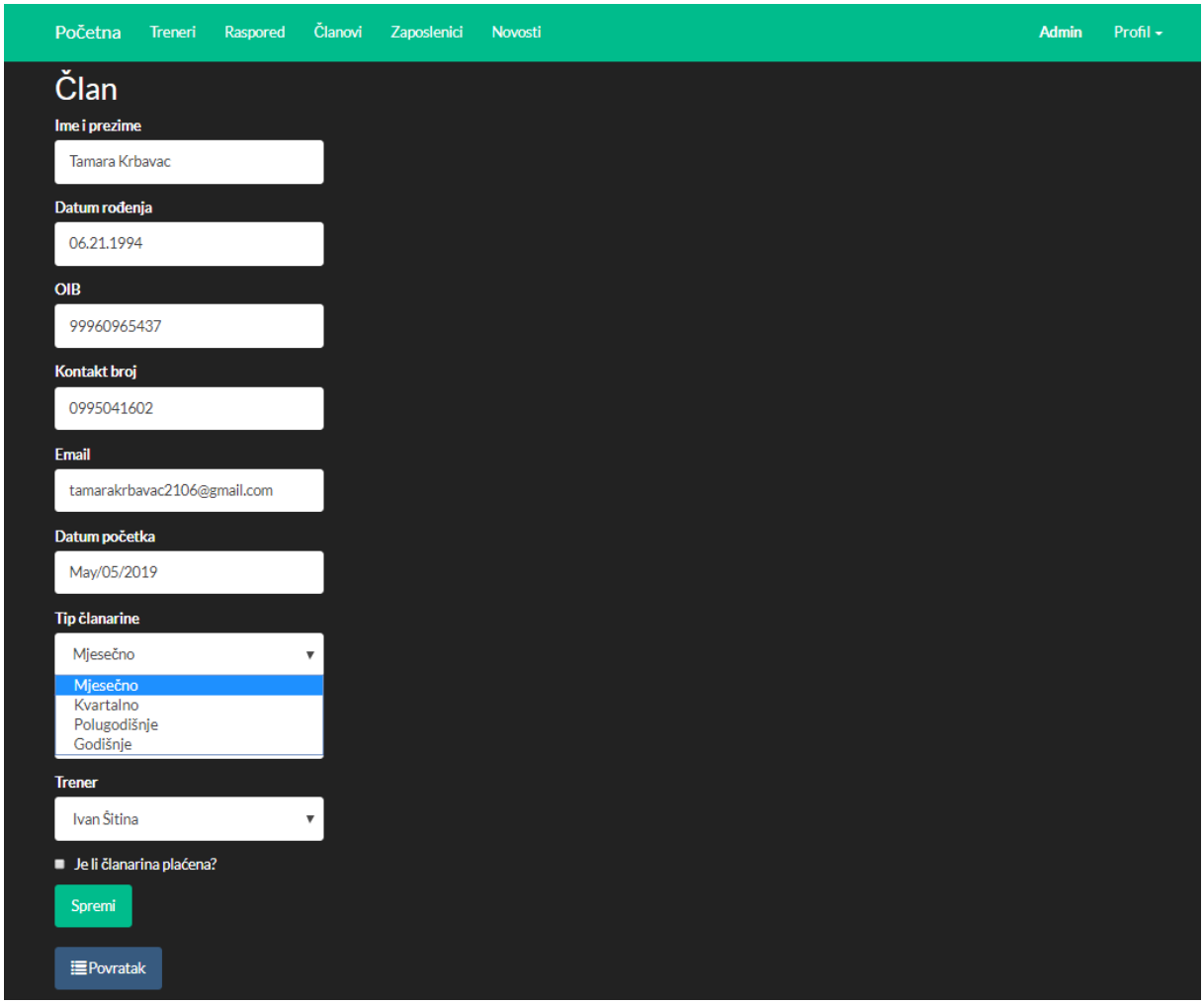
Osim uvida u članove, admin ima mogućnost i brisanja članova kao što je prikazano na slici 4.7.



Sl. 4.7. *Brisanje člana*

S obzirom da je navedena radnja vrlo osjetljiva, sustav će izbaciti poruku jesmo li sigurni da želimo to napraviti („Jeste li sigurni da želite izbrisati člana?“). Stoga će član biti izbrisan tek nakon potvrde, klikom na ponuđeni gumb OK.

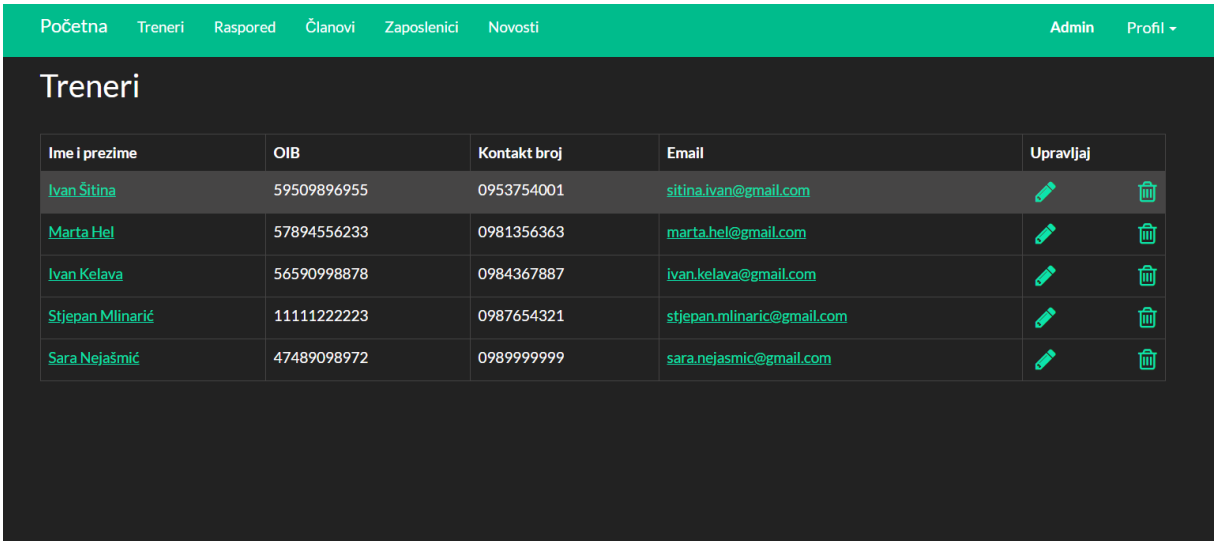
Rubrika „Aktivnost“ u tablici „Lista članova teretane“ (sl. 4.5.) označava je li korisnik trenutno član koji je platio članarinu za određeno razdoblje (zeleni kružić) ili nije (crveni kružić). Nakon isteka članarine, status aktivnosti člana prelazi u crveno, a ukoliko član ipak želi produžiti članarinu, admin je u mogućnosti to omogućiti preko forme koja je prikazana na slici 4.8. u nastavku.



Sl. 4.8. *Uređivanje statusa člana*

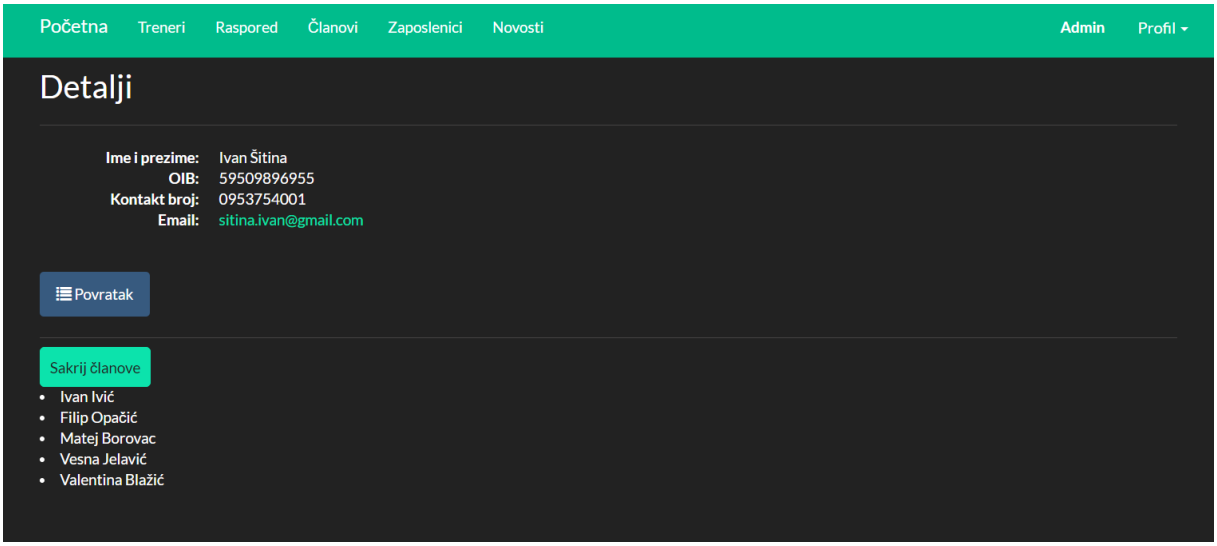
Na formi sa slike 4.8. predviđeno je da admin može članu produžiti članarinu. Upisom datuma početka članarine, automatski se računa i datum isteka članarine, a on ovisi o tipu članarine (mjesečno, kvartalno, polugodišnje, godišnje) koji je upisan. Tip članarine utječe i na ukupnu cijenu članarine. Član koji je upisan kvartalno dobiva 10%, polugodišnje 20% i godišnje 30% popusta na ukupnu cijenu tečaja.

Osim uvida u članove i manipulacije nad njihovim podacima, admin ima uvid i u zaposlenike (trenere) čija je lista vidljiva na slici 4.9.



Sl. 4.9. *Lista zaposlenika (trenera)*

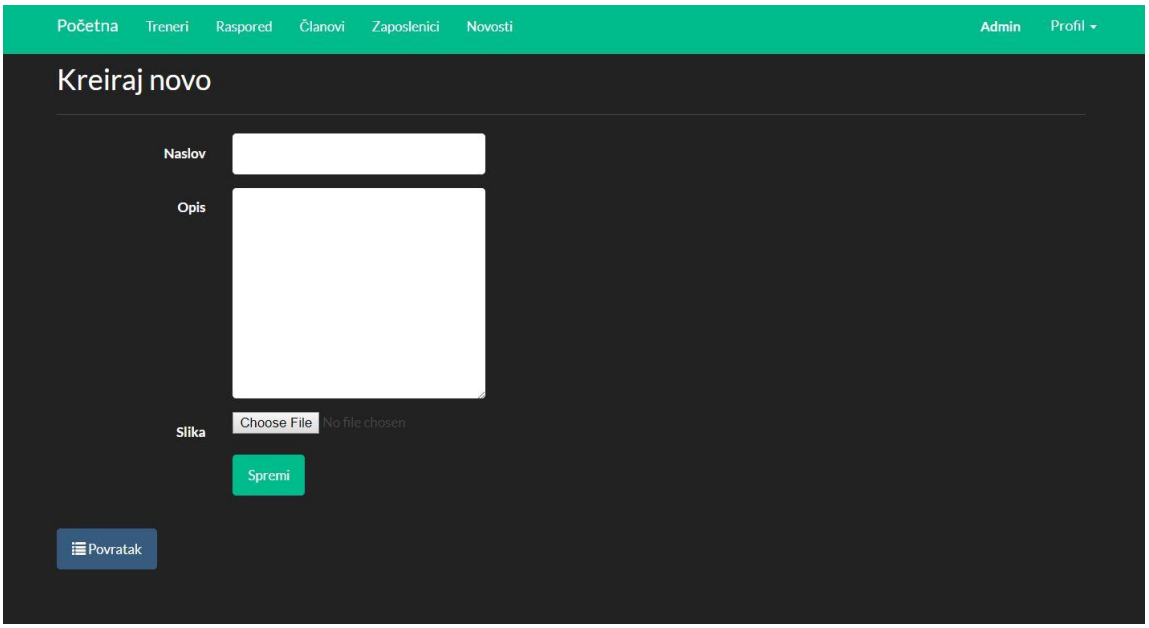
Klikom na ime trenera, otvaraju se detalji o odabranom treneru. Tim odabirom, osim općih podataka, admin ima mogućnost vidjeti i popis trenerovih članova što mu je omogućeno klikom na gumb „Pokaži članove“. Objašnjena radnja prikazana je na slici 4.10.



Sl. 4.10. *Detalji zaposlenika (trenera)*

Kao i kod članova, admin ima mogućnost uređivanja podataka i o zaposleniku kao i brisanje istih, klikom na sličicu olovke, odnosno koša za smeće u rubrici „Upravljaj“ koja se nalazi u tablici trenera.

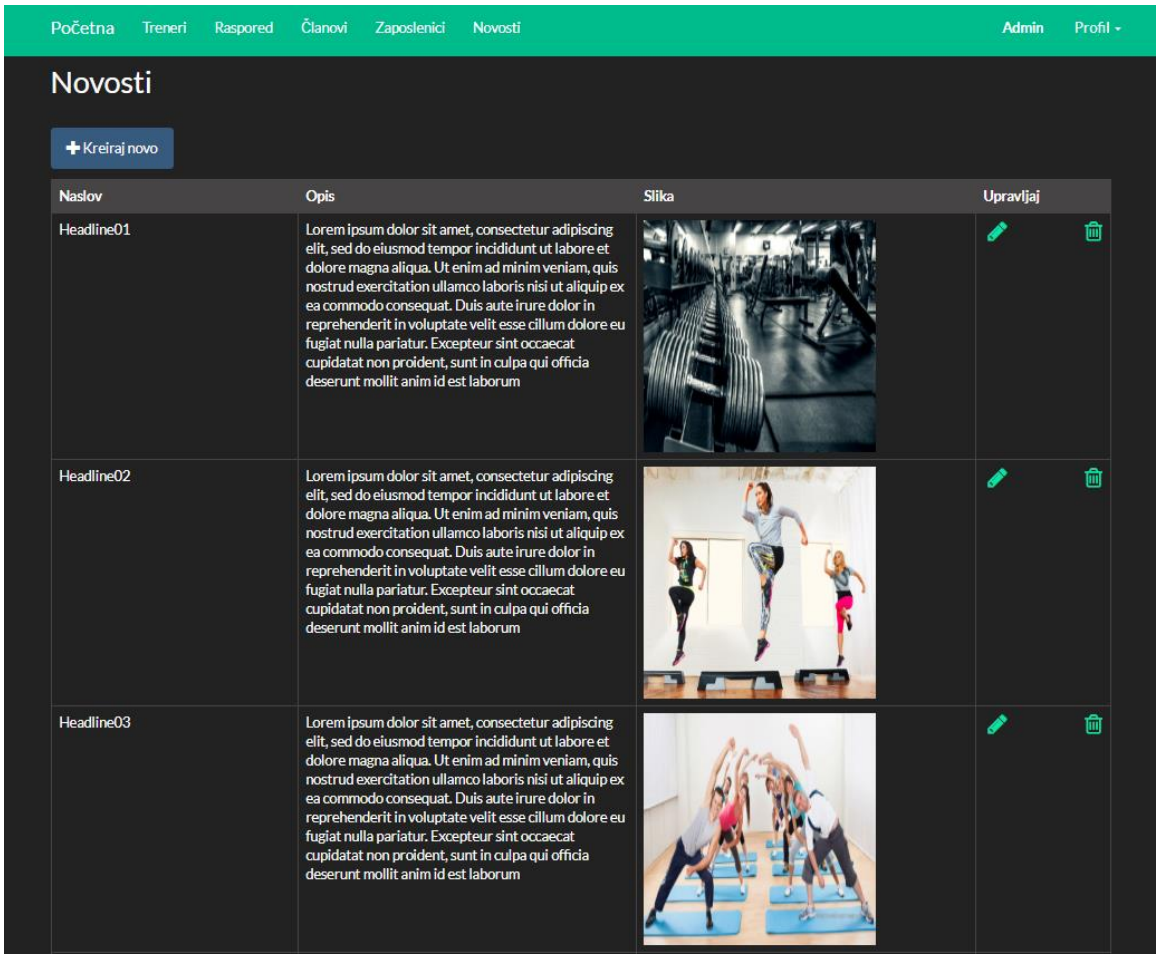
Na početku poglavlja spomenuto je da se na početnoj stranici web aplikacije nalaze novosti dodane od strane admina. Na slici 4.11. prikazana je forma gdje admin može dodati novu vijest i pridružiti sliku radi boljeg opisa.



Sl. 4.11. *Kreiranje nove vijesti*

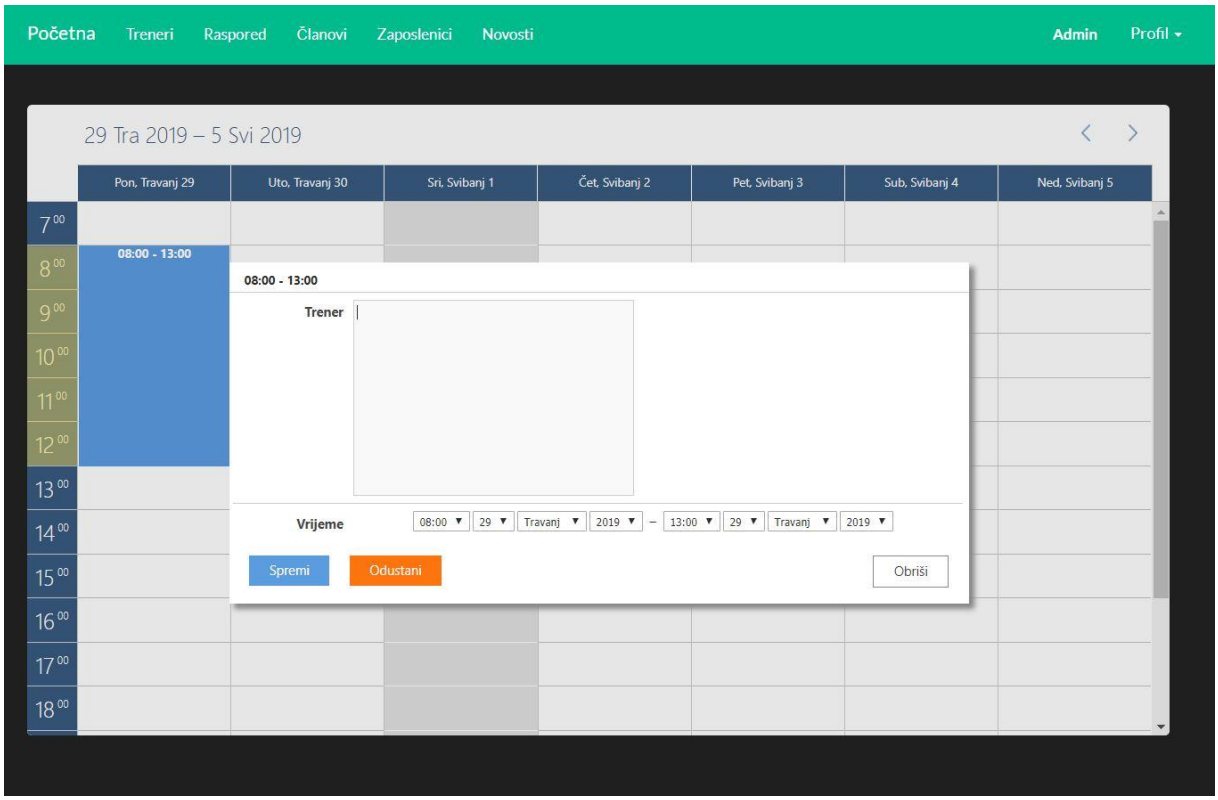
Nakon što je nova vijest kreirana, ona se prikazuje na početnoj stranici kao što je vidljivo na slici 4.1. te je osim slike i naslova vidljiv i datum kreiranja vijesti te broj pogleda. Klikom na gumb „Pročitaj više…“ otvara se širi opis u kojem pišu detalji objave.

Kao kod članova i zaposlenika, admin također ima mogućnost uređivanja i brisanja novosti kojima može pristupiti preko sličica olovke i koša za smeće što je vidljivo na slici 4.12.



Sl. 4.12. *Tablica novosti*

Admin za svakog zaposlenika (trenera) ima mogućnost upisivanja radnih sati u tablicu tjednog rasporeda.

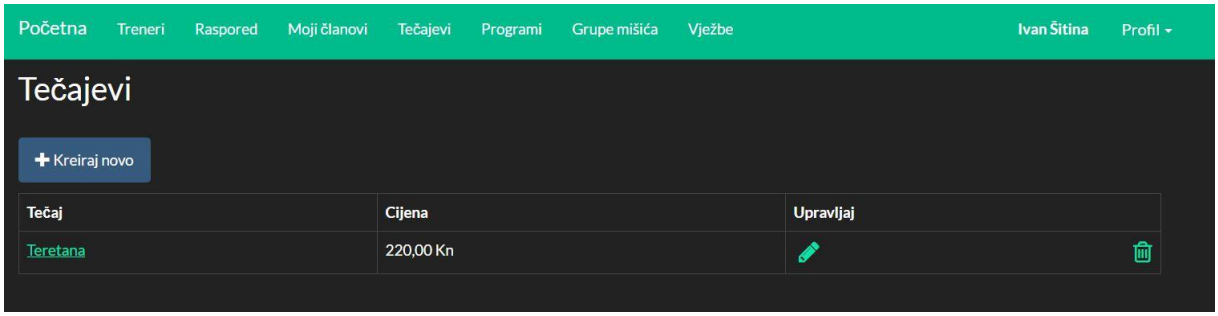


Sl. 4.13. *Tjedni raspored*

Na slici 4.13. prikazan je raspored rada kao i modalni prozor gdje admin dodaje radno vrijeme trenera na određeni dan u tjednu zadan u tablici. U rasporedu je vidljivo na koji tjedan se raspored odnosi, radna vremena trenera po danu i satu, a postoji mogućnost i uvida u tjedne koji su prošli i koji nadolaze, pomoću strelica koje se nalaze u gornjem desnom kutu rasporeda.

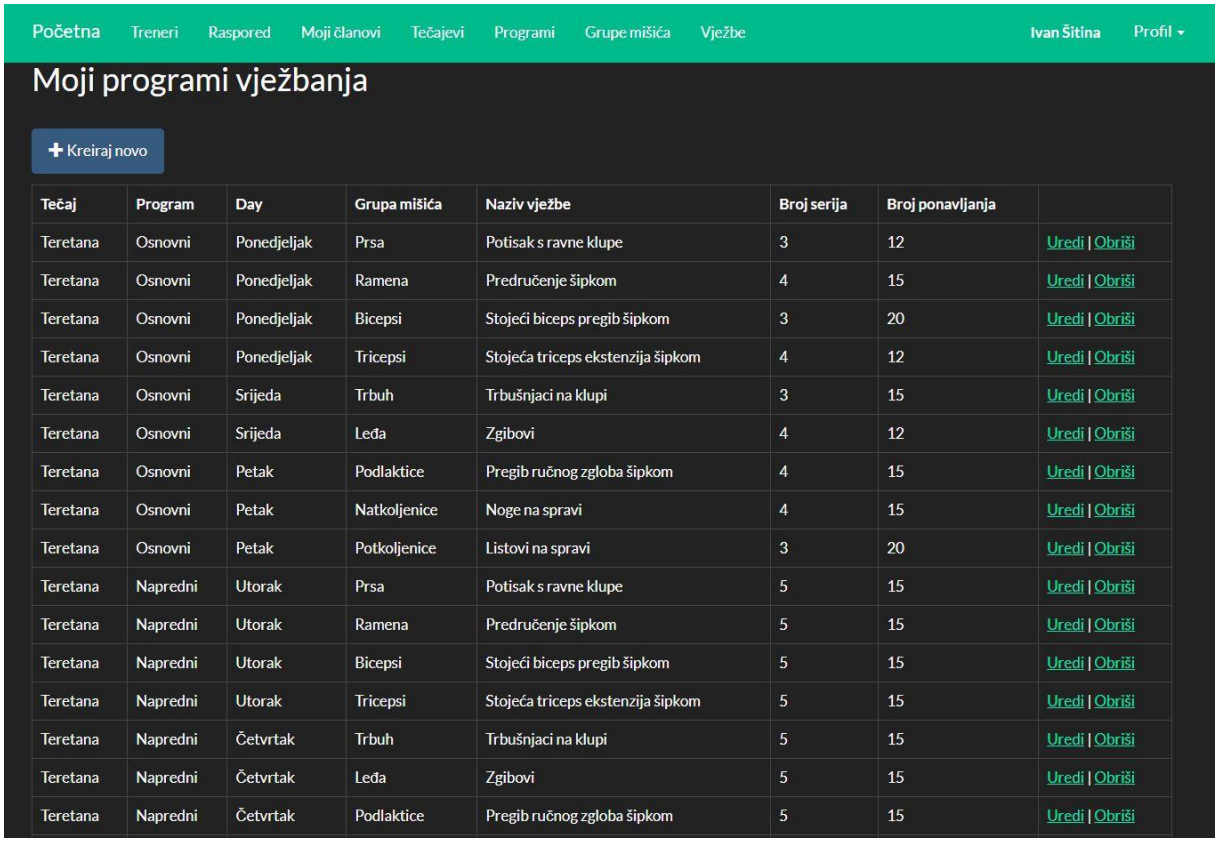
4.3. Trener rola

Nakon prijave u sustav, u roli trenera, otvara se početna stranica, ali i dodatne funkcionalnosti koje su potrebne trenerima za rad u sustavu. Trener ima pravo vidjeti i upravljati tablicama u koje su zapisani tečajevi, programi, grupe mišića i vježbe.



Sl. 4.14. *Lista tečaja*

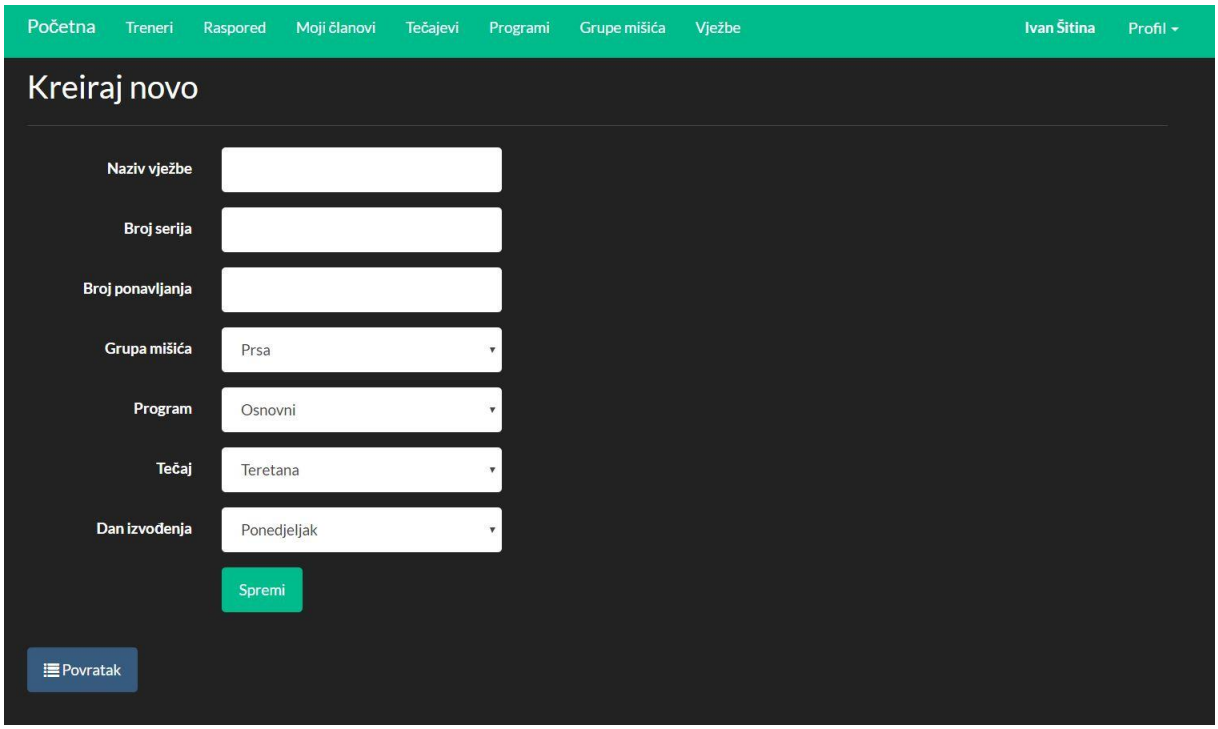
Na slici 4.14. prikazana je lista tečaja koje vidi samo korisnik s trener rolom. Trener može brisati, uređivati, ali i kreirati nove tečaje ako je to potrebno.



Tečaj	Program	Day	Grupa mišića	Naziv vježbe	Broj serija	Broj ponavljanja	
Teretana	Osnovni	Ponedjeljak	Prsa	Potisak s ravne klupe	3	12	Uredi | Obriši
Teretana	Osnovni	Ponedjeljak	Ramena	Predručenje šipkom	4	15	Uredi | Obriši
Teretana	Osnovni	Ponedjeljak	Bicepsi	Stojeći biceps pregib šipkom	3	20	Uredi | Obriši
Teretana	Osnovni	Ponedjeljak	Tricepsi	Stojeća triceps ekstenzija šipkom	4	12	Uredi | Obriši
Teretana	Osnovni	Srijeda	Trbuh	Trbušnjaci na klupi	3	15	Uredi | Obriši
Teretana	Osnovni	Srijeda	Leđa	Zgibovi	4	12	Uredi | Obriši
Teretana	Osnovni	Petak	Podlaktice	Pregib ručnog zgloba šipkom	4	15	Uredi | Obriši
Teretana	Osnovni	Petak	Natkoljenice	Noge na spravi	4	15	Uredi | Obriši
Teretana	Osnovni	Petak	Potkoljenice	Listovi na spravi	3	20	Uredi | Obriši
Teretana	Napredni	Utorak	Prsa	Potisak s ravne klupe	5	15	Uredi | Obriši
Teretana	Napredni	Utorak	Ramena	Predručenje šipkom	5	15	Uredi | Obriši
Teretana	Napredni	Utorak	Bicepsi	Stojeći biceps pregib šipkom	5	15	Uredi | Obriši
Teretana	Napredni	Utorak	Tricepsi	Stojeća triceps ekstenzija šipkom	5	15	Uredi | Obriši
Teretana	Napredni	Četvrtak	Trbuh	Trbušnjaci na klupi	5	15	Uredi | Obriši
Teretana	Napredni	Četvrtak	Leđa	Zgibovi	5	15	Uredi | Obriši
Teretana	Napredni	Četvrtak	Podlaktice	Pregib ručnog zgloba šipkom	5	15	Uredi | Obriši

Sl. 4.15. *Lista programa*

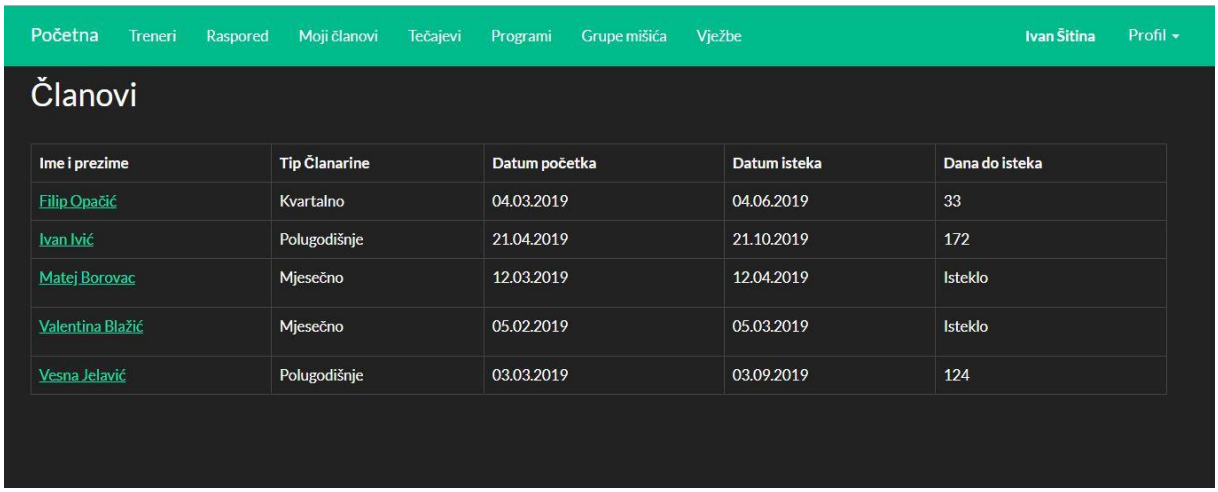
Na slici 4.15. prikazana je kontrolna tablica u kojoj trener ima mogućnost dodavanja i upravljanja vježbama koje je osmislio za svoje članove. Svaki trener u tablici vidi samo vježbe koje je on osmislio.



Sl. 4.16. *Forma za unos nove vježbe*

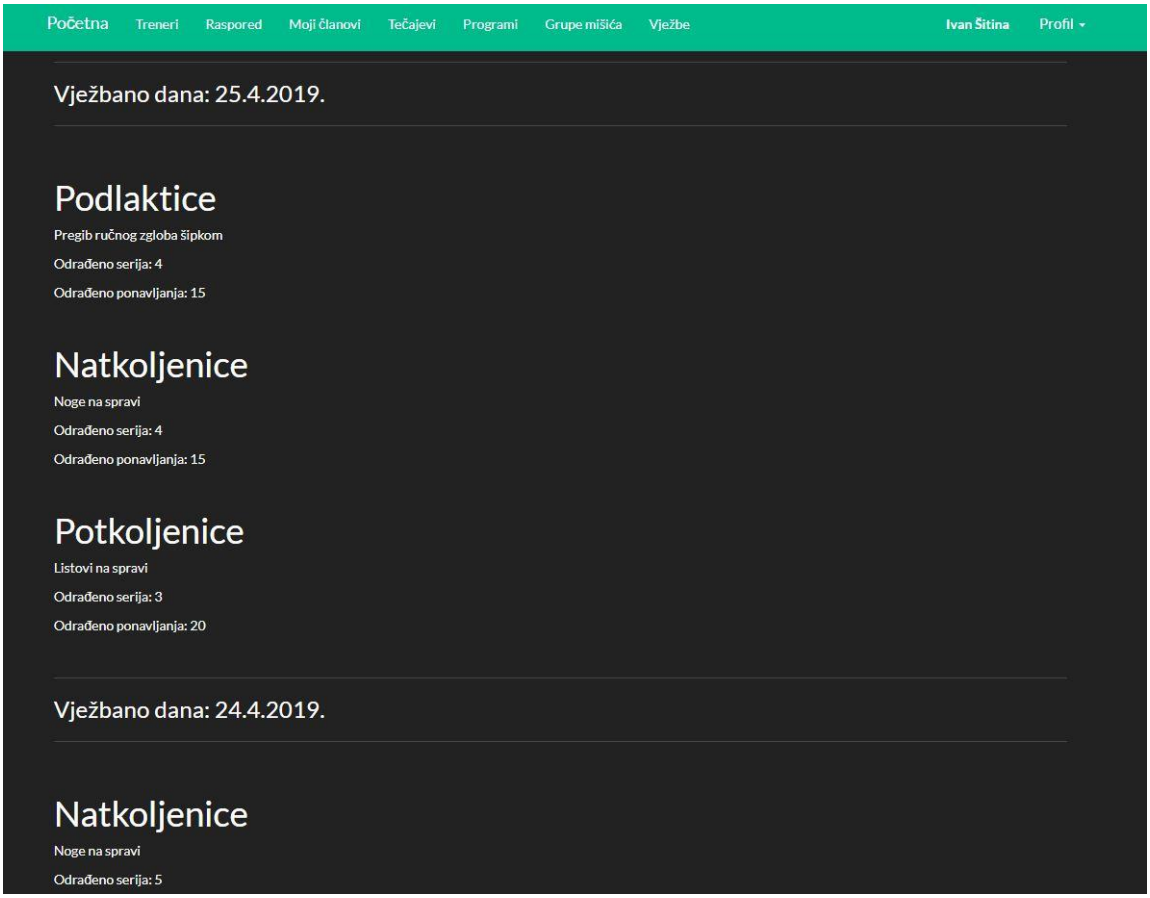
Na slici 4.16. prikazana je forma gdje trener dodaje određenu vježbu. U formu unosi naziv vježbe, broj predviđenih serija i ponavljanja, a mogućnost odabira iz padajuće liste ima u poljima „Grupa mišića“, „Program“, „Tečaj“ i „Dan izvođenja“ koja su prethodno definirana i kojima trener, također, može upravljati.

Trener ima pristup i listi svojih članova na način koji je predočen na slici 4.17.



Sl. 4.17. *Trenerovi članovi*

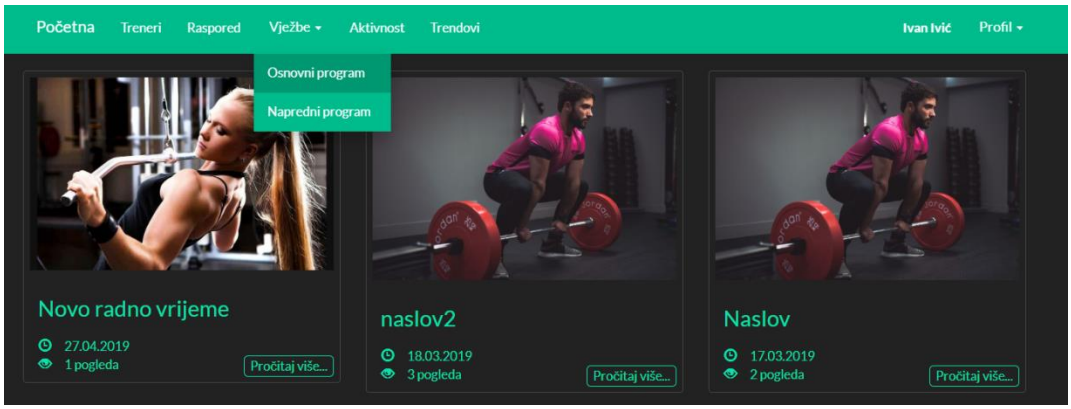
U tablici svojih članova trener vidi osnovne podatke o članarinama, a klikom na ime člana vidi uvid u sve odrađene vježbe sortirane po danima kako bi mogao pratiti napredak svojih članova. Nakon klika na ime člana, otvara se forma kao na slici 4.18.



Sl. 4.18. *Detalji o odrađenim vježbama*

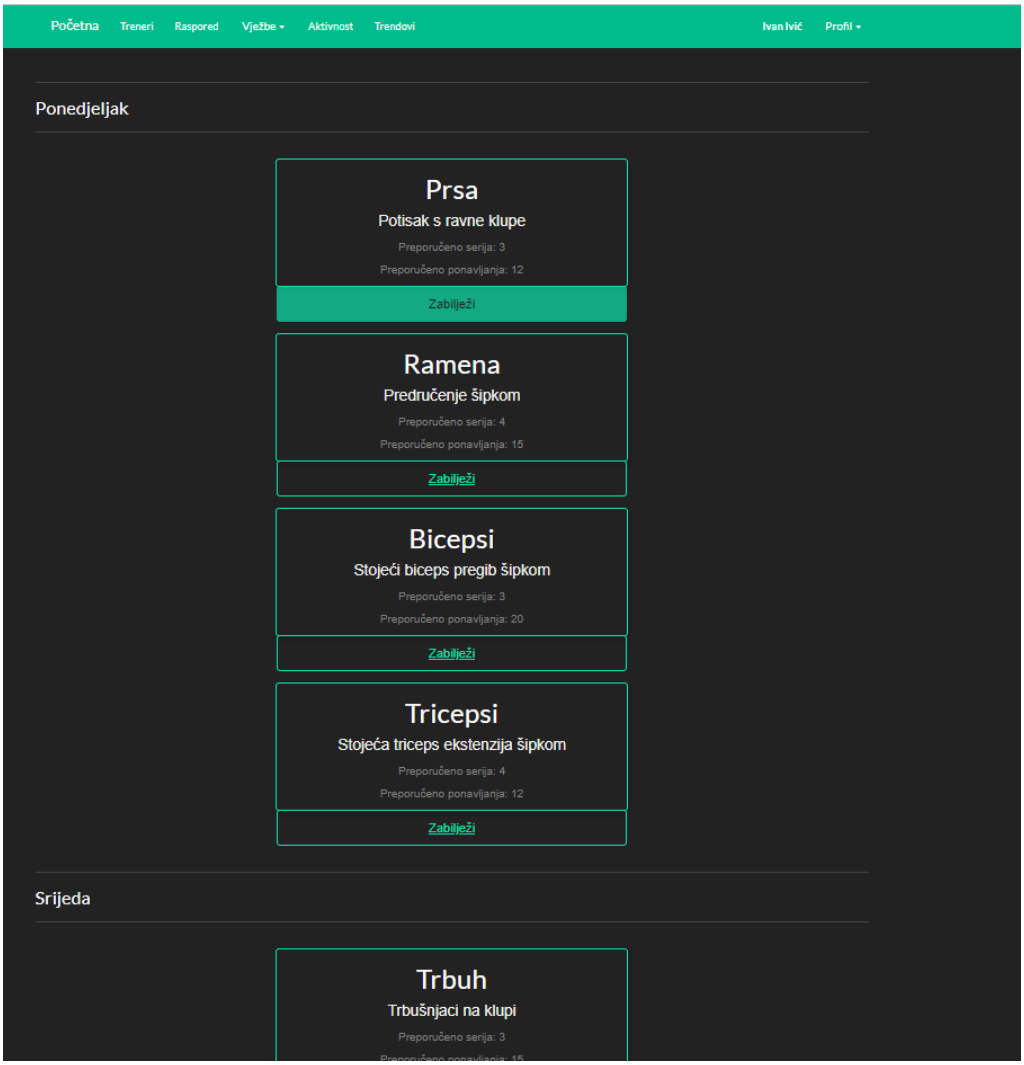
4.4. Rola korisnika

Nakon prijave u sustav kao korisnik, otvara se početna stranica, ali i dodatne funkcionalnosti koje su potrebne članovima za rad u sustavu. Treneri imaju skup vježbi koje pripadaju određenom programu. Programi mogu biti osnovni i napredni. Članovi mogu birati po kojem će programu vježbati kao što je prikazano na slici 4.19.



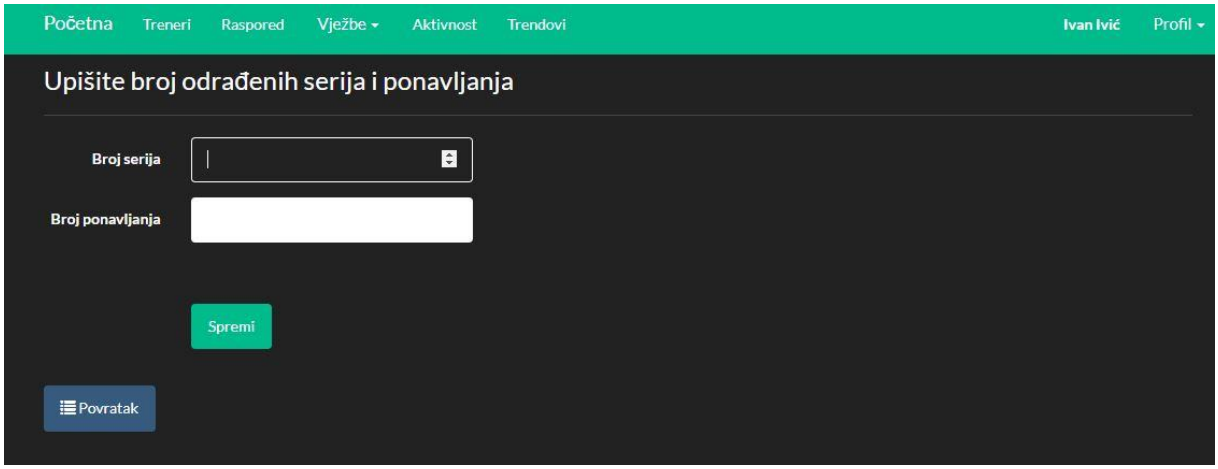
Sl. 4.19. *Izbor programa vježbanja*

Nakon što je član odabrao osnovni ili napredni program, otvara mu se forma na kojoj se nalaze sve vježbe koje je njegov trener predvidio za određeni dan u tjednu što je vidljivo na slici 4.20.



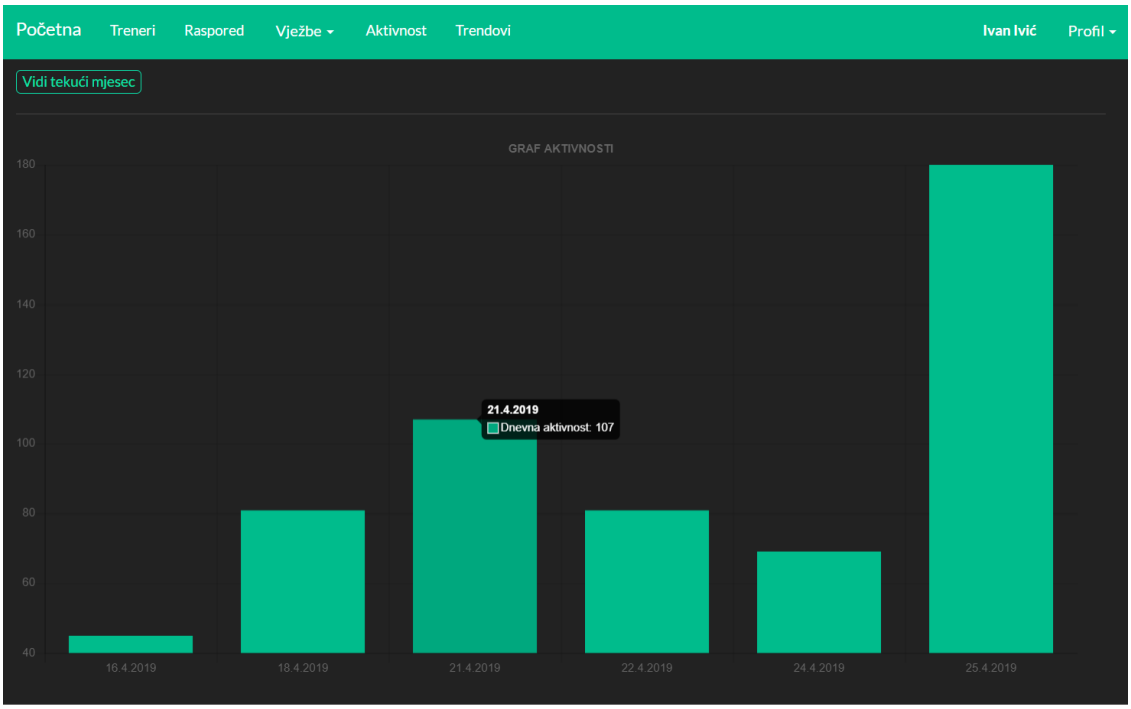
Sl. 4.20. *Program vježbanja*

Članovima je kod vježbi preporučen broj serija i ponavljanja koje trebaju napraviti. Nakon što završe s vježbom, klikom na gumb „Zabilježi“ imaju mogućnost unosa stvarnog broja serija i ponavljanja koje su odradili. Forma za unos je prikazana na slici 4.21.



Sl. 4.21. *Unos serija i ponavljanja*

Vođenjem evidencije, članovi stvaraju zapise o svom vježbanju i napretku. Osim što imaju uvid u odrađene vježbe po danu koje vide i njihovi treneri, članovi mogu vidjeti i svoj mjesečni graf opterećenja kao što je prikazano na slici 4.22. Broj koji navodi graf u stupcu, označava sumu umnožaka serija i ponavljanja u označenom danu. Na taj način i korisnik prima povratnu informaciju o napretku i opterećenju kroz mjesec. Osim toga, izlučeni podaci mogu i trenerima poslužiti za statističke podatke ili rangiranje korisnika s obzirom na napredak prema dobnim i spolnim kategorijama te slično.



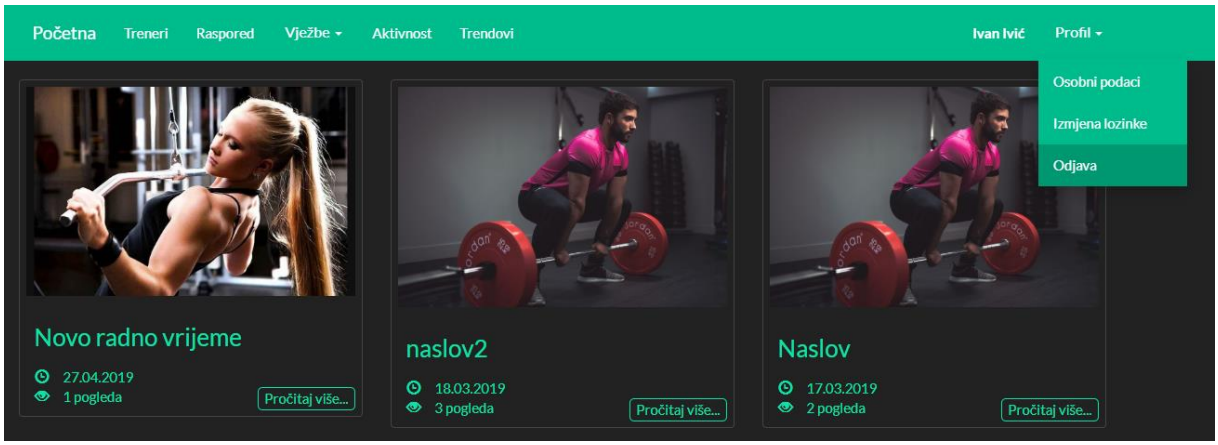
Sl. 4.22. *Graf mjesečne aktivnosti*

Osim uvida u graf trenutnog mjeseca, imaju uvid i u prethodni mjesec kako bi dobili što bolje predočenje o svom napretku.

4.5. Dodatno

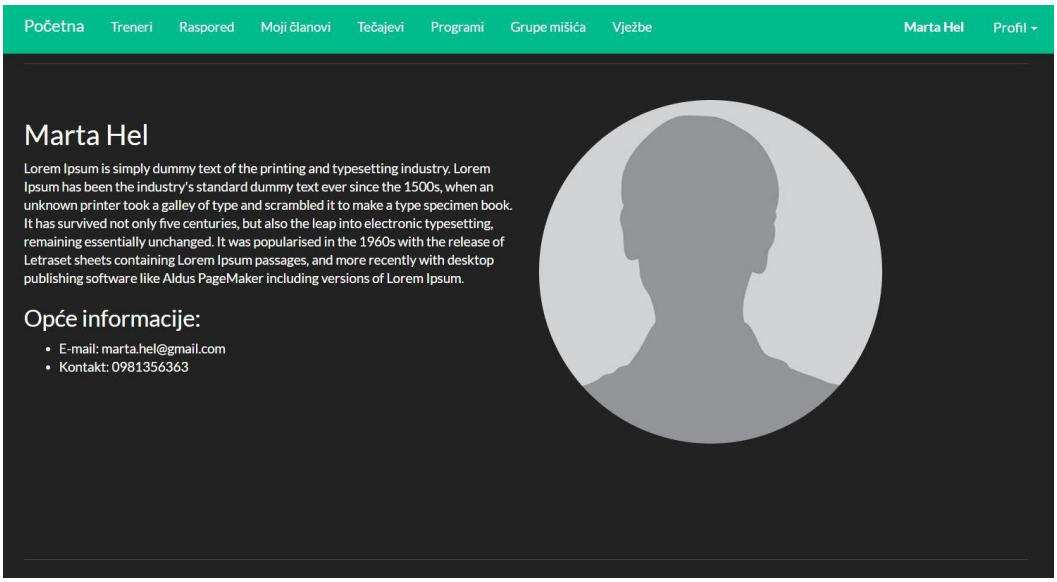
4.5.1. Profil

U gornjem desnom kutu sustava (Sl. 4.23) nalazi se ime osobe koja je prijavljena u web aplikaciju i padajuća lista „Profil“ u kojoj se nalaze osobni podaci prijavljenog korisnika, mogućnost izmjene lozinke i odjava iz sustava koja trenutnog korisnika vraća na formu za prijavu u sustav.



Sl. 4.23. *Odjava iz sustava*

Klikom na „Osobni podaci“ dolazi se na formu gdje su prikazani osobni podaci prijavljene osobe u obliku profila na način kako je prikazano na slici 4.24.

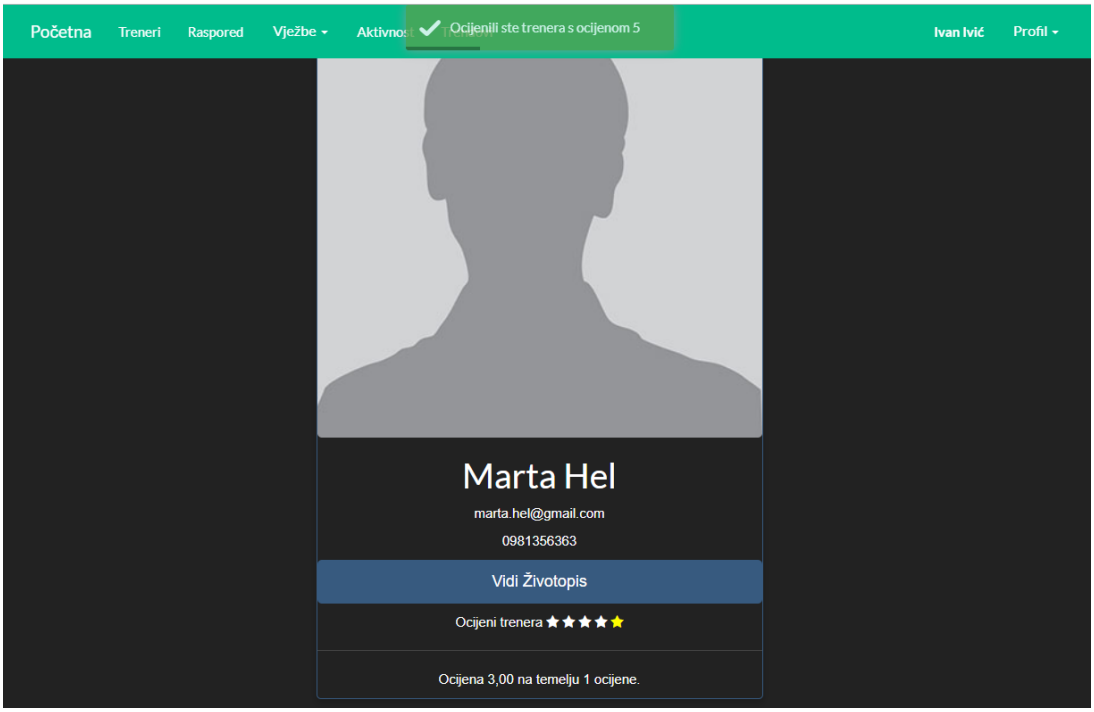


Sl. 4.24. *Moj profil*

S obzirom da polje za unos slike tijekom registracije nije obavezno, ako korisnik ne unese sliku sustav mu dodijeli predodređenu (eng. Default) sliku profila.

4.5.2. Ocjenjivanje trenera

Radi što lakšeg izbora trenera, članovi imaju uvid u njihov životopis kojemu mogu pristupiti preko gumba „Vidi životopis“ koji se nalazi na kartici trenera vidljivoj na slici 4.25.



Sl. 4.25. *Kartica trenera*

Osim što mogu vidjeti životopis, članovi imaju mogućnost ocijeniti trenera kako bi novim članovima, na osnovu te povratne informacije, bilo što lakše izabrati svojega. Samo prijavljeni korisnici mogu ocjenjivati, a svaki korisnik može trenera ocijeniti samo jednom. Na prethodnoj slici je vidljivo na temelju koliko glasova je formirana navedena srednja ocjena trenera.

5. ZAKLJUČAK

Cilj ovog rada bila je praktična izrada web aplikacije za upravljanje teretanom, koristeći neko razvojno okruženje. Za izradu web aplikacije korištene su tehnologije temeljene na Microsoftovim tehnologijama pa je tako za bazu podataka korišten Microsoft SQL Server, a za izradu aplikacije ASP.NET MVC i REST API s programskim jezikom C# u razvojnom okruženju Visual Studio. Klinjentsko sučelje prikazano je pomoću tehnologija Razor, HTML, CSS, JavaScript i JQuery.

S obzirom da su se prilikom razvijanja aplikacije koristile različite metode u implementaciji određenih modula, izradom ove aplikacije stečeno je praktično iskustvo te uvid u nedostatke i prednosti korištenja pojedinih metoda. Aplikacija nije napravljena kao tipično vođenje evidencije članova nekog sustava gdje ju koristi samo administrator. Izrađena je u tri različite role gdje korisnici, s obzirom na rolu, imaju određene funkcionalnosti koje mogu koristiti, a koje se međusobno i razlikuju. Izradom ove aplikacije nastojala su se pokazati teoretska znanja o programiranju web aplikacije, predočena u prvim dijelovima koja su zatim praktično pokazana u drugom dijelu rada, objašnjenjem izvedenog konkretnog primjera, odnosno izrade web aplikacije za upravljanje teretanom što se ilustriralo i kroz niz slikovnih priloga. Pokazana su stečena znanja u praktičnoj primjeni za pretpostavljene potrebe pri čemu se nastojalo što funkcionalnije pristupiti studiji slučaja. Međutim, za širu primjenu ove web aplikacije i cijelog sustava, preostaje još puno rada i nadogradnji, poput izrade prihvatljivijeg dizajna korisnicima, pogled u povijest svih članarina, mogućnost promjene slike profila, komentiranje objava od strane korisnika i slično.

LITERATURA

[1] https://techterms.com/definition/c_sharp [Pristup ostvaren 06.05.2019.]

[2] I. Lukić, M. Köhler, Osnove Internet programiranja, Elektrotehnički fakultet Osijek, Sveučilište J.J. Strossmayera u Osijeku, Osijek, 2011.

[3] https://www.w3schools.com/bootstrap/default.asp [Pristup ostvaren 06.05.2019.]

[4] http://purencool.com/javascript-and-jquery-what-is-the-difference [Pristup ostvaren 06.05.2019.]

[5] A. Vodanović, I. Pavić, MVC razvoj aplikacija : povezivanje s bazama podataka, Algebra, Zagreb, 2017.

[6] http://www.entityframeworktutorial.net/choosing-development-approach-with-entity-framework.aspx [Pristup ostvaren 06.05.2019.]

[7] T. Kaštelan, I. Mesić, Uvod u baze podataka, Algebra, Zagreb, 2009.

[8] S. Davila, T. Kaštelan, Osnove SQL-a, Algebra, Zagreb, 2010.

[9] I. Pavić, S. Davila, C# i .NET Framework 4.5, Algebra, Zagreb, 2017.

[10] https://stackoverflow.com/ [Pristup ostvaren 06.05.2019.]

[11] https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework [Pristup ostvaren 06.05.2019.]

[12] https://docs.microsoft.com/en-us/aspnet/web-api/ [Pristup ostvaren 06.05.2019.]

[13] https://docs.microsoft.com/en-us/aspnet/mvc/ [Pristup ostvaren 06.05.2019.]

[14] https://www.codecademy.com/articles/what-is-rest [Pristup ostvaren 06.05.2019.]

[15] https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10) [Pristup ostvaren 06.05.2019.]

[16] https://msdn.microsoft.com/en-us/library/5k850zwb.aspx [Pristup ostvaren 06.05.2019.]

[17] https://www.codeproject.com/ [Pristup ostvaren 06.05.2019.]

[18] http://csharp-video-tutorials.blogspot.com/p/aspnet-mvc-tutorial-for-beginners.html [Pristup ostvaren 06.05.2019.]

[19] https://cpratt.co/file-uploads-in-asp-net-mvc-with-view-models/ [Pristup ostvaren 06.05.2019.]

[20] https://technet.microsoft.com/en-us/library/bb496996.aspx [Pristup ostvaren 06.05.2019.]

SAŽETAK

U radu je obrađena problematika izrade web aplikacije, uz ilustraciju odabranog primjera web aplikacije za upravljanje teretanom. Aplikacija je podijeljena na tri glavna sloja, a to su baza podataka, servisni sloj i klijentski sloj. Web aplikacija ima mogućnost registriranja korisnika i to u tri role - administratorsku, rolu trenera i običnog korisnika.

Ključne riječi: ASP.NET, C#, CSS, HTML, Javascript, JQuery, MVC, SQL, Web API

ABSTRACT

Web application for gym management

The paper deals with web design issues. The application is divided into three main layers and those are database, service layer and client layer. The web application own the ability to register users in three different roles, such as administrator role, trainer role and ordinary user role.

Keywords: ASP.NET, C#, CSS, HTML, Javascript, JQuery, MVC, SQL, Web API

ŽIVOTOPIS

Ivan Šitina rođen je 27. prosinca 1990. godine u Požegi. Nakon završene Osnovne škole *Ivan Goran Kovačić* u Velikoj, 2005. godine upisao je Tehničku školu u Požegi, smjer računalstvo, koju je završio 2009. godine. Godine 2011. upisao je Stručni studij Elektrotehnike u Osijeku, smjer informatika. Obranivši završni rad *Svojstva realnih fluida* pod mentorstvom dr. sc. Željke Mioković – profesor visoke škole, 2014. godine stekao je titulu inženjer elektrotehnike, smjer informatika. U razdoblju od 2014. do 2015. godine položio je razlikovne obveze nakon čega upisuje diplomski studij.

Trenutno je student 2. godine diplomskog studija računarstvo, smjer informacijske i podatkovne znanosti na Fakultetu elektrotehnike računarstva i informacijskih tehnologija u Osijeku. Od stranih jezika koristi se engleskim, a rad na računalu poznaje odlično.

Prije početka diplomskog studija, odradio je akademiju u tvrtki *Span d.o.o*. stekavši znanja i vještine o izradi web aplikacija. Krajem akademske godine, još kao redovan student (u rujnu 2017. godine) zaposlio se u tvrtki *Zvijezda plus d.o.o.* gdje i danas radi kao suradnik za BI i ERP sustave.

_____

Potpis: Ivan Šitina