Univerzitet u Beogradu

Elektrotehnički fakultet



DIPLOMSKI RAD

VEB APLIKACIJA ZA IZVRŠAVANJE UPITA NAD BAZOM PODATAKA

|  |  |
| --- | --- |
| Mentor: | Kandidat: |
| doc. dr Miloš Cvetanović | Stefan Đorđević  0211/2013 |

Beograd, 2017.

Sadržaj

[1. Uvod 2](#_Toc494741630)

[2. Razvoj ASP.net MVC veb aplikacija 3](#_Toc494741631)

[2.1. ASP.net MVC 3](#_Toc494741632)

[2.2. „Front-end“ 3](#_Toc494741633)

[2.2.1. HTML i CSS 3](#_Toc494741634)

[2.2.2. CSHTML i Razor 4](#_Toc494741635)

[2.2.3. JavaScript i jQuery 4](#_Toc494741636)

[2.2.4. Bootstrap 5](#_Toc494741637)

[2.3. "Back-end" 6](#_Toc494741638)

[2.3.1. C# 6](#_Toc494741639)

[2.3.2. Entity framework 7](#_Toc494741640)

[2.3.3. LINQ 8](#_Toc494741641)

[3. Korisnički aspekt aplikacije 9](#_Toc494741642)

[3.1.1. Stranica za prijavu 9](#_Toc494741643)

[3.1.2. Stranica za izbor baze 11](#_Toc494741644)

[3.1.3. Stranica za izvršavanje upita 11](#_Toc494741645)

[3.1.4. Stranica „Studenti“ 13](#_Toc494741646)

[3.1.5. Stranica „Baze“ 14](#_Toc494741647)

[3.1.6. Stranica „Dodaj“ 14](#_Toc494741648)

[4. Implementacioni aspekt aplikacije 16](#_Toc494741649)

[4.1.1. Organizacija baze podataka 16](#_Toc494741650)

[4.1.2. Prijava korisnika 17](#_Toc494741651)

[4.1.3. Izbor baze 18](#_Toc494741652)

[4.1.4. Odjavljivanje 20](#_Toc494741653)

[4.1.5. Izvršavanje upita 22](#_Toc494741654)

[4.1.6. Dodavanje studenta i baze 24](#_Toc494741655)

[5. Zaključak 26](#_Toc494741656)

[6. Literatura 27](#_Toc494741657)

1. Uvod

Računari su od svog nastanka pa do danas predstavljali vrhunac tehnološkog dostignuća čoveka. Danas računare imamo u raznim oblicima i prihvatamo ih kao deo svakodnevnog života. Međutim, na samom početku njihovog razvoja računari su bili potpuno nedostupni prosečnom čoveku.

Iako lični računari postoje već skoro 40 godina, tek u poslednje dve decenije oni postaju nezamenljivi u svakodnevnom životu i radu. Jedan od razloga sigurno leži u intenzivnom razvoju interneta. Osim što je omogućio komunikaciju na globalnom nivou, internet je u velikoj meri uticao na način na koji se današnji lični računari koriste. Prevashodno, razvoj interneta doneo je potpuno novu vrstu aplikacija – veb aplikacije. Jedna od osnovnih prednosti ovih aplikacija jeste njihova pristupačnost. Za njihovo korišćenje dovoljna je internet konekcija i računar skromnih performansi.

I pored pomenute dostupnosti jedna sfera društva u kojoj su veb aplikacije teško pronašle mesto jeste obrazovanje. Tek u poslednjih par godina možemo videti dramatičan porast u broju onlajn kurseva. Međutim, većina tih kurseva jednostavno prezentuju sadržaj, najčešće u video formatu. Iako je to znatni napredak u odnosu na tradicionalne metode učenja, retki su projekti koji su u potpunosti iskoristili pun potencijal veb aplikacija u ovoj oblasti.

Taj potencijal leži u interaktivnosti koju one pružaju. Odatle potiče i motivacija za ovaj rad. Veb aplikacija „Vežbaj MSSQL“ zamišljena je kao dopunsko nastavno sredstvo u izučavanju baza podataka. Ova interaktivna veb aplikacija njenim korisnicima treba da omogući da se konektuju na neku od ponuđenih baza podataka i nad njom izvršavaju upite. Sagledavajući rezultate upita korisnici bi lako mogli da provere i utvrde svoja znanja. Ovakva aplikacija bi eliminisala neke od koraka koji najčešće demotivišu studente da stečena znanja testiraju u praksi kao što su podizanje servera i kreiranja samih baza. Iako su i ovo jako bitni elementi izučavanja baza podataka činjenica je da će neko ko se prvi put susreće sa njima teško uspeti da ih samostalno izvede. Pored olakšavanja savladavanja materije studentu, ovakvi projekti bi mogli doneti i novčanu uštedu fakultetima. Umesto izdvajanja sredstava za kupovinu licenci za sve studente dovoljno bi bilo nabaviti jednu licencu za samu veb aplikaciju.

U nastavku najpre će biti reči o tehnologijama koje su korišćene za njegovu realizaciju, zatim ćemo se osvrnuti na korisnički aspekt same aplikacije koji se može shvatiti kao neka vrsta korisničkog uputstva i demonstracije onoga što aplikacija nudi. Konačno, poslednja glava tiče se probranih delova iz aspekta implementacije aplikacije.

1. Razvoj ASP.net MVC veb aplikacija

Svrha ovog poglavlja jeste kratak pregled tehnologija korišćenih u izradi veb aplikacije koja je tema ovog rada. O samim tehnologijama postoji mnoštvo literature i potpuno razumevanje ovih tehnologija nije cilj ovog rada. Ono što jeste cilj je da čitalac bude svestan koja je svrha svake od tehnologija odnosno šta je to što ona donosi. U poglavlju 4, u kom se govori o detaljima implementacije, biće nagoveštena upotreba svake od ovih tehnologija. Pre upoznavanja sa pojedinačnim tehnologijama izložene su neke osnovne informacije o ASP.net MVC framework-u.

* 1. ASP.net MVC

ASP.net predstavlja okvir za razvoj softvera (eng. framework) namenjen razvoju veb sajtova, veb aplikacija i veb servisa. Dizajniran je od strane Majkrosofta kao besplatan i „open-source“ projekat. Predstavlja naslednika ASP (eng. Active Server Pages) tehnologije. Prvi put je predstavljen januara 2002. godine zajedno sa verzijom 1.0 .Net framework-a i izgrađen je na CLR-u (Common Language Runtime), komponenti .Net framework-a koja predstavlja samu virtuelnu mašinu.

ASP.net MVC predstavlja deo ASP.net okvira koji koristi MVC (eng. Model View Controller) šablon softverske arhitekture. Model predstavlja centralnu komponentu šablona i izražava ponašanje aplikacije nezavisno od korisničkog interfejsa. Direktno radi sa podacima i implementira logiku i pravila aplikacije. Kontroler prihvata ulaz i konvertuje ga u komande za Model ili View. Poslednja komponenta šablona je View – predstavlja oblik reprezentacije informacije korisniku. Moguće je da postoje više različitih View – ova za istu informaciju.

ASP.net je kompatibilan sa velikim brojem tehnologija i drugih framework-a što je jedan od razloga zbog kojih danas ima značajan udeo na internetu.

* 1. „Front-end“
     1. HTML i CSS

**HTML** (eng. HyperText Markup Language) je jezik za označavanje takozvanog hiperteksta. Predstavlja opisni jezik specijalno namenjen za opis veb stranica. Pomoću njega se jednostavno mogu odvojiti elementi kao što su naslov, paragrafi, multimedije, korisničke kontrole i slično. Pored toga, u HTML standard ugrađeni su elementi koji detaljnije opisuju sam dokument kao što su ključne reči, podaci o autoru i slično. Ovi podaci su opštepoznati kao metapodaci i jasno su odvojeni od sadržaja dokumenta. HTML je nastao uprošćavanjem SGML-a (eng. Standard Generalized Markup Language) sa svrhom opisa dokumenata koji se objavljuju na vebu. U početku je bio prilično ograničen, ali kako je veb rastao tako ja rasla i potreba za bogatijim sadržajem pa je u tom smeru razvijan i sam HTML standard.

XHTML (eng. Extensible HyperText Markup Language) je deo XML (eng. Extensible Markup Language) familije jezika. U nekim segmentima predstavlja pandan, a u nekim drugim proširenje u odnosu na sam HTML. Dok je HTML (do verzije 5) bio definisan kao implementacija SGML standarda, XHTML predstavlja implementaciju XML-a, restriktivnijeg podskupa SGML-a. XHTML dokumenti su jako formatirani i kao takvi mogu se, za razliku od HTML-a, parsirati standardnim XML parserima.

Međutim, iako preporučen od strane W3C (World Wide Web Consortium), prelazak na HTML nije bio moguć odjednom. HTML je imao veliki udeo na internetu, a dolaskom HTML verzije 5 otklonjene su mnoge mane zbog kojih je XHTML stekao prednost.

**CSS** (eng. Cascading Style Sheets) je jezik koji služi za opisivanje načina prezentacije dokumenta napisanog u nekom „Markup“ jeziku. Iako se najčešće koristi da definiše vizuelni stil veb stranica i korisničkih interfejsa, CSS može da se primeni na bilo koji XML dokument uključujući standardni XML ili SVG. Osnovna svrha CSS-a je da omogući razdvajanje stila prezentovanja od samog sadržaja koji se prezentuje. Ovo razdvajanje povećava jednostavnost pojedinačnih dokumenata, omogućava veću fleksibilnost i kontrolu pri specifikaciji načina prezentacije i omogućava da više stranica sa sadržajem (poput HTML stranica) dele istu CSS datoteku. Korišćenjem CSS-a znatno se povećava jedna od najpoželjnijih karakteristika programskog koda – njegova centralizovanost, čime kod postaje čitljiviji i jednostavniji za održavanje. Specifikacija CSS-a je takođe pod okriljem W3C-a.

* + 1. CSHTML i Razor

Iako je ASP.net MVC u potpunosti kompatibilan sa standardnim HTML-om, u projektu koji je tema ovog rada nisu korišćeni čisti HTML fajlovi. ASP.net ima poseban format fajlova koji predstavlja proširenje HTML-a, takozvani CSHTML. Skraćenica CSHTML potiče od programskog jezika C# (eng. C Sharp) i HTML. U ovakvim datotekama moguće je pored standardnog HTML koda koristiti i C# kod. Upotreba C# u HTML-u omogućava se i izvršavanje znatno kompleksnijih zadataka kao što su pristup bazi. Iako je to moguće, ne predstavlja dobru praksu jer načelno narušava MVC arhitekturu spajajući View i Model. Sa druge strane je korišćenje elemenata C# jezika kao što su kontrolne strukture grananja i petlji može značajno da umanji količinu HTML koda bez narušavanja MVC principa.

Očigledna prepreka u ostvarivanju ovakve tehnologije jeste činjenica da veb pretraživači ne mogu da izvršavaju C# kod. Međutim pokazuje se da za izvršavanjem C# koda na klijentskoj strani uopšte nema potrebe. Naime, C# kod CSHTML stranica uopšte ne dolazi do veb pretraživača. Taj kod se izvršava na serveru, pre slanja odgovora pretraživaču. Na ovaj način omogućeno je dinamičko kreiranje veb stranica.

Upotreba C# koda u CSHTML stranicama mora da poštuje Razor sintaksu. Razor je jezik za označavanje (eng. markup language) koji se koristi u CSHTML stranicama. Pored C#, Razor podržava i korišćenje jezika Visual Basic.

* + 1. JavaScript i jQuery

JavaScript je slabo tipiziran programski jezik visokog nivoa. Spada u interpretativne jezike što znači da većina implementacija ovog jezika instrukcije izvršavaju direktno bez prethodnog kompajliranja u mašinske instrukcije. Ovo se ostvaruje tako što se svaka naredba pretvara u jednu ili više rutina koje su unapred prevedene u mašinski kod. Takođe, JavaScript je objektno baziran jezik. Kao i objektno orijentisani jezici, objektno bazirani jezici koriste ideju enkapsulacije stanja i operacija unutar objekata. Razlika u odnosu na objektno orijentisane jezike ogleda se u tome što objektno bazirani jezici ne podržavaju nasleđivanje.

Zajedno sa HTML-om i CSS-om, JavaScript predstavlja jedan od tri osnovne veb tehnologije. Osnovna namena JavaScript-a je da donese interaktivnost veb stranicama. Većina veb sajtova i aplikacija koriste JavaScript i svi moderni veb pretraživači pružaju podršku za ovaj jezik. Podrška se zasniva na posedovanjem interpretera za izvršavanje JavaScript koda. Postoji veliki broj ovakvih interpretera i svaki reprezentuje drugačiju implementaciju samog jezika. Sve implementacije zasnovane su ECMAScript specifikaciji s tim da neki interpreteri ne podržavaju kompletnu specifikaciju dok drugi podržavaju i dodatne komponente koje nisu deo same specifikacije.

JavaScript je jezik koji podržava više od jedne paradigme programiranja. Neke od paradigmi koje JavaScript podržava su objektno orijentisano programiranje, funkcionalno programiranje i programiranje vođeno događajima koje je od posebnog značaja za ovaj rad. Takođe, jezik poseduje interfejse za rad sa tekstom, nizovima, datumima, regularnim izrazima i bazičnom manipulacijom DOM (eng. Document Object Model) objektima. Sa druge strane jezik ne podržava bilo kakve ulazno-izlazne operacije poput rada sa mrežom, skladištenje ili rada sa grafičkim okruženjem. U tom pogledu se u potpunosti oslanja na sistem u koji je ugrađen.

Inicijalno JavaScript je bio zamišljen kao programski jezik za veb koji bi se izvršavao isključivo u veb pretraživačima odnosno na klijentskoj strani. U ovom radu, JavaScript je korišćen isključivo u tu svrhu. Međutim, treba napomenuti da JavaScript danas ima znatno širu upotrebu. Može se naći kao deo veb servera, u bazama podataka čak i u programima koji nisu striktno vezani za veb kao što su procesori reči. Iako postoji velika sličnost sa programskim jezikom Java koja se između ostalog odnosi na sintaksu i deo biblioteka, u pogledu dizajna ovo su dva veoma različita jezika.

**jQuery**  je međuplatformska JavaScript biblioteka. Dizajnirana je da pojednostavi kreiranje skripti za klijentsku stranu. Biblioteka je besplatna, „open-source“ i koristi permisivnu MIT licencu. Analize pokazuju da je jQuery ubedljivo najrasprostranjenija JavaScript biblioteka. Sama sintaksa dizajnirana je tako da olakša navigaciju dokumentom, odabiranje DOM elemenata, kreiranje animacija, obradu događaja i razvoj AJAX aplikacija. Upravo poslednja pomenuta namena je ona zbog koje se u projektu koji je tema ovog rada jQuery najviše koristi.

AJAX (eng. Asynchronus Javascript And Xml) pretstavlja upotrebu XML Http zahteva za komunikaciju sa serverom. AJAX omogućava prijem i slanje informacija u različitim formatima kao što su JSON (JavaScript Object Notation), XML, HTML i tekstualne datoteke. Najprivlačnija karakteristika AJAX-a je njegova asinhrona priroda. Ovo omogućava da se korišćenjem AJAX tehnologije uspostavi komunikacija sa serverom, izvrši razmena podataka i osveži deo sadržaja stranice bez potrebe za ponovnim učitavanjem cele stranice. Ovim razdvajanjem sloja razmene podataka od sloja prezentacije istih otvorile su se velike mogućnosti u domenu razvoja veb sajtova i veb aplikacija. Treba napomenuti da AJAX nije zasebna tehnologija već predstavlja specifičan skup tehnologija sa jedinstvenom namenom.

* + 1. Bootstrap

Bootstrap predstavlja besplatan, „open-source“ radni okvir za „front-end“. Koristi se isključivo za dizajniranje stranica veb sajtova i veb aplikacija. Boostrap poseduje veliku kolekciju kreiranih komponenata za korisnički interfejs. Ovo omogućava ubrzani razvoj veb sajtova i veb aplikacija. Sam Bootrstap nije zasebna tehnologija. On predstavlja kolekciju HTML-a, CSS-a i JavaScript-a. Bootstrap se u velikoj meni oslanja i na jQuery. Bootsrap kontrole karakteriše jednostavna inkorporaciju u projekat, veliki raznovrsnost dizajna i dinamičnost samih komponenti. Budući da je „open-source“ ostavlja veliki prostor za modifikaciju samih komponenti tako da su tom pogledu mogućnosti neograničene. Ono što Bootsrap izdvaja od drugih kolekcija „front-end“ komponenti jeste činjenica da je veliki akcenat stavljen na mobilnost.

Iako tačnih podataka nema, Bootsrap svakako važi za jedan od najrasprostranjenijih kolekcija komponenti danas. Deo velike zajednice korisnika konstantno doprinosi daljem razvoju popularnosti Bootstrap-a. Jedan takav vid doprinosa jesu teme dizajnirane za Bootstrap. Postoji kolekcija zvaničnih tema koje su razvijene od strane Bootstrap razvojnog tima kao i teme koje su kreirali nezavisni programeri. Jedan vid motivacije entuzijasta za razvoj ovakvih tema jeste mogućnost naplate korišćenja tih tema. U izradi projekta koji je tema ovog rada korišćena je jedna besplatna tema pod nazivom „sb-admin“. Upotreba ove teme je omogućila ostvarivanje relativno dobrog vizuelnog dizajna uz minimalan napor i prebacila fokus na implementaciju poslovne logike.

Postoji više načina upotrebe Bootstrap-a među kojima se izdvajaju uključivanje izvornog koda i korišćenje CDN-a (Content Delivery Network). Prednost CDN-a je u jednostavnosti korišćenja – dovoljno je referencirati odgovarajuće datoteke na internetu i one će se automatski učitavati. Sa druge strane funkcionisanje same aplikacije zavisi od funkcionisanja CDN-a što nije dobro. Ono što takođe nije dobro je dodatni mrežni saobraćaj koji nastaje usled komunikacije sa CDN-om. U ovom radu Bootstrap je korišćen uključivanjem izvornog koda u sam projekat.

* 1. "Back-end"
     1. C#

**Programski jezik C#** može se okarakterisati kao elegantan jezik prilično jednostavne sintakse sa velikim mogućnostima. Spada u objektno orijentisane jezike. Sam jezik danas ima prilično široku upotrebu. Može se koristiti za programiranje klijentskih aplikacija na Windows operativnom sistemu, programiranje XML veb servisa, distribuiranih aplikacija, klijent – server aplikacija i mnogo više. U ovom radu C# nalazi primenu kao jezik na kom je napisan serverski deo same veb aplikacije.

Sintaksa jezika je dosta ekspresivna, a opet jednostavna kako za korišćenje tako i za učenje. Programeri koji su prethodno koristili jezike kao što su C, C++ ili Java jako brzo se prilagođavaju ovom jeziku. Razlog leži u tome što C# u velikoj meri pojednostavljuje neke od kompleksnih elemenata pomenutih jezika dok sa druge strane uvodi novine koje dodatno olakšavaju razvoj aplikacija. Neke od takvih novina jesu nulabilni tipovi, delegati, lambda izrazi i mogućnost direktnog pristupa memoriji.

Budući da spada u objektno orijentisane jezike C# podržava koncepte enkapsulacije, nasleđivanja i polimorfizma. Sve promenljive i metode, uključujući i metodu Main koja predstavlja ulaznu tačku aplikacije su enkapsulirane unutar definicija klasa. Klasa može direktno da nasledi samo jednu roditeljsku klasu, ali može da implementira veći broj interfejsa. Metode koje nadjačavaju metodu roditeljske klase zahtevaju upotrebu ključne reči override kako bi se izbegla slučajna redefinicija metode.

Pored ovih osnovnih objektno orijentisanih principa, C# olakšava proces razvoja softverskih komponenti kroz nekoliko inovativnih jezičkih konstrukata uključujući:

* Enkapsulirani potpisi metoda (takozvani delegati) koji omogućavaju tipovski bezbedna obaveštenja o događajima.
* Svojstva koji služe kao pristupnice privatnim delovima klasa
* LINQ koji omogućava ugrađenu mogućnost kreiranja jedinstvenih upita nezavisnih od tipa izvora podataka. Više o ovome u odeljku 2.3.3.

Treba napomenuti i da je proces kompilacije znatno jednostavniji u poređenju sa jezicima C i C++ i fleksibilniji u odnosu na jezik Java. Nema zasebnih datoteka koje predstavljaju zaglavlja, a i ne zahteva se redosled deklaracija metoda i tipova. Izvorna C# datoteka može deklarisati više od jedne klase, strukture, interfejsa ili događaja.

* + 1. Entity framework

Entity radni okvir predstavlja „open-source“ ORM (eng. Object-Relational Mapping) tehnologiju. ORM tehnologije predstavljaju tehniku za pretvaranje podataka između nekompatibilnih sistema tipova korišćenjem objektno orijentisanog programiranja. Ovo efektivno stvara virtuelnu bazu objekata koja može da se koristi direktno iz samog objektno orijentisanog programskog jezika.

  
*Slika 2.1 Odnos Aplikacija-ORM-Baza*

Entity može biti koristan u jednoj od tri situacije:

1. Ukoliko već imamo postojeću bazu podataka koju želimo da koristimo u svojoj aplikaciji ili u situaciji kada želimo da prvo dizajniramo bazu podataka. Ovo je takozvani „Database first“ pristup.
2. Suprotno od prethodnog pristupa – ako želimo da ne zalazimo u dizajn same baze podataka već da dizajniramo aplikaciju zajedno sa modelima koji predstavljaju podatke koji će se u aplikaciji koristiti. U tom slučaju Entity može samostalno, na osnovu modela da u potpunosti generiše bazu podataka. Ovaj pristup se naziva „Code first“
3. Treća situacija je takođe neka vrsta „Database first“ pristupa, ali se umesto baze dizajnira njen model u neko vizuelno dizajneru. Entity radni okvir na osnovu tog dizajna generiše sa jedne strane bazu podataka, a sa druge strane modele koji će se tu bazu predstavljati u samoj aplikaciji.

Navedene situacije ilustrovane su na slici 2.2.

  
*Slika 2.2 Slučajevi upotrebe ORM-a*

Inicijalno Entity radni okvir je bio deo .Net radnog okvira, ali od njegove šeste verzije postaje nezavistan od .Net radnog okvira.

* + 1. LINQ

LINQ je Majkrosoftova .Net tehnologija prvi put objavljena 19. novembra 2007. godine kad sastavni deo .Net verzije 3.5. LINQ nije tehnologija nije vezana za konkretan jezik. Može se koristiti u bilo kom .Net jeziku, a najčešća upotreba je u jezicima C#, F#, i VB.NET. Cilj LINQ-a je da integriše mogućnost upita u programske jezike na jedinstven način, nezavistan od vrste upita.

Upit je suštinski izraz čijim izvršavanjem se pristupa podacima iz nekog izvora. Upiti se obično izražavaju preko nekog od jezika za upite. Vremenom došlo je do razvoja više jezika kojima se upiti mogu izvršavati. Tako na primer imamo SQL za relacione baze podataka ili Xquery za XML. Problem koji ovde nastaje je da programeri usled ove raznovrsnosti moraju da nauče zaseban upitni jezik za svaki tip izvora podataka sa kojim rade. LINQ pojednostavljuje situaciju uvodeći konzistentan model rada sa podacima za različite tipove podataka i izvora podataka. Sve LINQ operacije sastoje se iz tri karakteristične akcije: dohvatanje izvora podataka, kreiranje upita i izvršavanja upita. Osim što je nezavistan od samih jezika za upite LINQ je lako razumljiv čak i nekome ko nije sa njim upoznat.

Sve u svemu LINQ je vrlo moćna tehnologija koja u velikoj meri doprinosi jednostavnosti razvoja i održavanja svih vrsta aplikacija.

1. Korisnički aspekt aplikacije

U ovom poglavlju sagledaćemo kompletnu aplikaciju iz korisničkog ugla. Većina veb aplikacija ima potrebu za klasifikacijom njenih korisnika. Jedna od mogućnosti da se ovo izvede jeste pomoću korisničkih naloga. Takav pristup primenjen je i u ovom radu. Korisnici su klasifikovani u dve kategorije: studenti i administratori. Tačnije, postoji jedan administrator i više studenata. U zavisnosti od toga da li je korisnik student ili administrator, imaće pristup odgovarajućim stranicama same aplikacije. Svi korisnici, po otvaranju aplikacije, pristupaju stranici za prijavu. Pored stranice za prijavu administrator ima pravo pristupa stranicama „Studenti“, „Baze“ i „Dodaj“. Sa druge strane, student, pored stranice za prijavu ima pravo pristupa stranici za izbor baze i stranici za izvršavanje upita.

* + 1. Stranica za prijavu

Stranica za prijavu je prva stranica koja se pojavljuje po otvaranju aplikacije, nezavisno od toga da li je aplikaciju otvorio administrator ili student. Stranica je prilično jednostavna i sastoji se od naslova, podnaslova i forme za prijavu. Formu za prijavu čine dva tekstualna polja i dugme za prijavu. U prvo tekstualno polje unosi se korisničko ime, a u drugo lozinka. Zajedno oni čine kredencijale korisnika za pristup aplikaciji. Na osnovu kredencijala aplikacija utvrđuje identitet korisnika.

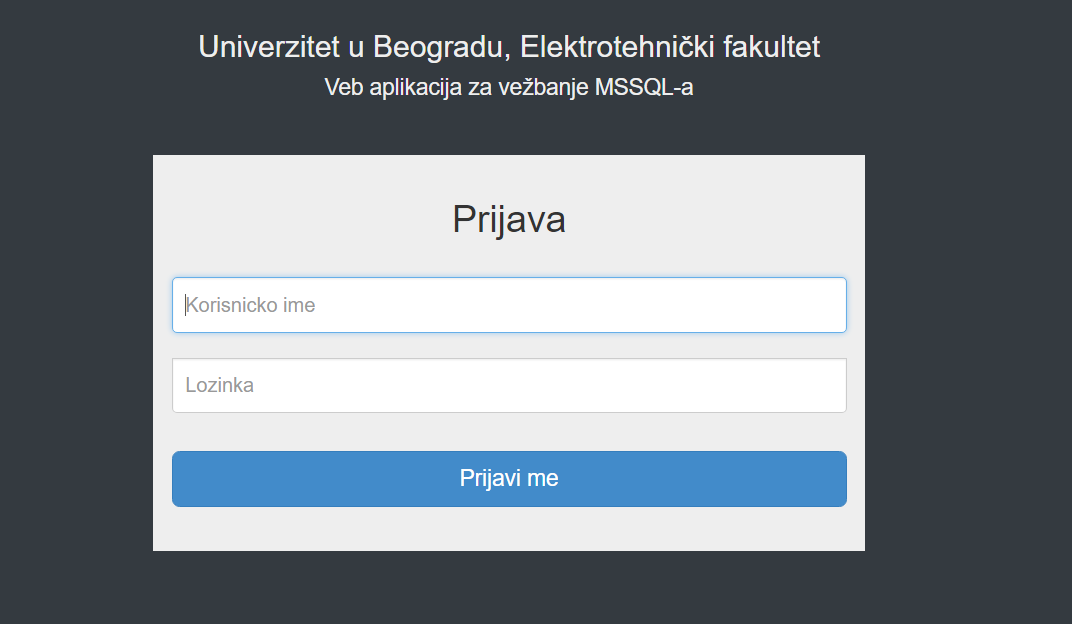
Pritiskom na dugme „Prijave me“ kredencijali se šalju ka serveru. Server na osnovu kredencijala u bazi podataka traži korisnika. Za to vreme korisniku je prikazan ekran za čekanje. Ukoliko u bazi podataka ne postoji korisnik sa zadatim kredencijalima, ispod dugmeta za prijavu pojavljuje se poruka „Pogrešno korisničko ime ili lozinka“. U suprotnom identitet korisnika je utvrđen, a samim tim i da li se radi o studentu ili o administratoru.

Ukoliko se radi o studentu, preusmerava se na stranicu za izbor baze. U suprotnom, administrator se preusmerava na stranicu „Studenti“.

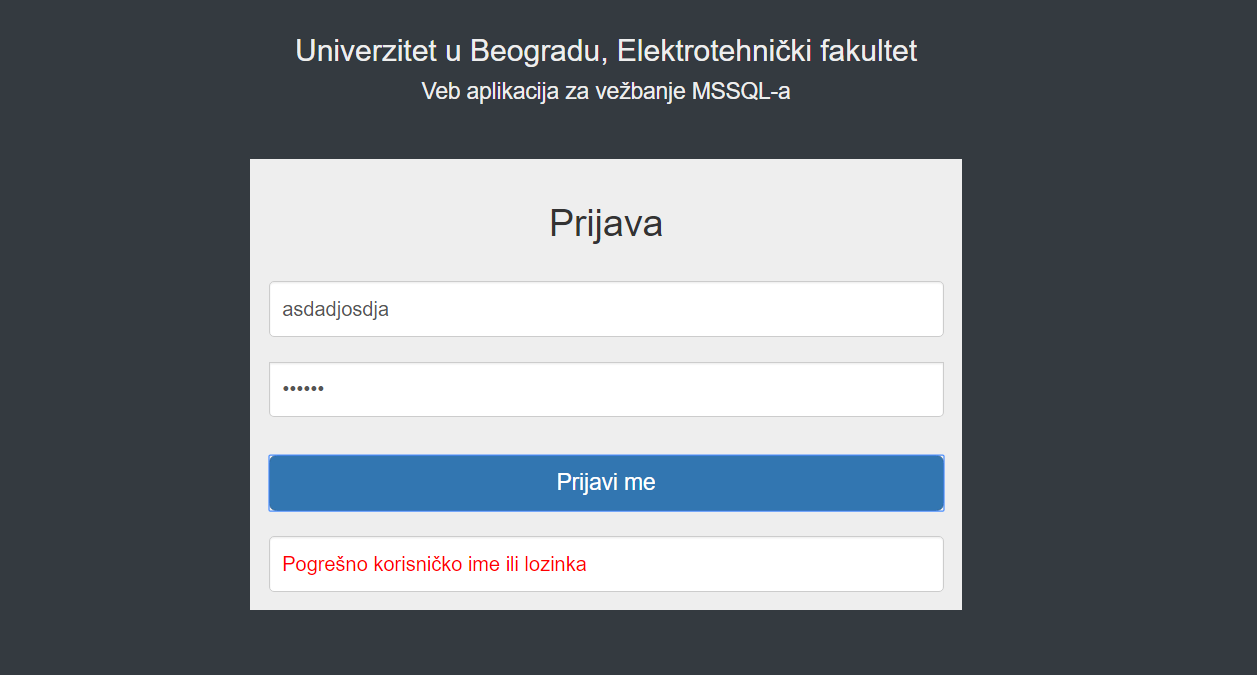
Na svim stranicama osim ove, u gornjem desnom uglu nalazi se ime i prezime prijavljenog korisnika kao i dugme „Odjavi se“. Ime i prezime su informativnog karaktera. Pritiskom na dugme „Odjavi se“ otvara se prozor dijaloga. Svrha ovog prozora je potvrda da je odjavljivanje zaista željena akcija korisnika. Ukoliko to nije slučaj, prozor se može zatvoriti pritiskom na „x“ ili na dugme „Otkaži“. Dugme „Odjavi se“ potvrđuje nameru korisnika i nakon pritiska na ovo dugme korisnik se preusmerava na stranicu za prijavu.

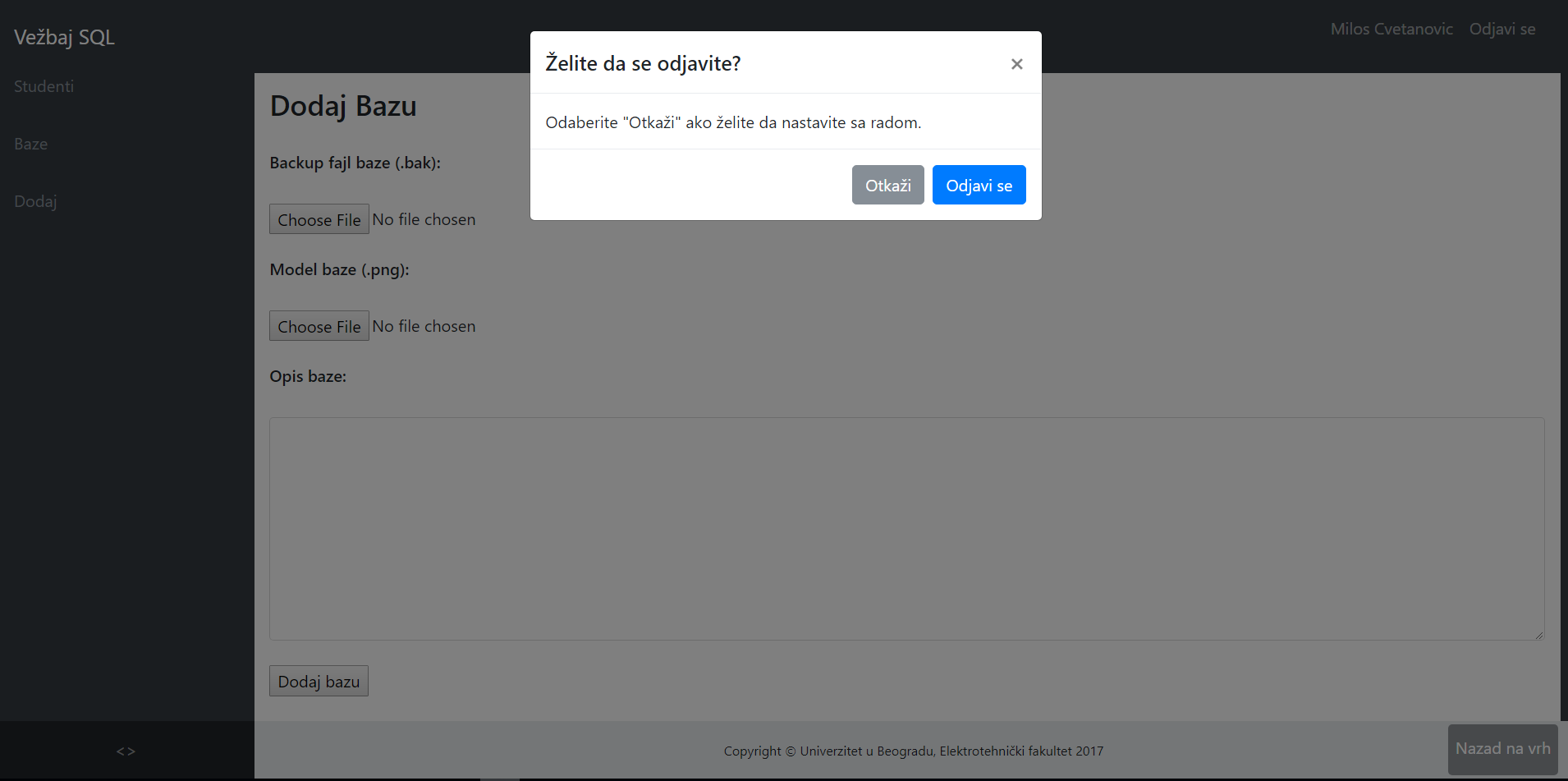
Takođe na svim stranicama osim ove postoji i dugme „Nazad na vrh“. Ovo dugme je vidljivo samo kada kompletan sadržaj stranice ne može da se prikaže na ekranu, a korisnik ne posmatra sam početak njenog sadržaja. Ovo dugme predstavlja prečicu korisniku za brz povratak na vrh strane.

Poslednji element koji je karakterističan za sve stranice osim ove jeste podnožje stranice sa natpisom o autorskim pravima.



*Slika 3.1 Stranica za prijavljivanje*



*Slika 3.2 Primer neuspešnog prijavljivanja*

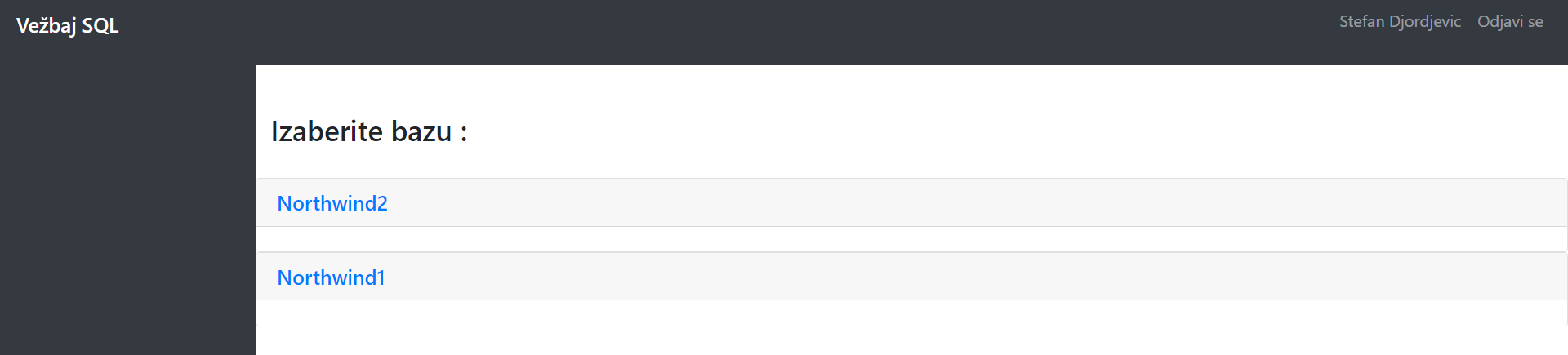
*Slika 3.3 Dijalog pri odjavljivanju*

* + 1. Stranica za izbor baze

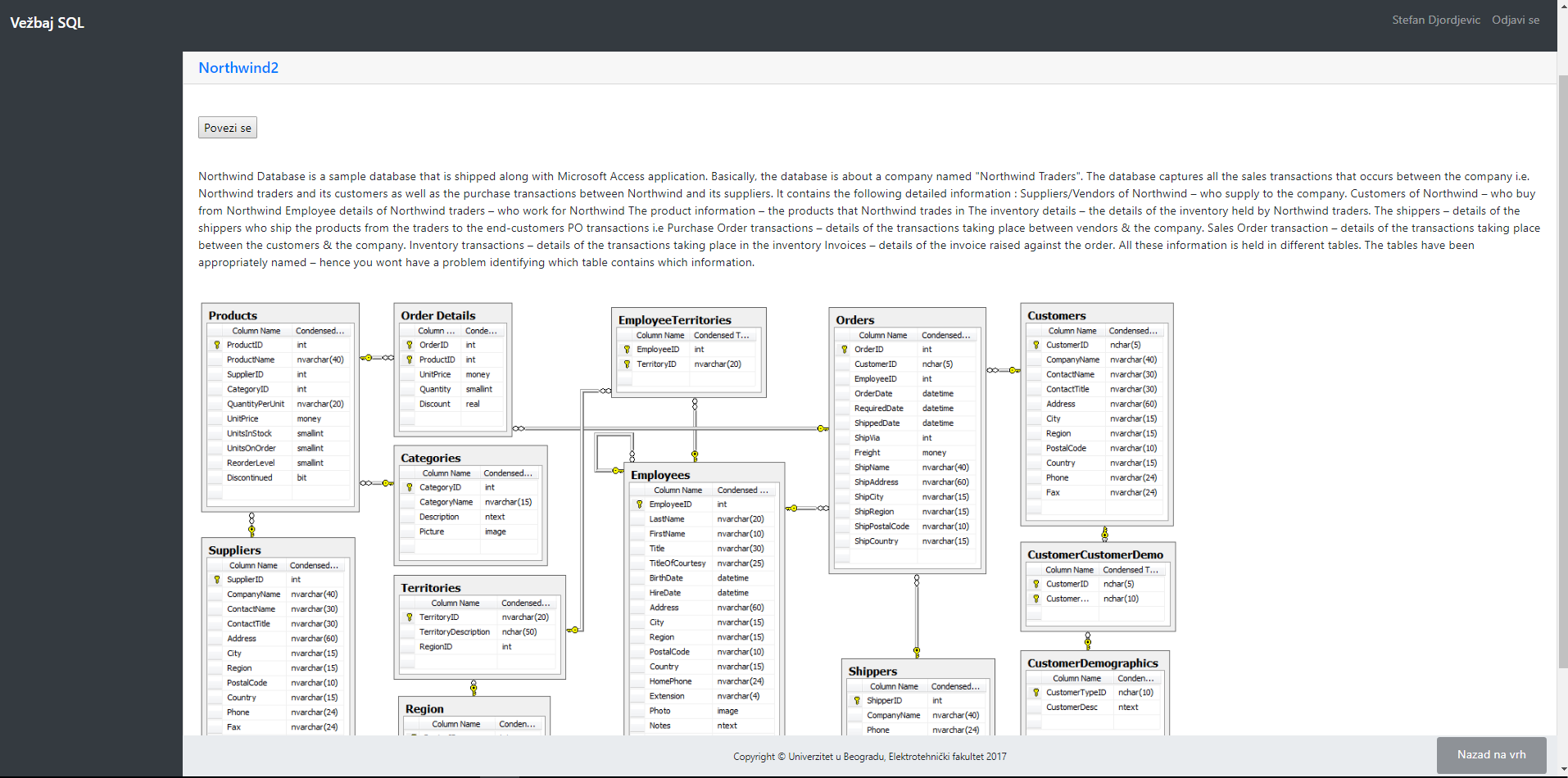
Na ovoj stranici student dobija listing svih baza nad kojima može da izvršava upite. Klikom na neku od baza otvara se padajući meni koji sadrži tekstualni opis i UML model te baze. Takođe, u padajućem meniju se nalazi i dugme “Poveži se”.

Ponovnim klikom na neku bazu, njen meni se zatvara. Na ovaj način studentu je omogućen pregled i upoznavanje sa svakom od ponuđenih baza.

Nakon što odluči nad kojom bazom želi da izvršava upite, korisnik se povezuje na bazu pritiskom na dugme „Poveži se“ koje pripada toj bazi. Ova akcija preusmerava korisnika na stranicu za izvršavanje upita.



*Slika 3.4 Lista izbora baze*



*Slika 3.5 Pregled sadržaja baze pre povezivanja*

* + 1. Stranica za izvršavanje upita

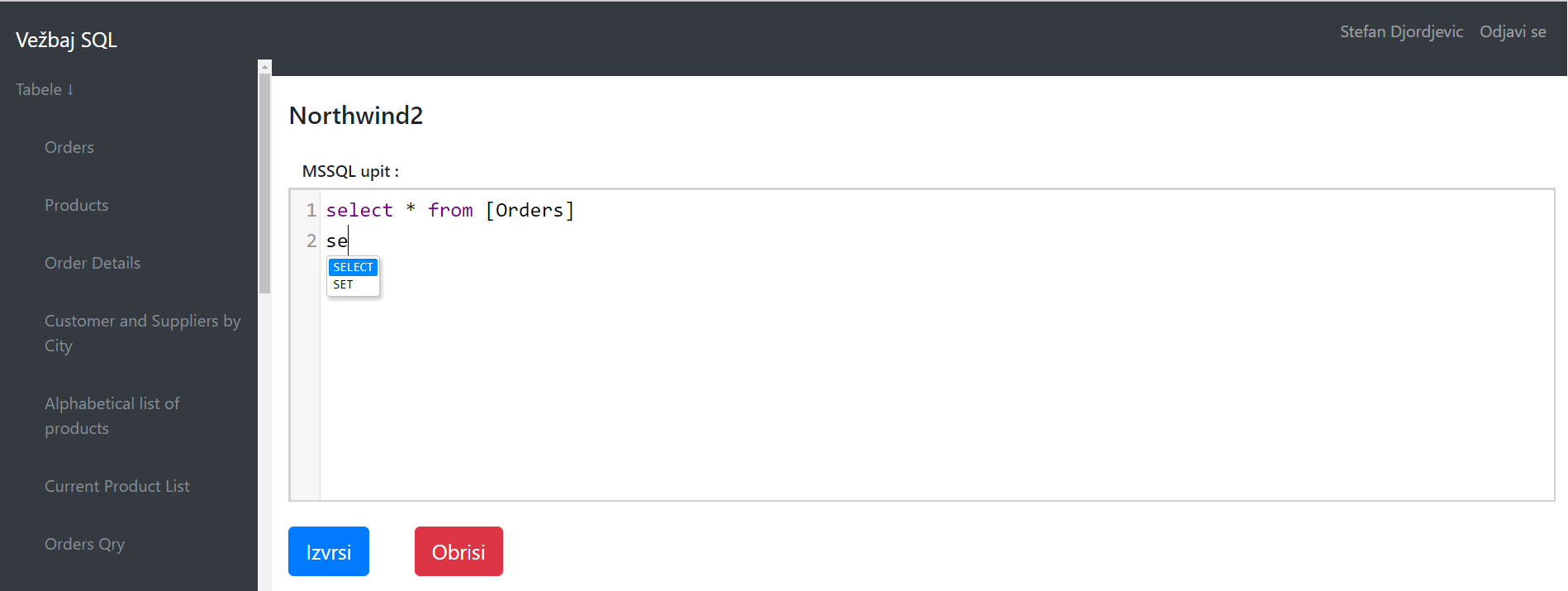
Stranica za izvršavanje upita predstavlja srž aplikacije. Sa lave strane nalazi se navigacioni meni koji se sastoji od dve stavke: „Nazad“ i „Tabele“. Taster „Nazad“ omogućava povratak na prethodnu stranicu (stranica za izbor baze). Pritiskom na taster „Tabele“, unutar navigacionog menija otvara se padajuća lista. Elementi padajuće liste predstavljaju imena tabela koja postoje u bazi podataka na koju se korisnik povezao. U dnu navigacionog menija nalazi se taster označen sa „<>“ koji omogućava sklapanje tj. proširenje navigacionog menija.

Pored navigacionog menija na stranici se nalazi i njen sadržaj. Sadržaj je sačinjen iz dve celine: korisničkog interfejsa za izvršavanje upita nad bazom i dela koji prikazuje rezultat izvršavanja upita.

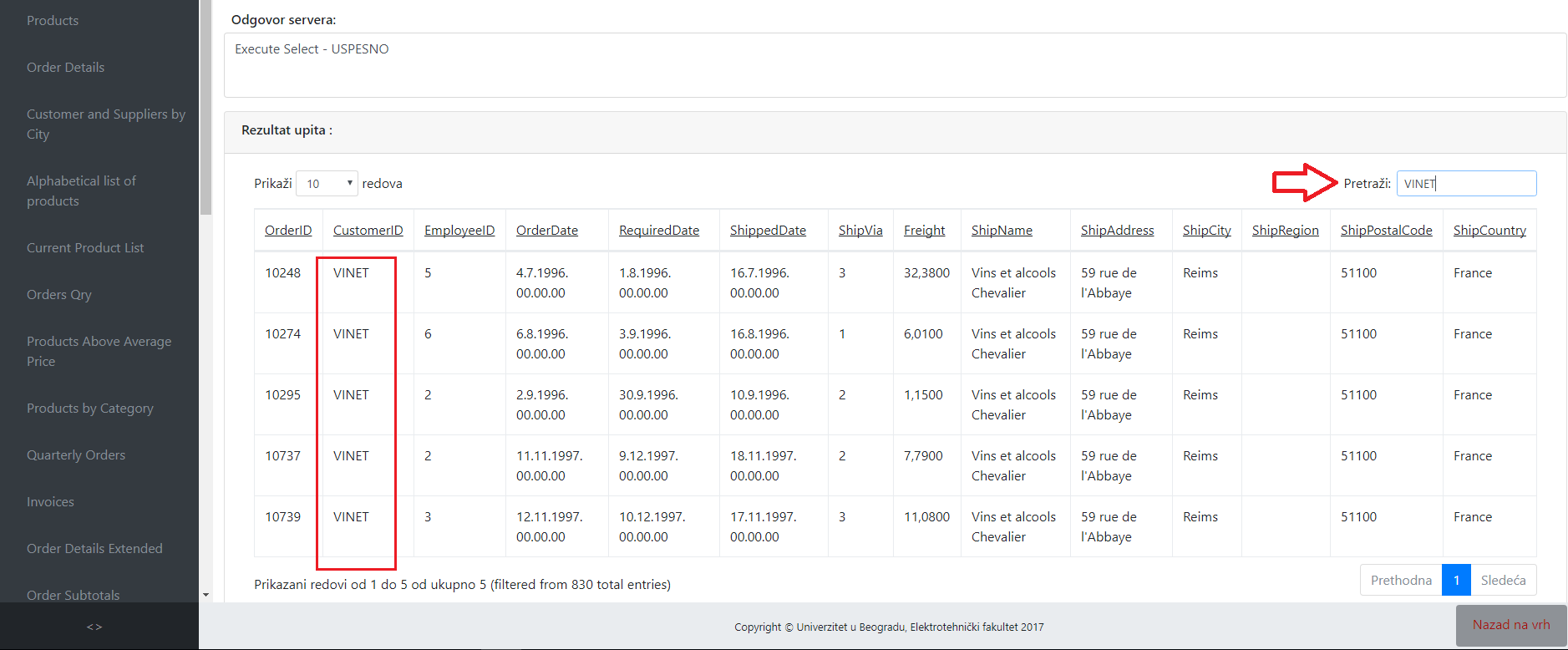
Korisnički interfejs čini tekstualno polje za unos samog upita i dugme „Izvrši upit“. Za razliku od ostalih tekstualnih polja u aplikaciji, ovo tekstualno polje ima nekoliko mogućnosti koje ubrzavaju i olakšavaju pisanje samih upita. Najpre, tu je prepoznavanje ključnih reči MSSQL-a i njihovo isticanje različitim bojama. Pored ovoga, tu je i podrška za automatsko dovršavanje reči koje korisnik unosi (ukoliko je to MSSQL ključna reč). I na kraju, ukoliko automatsko dovršavanje reči ne može da jednoznačno odredi o kojoj ključnoj reči se radi prikazuje se lista mogućih reči. Isticanje ključnih reči se aktivira samostalno kada se unese cela ključna reč dok se automatsko dovršavanje, kao i u većini modernih editora koda, aktivira kombinacijom tastera Ctrl i Space. Nakon što unese željeni upit, korisnik započinje njegovo izvršavanje pritiskom na „Izvrši“. Kao i kod prijave, za vreme obrade zahteva od strane servera, korisniku se prikazuje ekran za čekanje. Za brisanje sadržaja ovog tekstualnog polja, kao prečica, uvedeno je dugme „Obriši“.

Deo interfejsa koji prikazuje rezultat izvršavanja upita sastoji se od tabele i tekstualnog polja. U tekstualnom polju se prikazuje odgovor servera. Odgovor servera počinje porukom o uspešnosti izvršavanja. Ako je izvršavanje bilo neuspešno, ostatak odgovora predstavlja pouku o grešci do koje je došlo. Sa druge strane, ako je izvršavanje bilo uspešno, ostatak odgovora postoji samo ukoliko je upit promenio podatke u bazi i predstavlja broj izmenjenih redova. Ukoliko se radi o upitu koji menja bazu, tabela nema nikakvo značenje. Ukoliko se pak radi o „SELECT“ upitu, tabela predstavlja rezultat tog upita.

Upit koji se često koristi jeste „SELECT \* FROM [*Ime Tabele*]“ kojim se prikazuje kompletan sadržaj tabele čije je ime navedeno u upitu. Obzirom na njegovu čestu upotrebu, aplikacija poseduje funkcionalnost koja omogućava njegovo automatsko unošenje u tekstualno polje za upit. Dovoljno je da korisnik u padajućem meniju „Tabele“ odabere željenu tabelu i pomenuta funkcionalnost će se aktivirati.



*Slika 3.6 Zadavanje upita i automatsko dovršavanje ključnih reči*



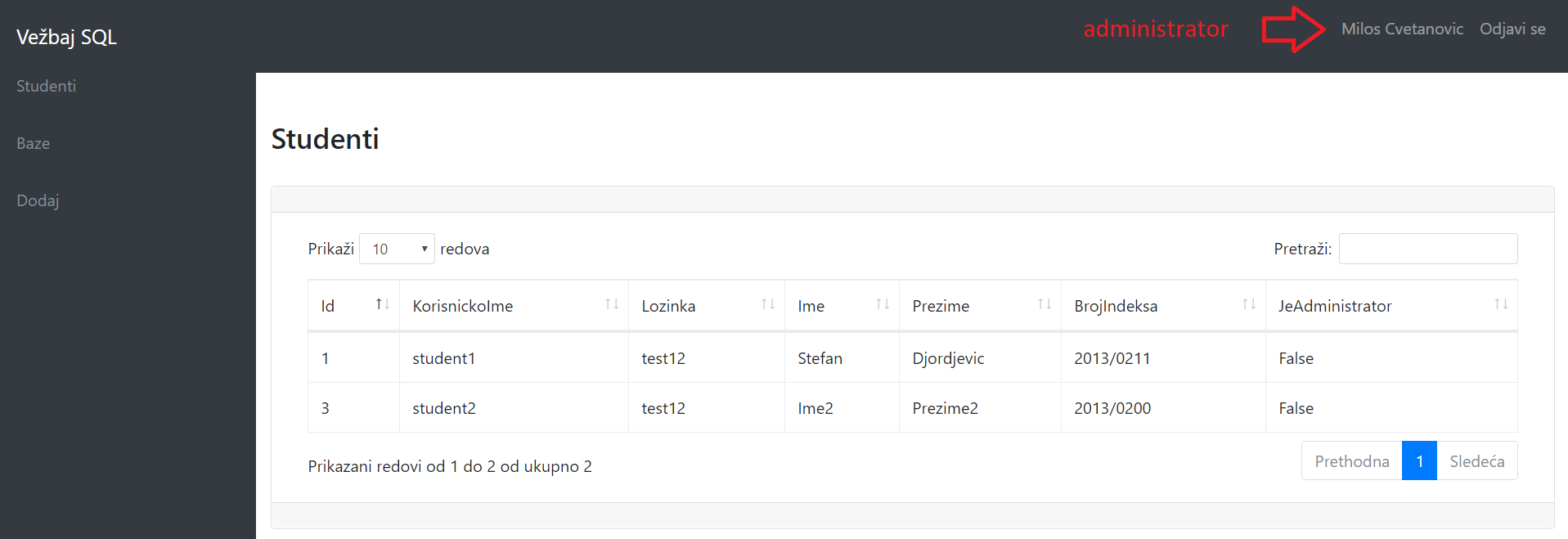
*Slika 3.7 Prikaz rezultata upita sa filtriranjem*

* + 1. Stranica „Studenti“

Stranica „Studenti“ je stranica na koju se administrator inicijalno preusmerava nakon uspešne prijave. U svom sadržaju, stranica ima tabelu u kojoj prikazuje informacije o svim studentima. Informacije su prikazane u kolonama: „Id“, „Korisnicko Ime“, „Lozinka“, „Ime“, „Prezime“ i „JeAdministrator“. Kako tabela prikazuje samo studente, vrednost poslednje kolone će uvek biti „False“. Tabela poseduje mogućnost prikaza 10, 25, 50 ili 100 ulaza, pretragu ulaza i sortiranje ulaza po bilo kojoj koloni. Takođe, tabela ima onoliko sopstvenih stranica koliko je neophodno da za izabrani broj ulaza koji se prikazuju, prikaže sve ulaze.

Sa leve strane nalazi se navigacija pomoću koje administrator može pristupiti stranama „Baze“ i „Dodaj“ ili osvežiti trenutnu stranicu. U dnu navigacionog menija nalazi se taster označen sa „<>“ koji omogućava sklapanje tj. proširenje navigacionog menija.

Studenti prikazani na ovoj stranici imaju mogućnost da koriste studentski segment aplikacije.

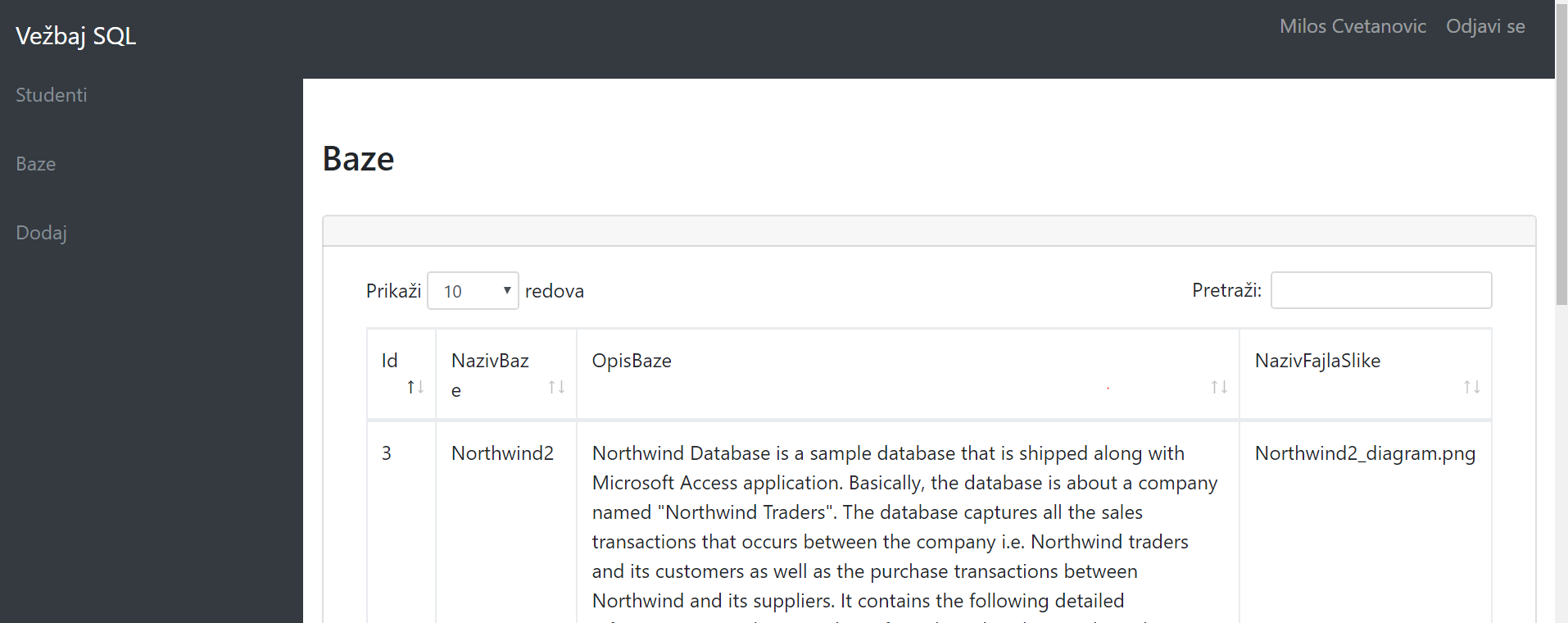


*Slika 3.8 Prikaz studenata administratoru*

* + 1. Stranica „Baze“

U svom sadržaju, stranica ima tabelu u kojoj prikazuje informacije o svim bazama na koje student može da se poveže. Dakle, ovo je isti onaj skup baza koji je studentu prikazan na stranici za izbor baze. Informacije su na ovoj stranici prikazane u kolonama: „Id“, „Naziv baze“, „Opis Baze i „Naziv fajla slike“. Tabela poseduje mogućnost prikaza 10, 25, 50 ili 100 ulaza, pretragu ulaza i sortiranje ulaza po bilo kojoj koloni. Takođe, tabela ima onoliko sopstvenih stranica koliko je neophodno da za izabrani broj ulaza koji se prikazuju, prikaže sve ulaze.

Sa leve strane nalazi se navigacija pomoću koje administrator može pristupiti stranama „Studenti“ i „Dodaj“ ili osvežiti trenutnu stranicu. U dnu navigacionog menija nalazi se taster označen sa „<>“ koji omogućava sklapanje tj. proširenje navigacionog menija.



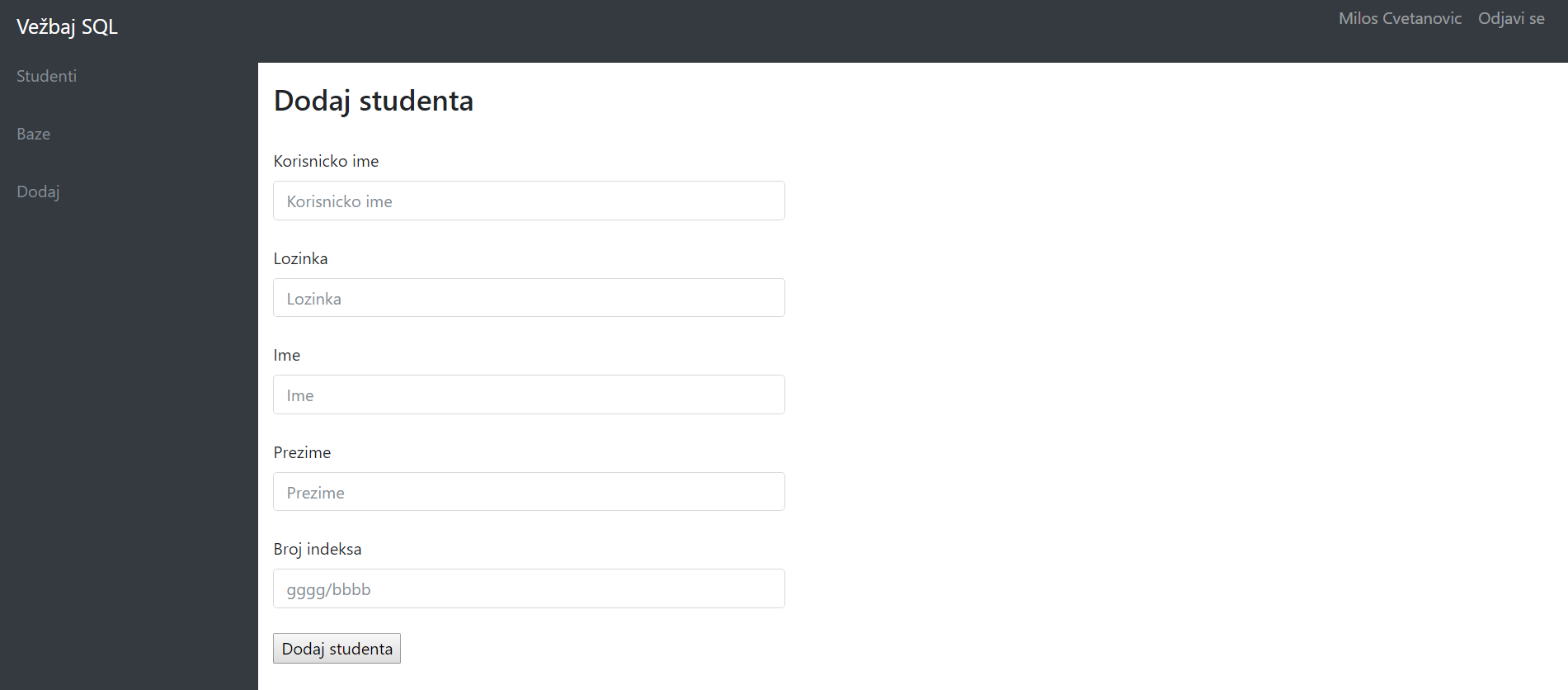
*Slika 3.9 Prikaz baza administratoru*

* + 1. Stranica „Dodaj“

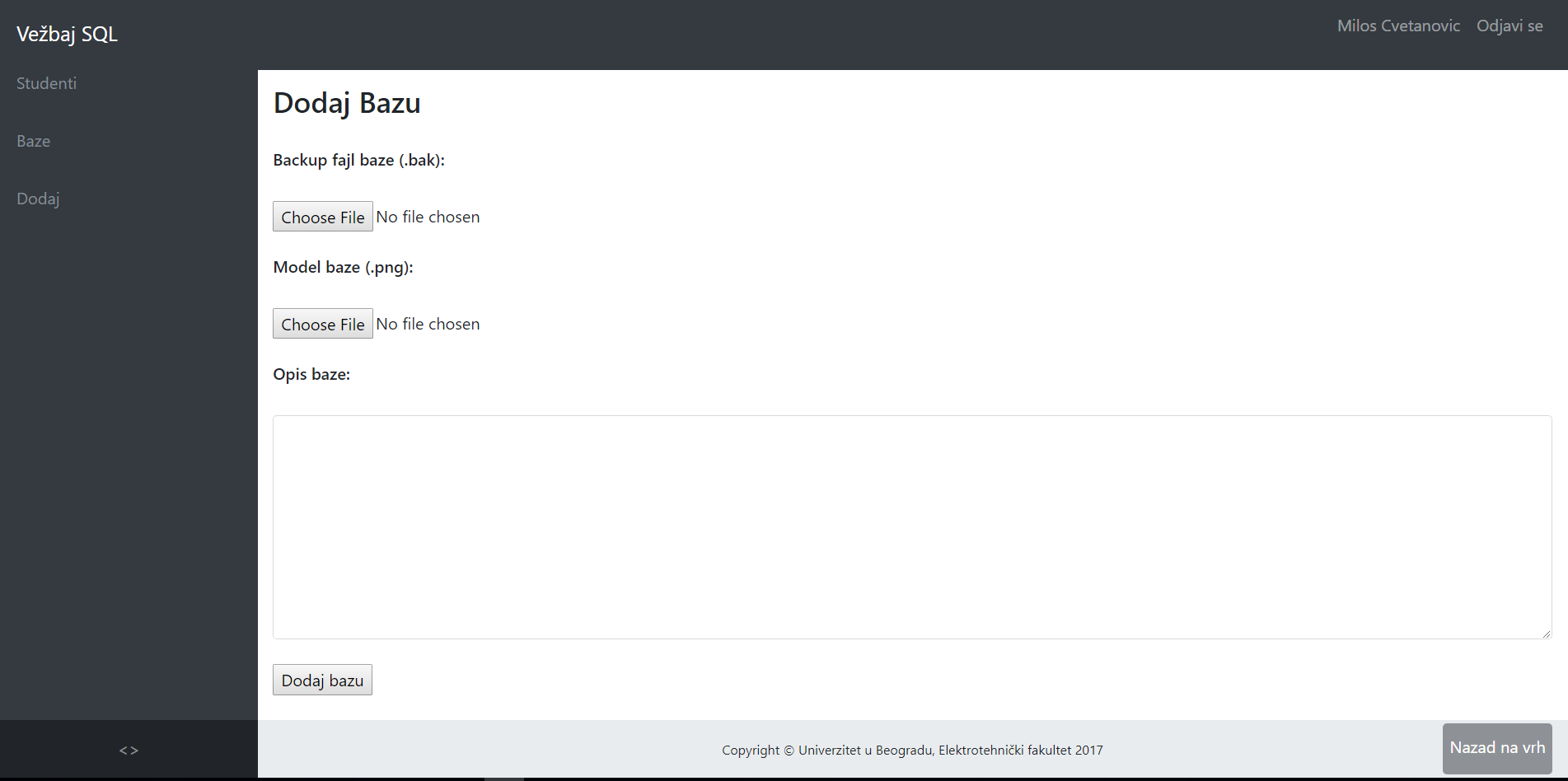
Na stranici „Dodaj“, u sadržaju stranice, implementirane su dve funkcionalnosti: dodavanje studenta i dodavanje baze.

Dodavanje studenta podrazumeva unošenje svih podataka koji opisuju studenta: „Korisničko ime“, „Lozinka“, „Ime“, „Prezime“ i „Broj indeksa“. Na ovaj način se novim studentima omogućava pristup aplikaciji. Takođe, novi student biće prikazan u tabeli na stranici „Studenti“.

Dodavanje baze je funkcionalnost koja omogućava da administrator postavi novu bazu na server. Nakon toga, studenti će moći da se povežu na tu bazu i nad njom vežbaju upite. Takođe, baza će se pojaviti i na stranici „Baze“. Administrator najpre mora da kreira MSSQL bazu pomoću nekog alata poput „Microsoft SQL Management Studio“ i zatim sačuva njenu rezervnu kopiju kao fajl sa ekstenzijom „bak“. Takođe, administrator treba da pripremi sliku UML modela kreirane baze u vidu fajla sa ekstenzijom „png“. Ovo se takođe može jednostavno generisati u pomenutom alatu. Tek kada je administrator pripremio neophodne fajlove, može da izvrši dodavanje. Dodavanje se sastoji od odabiranja pomenutih fajlova i unošenja tekstualnog opisa baze koju dodajemo. Naziv fajla koji predstavlja rezervnu kopiju baze (bez ekstenzije) određuje naziv koji će baza imati u aplikaciji. Ukoliko baza sa istim imenom već postoji na serveru, ona će biti obrisana pre dodavanja nove baze.

Sa leve strane nalazi se navigacija pomoću koje administrator može pristupiti stranama „Studenti“ i „Baze“ ili osvežiti trenutnu stranicu. U dnu navigacionog menija nalazi se taster označen sa „<>“ koji omogućava sklapanje tj. proširenje navigacionog menija.  
  


*Slika 3.10 Dodavanje novog studenta*



*Slika 3.11 Dodavanje nove baze*

1. Implementacioni aspekt aplikacije

U ovom poglavlju sagledaćemo neke od implementacionih aspekata aplikacije. Izdvojene su najznačajnije funkcionalnosti. Osvrnućemo se na ciljeve funkcionalnosti, neke od problema na koje sam naišao prilikom njihove implementacije, načine na koji su prevaziđeni i tehnologije koje su u tu svrhu korišćene. Takođe, u slučaju određenih funkcionalnosti, razmotrićemo veći broj mogućih rešenja i obrazložiti razloge zbog kojih su donete određene implementacione odluke.

* + 1. Organizacija baze podataka

Kao i većina veb aplikacija, ova aplikacija ima bazu podataka u kojoj čuva različite vrste podataka. Specifičnost ove aplikacije odnosno njene baze ogleda se u činjenici da su neki od podataka sa kojima aplikacija radi po tipu takođe baze podataka. Stoga u ovom tekstu treba obratiti posebnu pažnju na termin „baza podataka“ i kontekst u kom se taj termin koristi. Svrha ovog odeljaka je da detaljno razmotri organizaciju podataka i jasno napravi razliku između „prave“ baze podataka i „korisničkih“ baza podataka.

Korisničke baze podataka predstavljaju baze podataka nad kojima korisnici aplikacije (studenti) mogu da izvršavaju MSSQL upite. Iako su po tipu baze podataka, sa stanovišta aplikacije ove baze predstavljaju obične podatke.

Za razliku od korisničkih baza podataka kojih može biti više, postoji samo jedna „prava“ baza podataka. Ova baza postoji kao baza podataka na samom serveru pod imenom „BazaAplikacije“.

Striktno logički gledano, „BazaAplikacije“ je hijerarhijski na nivou iznad korisničkih baza. Ukoliko bi organizacija podataka aplikacije pratila logičku strukturu imali bismo potrebu da unutar baze „BazaAplikacije“ čuvamo korisničke baze. Iako semantički potpuno ispravno, u implementacionom smislu ovakav način organizacije uvodi nepotrebnu kompleksnost. Druga varijanta je da „BazaAplikacije“ i korisničke baze budu na hijerarhijski istom nivou tj. da budu baze podataka na serveru. Aplikacija kao svoju bazu koristi bazu „BazaAplikacije“. Unutar svoje baze, aplikacija čuva podatke o korisničkim bazama na osnovu kojih može da se poveže na bilo koju od korisničkih baza. Ovom indirekcijom pri povezivanju aplikacije na korisničku bazu ostvareno je logičko razdvajanje „prave“ baze od korisničkih, iako su one fizički jednake baze podataka na serveru.

„BazaAplikacije“ ima dve tabele: „Baza“ i „Korisnik“. Prva čuva podatke o korisničkim bazama, a druga kredencijale i lične podatke korisnika. Obe tabele imaju kolonu „Id“ koja predstavlja jedinstveni identifikator. Pored identifikatora „Baza“ ima i kolone: „NazivBaze“, „OpisBaze“ i „NazivFajlaSlike“, a „Korisnik“ kolone: „KorisničkoIme“, „Lozinka“, „Ime“, „Prezime“, „BrojIndeksa“ i „JeAdministrator“.

Obzirom da se korisničke baze kreiraju dinamički (u smislu da nisu poznate u trenutku pisanja aplikacije) mi njihovu strukturu ne poznajemo. Ovo uvodi neka ograničenja i predstavlja dodatnu komplikaciju nekih funkcionalnosti. Jedno od takvih ograničenja je pristup bazi. Naime, preporuka Majkrosofta je da se u radu sa bazama podataka koristi Entity framework. Najnovija verzija Entity framework-a podržava samo code-first pristup. Takav pristup zahteva da u trenutku pisanja modela u potpunosti poznajemo strukturu baze odnosno njenih tabela. Ovo dovodi do zaključka da Entity framework ne možemo koristiti za pristup korisničkim bazama podataka.

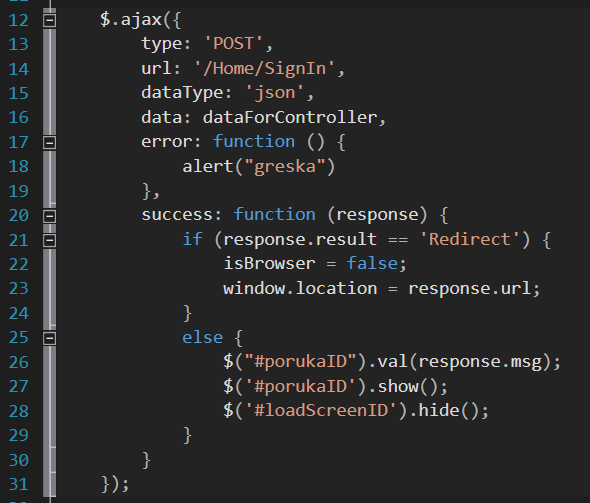
* + 1. Prijava korisnika

Funkcionalnost obuhvata serversku proveru kredencijala koje je korisnik uneo. Ukoliko su kredencijali ispravni korisnik treba da se preusmeri na odgovarajuću stranicu. Međutim, neophodno je definisati i ponašanje aplikacije u slučaju u kom kredencijali nisu validni. Očigledno, u takvoj situaciji korisniku treba omogućiti ponovni unos kredencijala.

Jedno pravolinijsko rešenje ove situacije bilo bi da nakon što server ustanovi da je korisnik uneo nevalidne kredencijale, preusmeri korisnika na stranu čiji je sadržaj identičan početnoj s tim da sadrži i dodatnu poruku koja korisnika obaveštava da uneti kredencijali nisu ispravni. Ovaj pristup nije dobar jer za istu funkcionalnost koristimo dve gotovo identične strane.

U cilju centralizacije koda, kao drugo rešenje, nameće se korišćenje jedne stranice s tim da se poruka o grešci prikazuje počevši od drugog uzastopnog pristupa stranici. Ovim pristupom je u potpunosti rešen problem necentralizovanog koda. Međutim, mana ovakvog pristupa je u tome što radi promene dela sadržaja stranice (poruka o grešci) pretraživač učitava kompletnu stranicu.

Idealno rešenje bilo bi kada bi na neki način mogli da, u konkretnoj situaciji, učitamo samo poruku o grešci, a da pri tom ostanemo na istoj stranici. Upravo ovakav pristup može se realizovati korišćenjem AJAX-a

  
*Slika 4.1 Deo implementacije JS funkcije signIn()*

Deo koda prikazan na slici poziva se iz JavaScript funkcije signIn (razlikovati od istoimene akcije kontrolera SignIn) čije se izvršavanje okida pritiskom na dugme „Prijavi me“ stranice za prijavu. Pre izvršavanja prikazanog dela koda se, između ostalog, u promenljivu dataForController smeštaju kredencijali koje je uneo korisnik. Prikazani deo koda predstavlja poziv jQuery.ajax funkcije. Parametar data predstavlja podatke koji će se poslati serveru kada se poziv izvrši. U ovom slučaju to su kredencijali koje je korisnik uneo. Parametar type definiše vrstu AJAX zahteva (GET ili POST). Obzirom da ovim zahtevom serveru šaljemo kredencijale koji su suštinski tajne informacije i da zahtevi tipa GET podatke prosleđuju kroz URL pretraživača, definisali smo da koristimo POST vrstu zahteva. U parametru url definišemo kontroler (HomeController – navodi se samo Home, sufiks Controller se podrazumeva) i akciju tog kontrolera (SignIn) za koju želimo da se izvrši ovim pozivom. Parametar dataType definiše tip povratnih podataka koje ćemo dobiti od servera. Ovaj tip i povratni tip akcije SignIn moraju biti usklađeni da bi poziv bio uspešan. Na kraju, parametri error i success definišu call-back funkcije koje će se izvršiti nakon što dobijemo odgovor od servera. Prvi parametar definiše akciju u slučaju uspešnog, a drugi u slučaju neuspešnog poziva. Obratimo pažnju na funkciju koja se izvršava u slučaju uspešnog poziva. Ova funkcija ima jedan argument (response) preko koga pristupa podacima koji predstavljaju odgovor servera. Ako od servera u delu result odgovora dobijemo string „Redirect“ to znači da su kredencijali validni i da korisnika treba preusmeriti na stranicu čiji je url zapisan u url delu odgovora (window.location = response.url;). U suprotnom, ne učitavamo novu stranicu već samo prikazujemo poruku o grešci ($("#porukaID").val(response.msg); $('#porukaID').show();)

Sagledaćemo još jedan aspekt ove funkcionalnosti. Tačnije ovaj aspekt važi za sve funkcionalnosti koje imaju nekakav AJAX poziv. Obzirom da ovakvi pozivi tipično čekaju na rezultat od servera u zavisnosti od raznih faktora (brzine internet konekcije, količine podataka koja se šalje u oba smera, performansi servera, kompleksnosti obrade koju server vrši…) vreme čekanja može da potraje. Poželjno je da korisniku za to vreme bude prikazano obaveštenje da server obrađuje zahtev i da je neophodno da sačeka. Ovo poruka obaveštenje realizovano je u vidu ekrana za čekanje. Sam ekran, iako se prikazuje preko cele stranice, ne predstavlja zasebnu veb stranicu već je kao i poruka o grešci deo stranice za prijavu.

Ekran za čekanje se na stranici za prijavu nalazi u vidu elementa <div id="loadScreenID" class="loadScreen">. Inicijalno po učitavanju stranice za prijavu element je skriven. Po ulasku u JavaScript funkciju singIn, jQuery naredbom $('#loadScreenID').show(), element se prikazuje. Element se na kraju uklanja unutar call-back funkcije za uspešan AJAX poziv naredbom $('#loadScreenID').hide().

Poslednji aspekt ove funkcionalnosti je pamćenje informacije o prijavljenom korisniku. Već u narednom odeljku videćemo jedan od razloga zašto nam je neophodno da kroz čitavu aplikaciju znamo koji korisnik je prijavljen. Jedan, doduše trivijalan, razlog jeste što želimo da prijavljenom korisniku, kroz čitavu aplikaciju prikažemo njegovo ime i prezime kao informaciju da je uspešno prijavljen. Obzirom da svaki korisnik ima sopstvenu sesiju, rešenje koje se nameće je da takve informacije čuvamo u objektu sesije. Akcija SignIn, nakon utvrđivanja da su kredencijali korisnika validni, pamti njegovo korisničko ime u objektu sesije.

* + 1. Izbor baze

Zamisao aplikacije je da student može da vežba upite nad više različitih baza. Takođe aplikacija treba da omogući da više studenata istovremeno radi sa aplikacijom preko različitih profila. Dakle, situacija u kojoj više studenata rade sa istom bazom je sasvim legitimna. Uzimajući u obzir ovakav vid korišćenja, postavlja se pitanje deljenja podataka odnosno deljenja korisničkih baza.

U situaciji kada se nad bazama izvršavaju SELECT upiti konflikata nema jer takvi upiti suštinski predstavljaju čitanje podataka - istovremeno čitanje podataka od strane više procesa ne predstavlja problem. Sa druge strane, dovoljno je da jedan korisnik izvrši INSERT, UPDATE ili DELETE kako bi došlo do konflikta. Ovakvo korišćenje je takođe legitimno pa je potrebno razrešiti konflikt koji ovde nastaje.

Jedno od mogućih rešenja je posmatrati ovaj problem kao takozvani „readers-writers“ problem koji se javlja kod konkurentnih i distribuiranih aplikacija. Jedna mana ovakvog pristupa je što u situaciji kada neki korisnik izvršava INSERT, UPDATE ili DELETE upit preostali korisnici su blokirani i ne mogu izvršavati operacije nad istom bazom.

Drugo rešenje jeste da se nakon što korisnik izabere bazu kreira kopija te baze koja će se dodeliti korisniku koji je izabrao. Na ovaj način, svaki korisnik u svakom trenutku radi nad svojom kopijom baze. Okruženja korisnika su u potpunosti izolovana i nema konflikata u pogledu deljenja podataka. Jedna od mana ovakvog pristupa je što zahteva dobre performanse servera. Operacija kreiranja kopije baze zavisi naravno od veličine baze, ali u proseku spada u zahtevne operacije i svakako je znatno zahtevnija od izvršavanja upita nad bazom.

Bez obzira na performanse i kompleksnost implementacije, drugo rešenje je jedino primenljivo. U slučaju prvog rešenja jedan mogući scenario je da korisnik A izvrši SELECT upit nad tabelom. Zatim, korisnik B izvršava DELETE upit nad istom bazom. Konačno, korisnik A ponovo izvršava isti upit. Postoji mogućnost da korisnik A u dva uzastopna i identična SELECT upita nad istom tabelom dobije različite rezultate. Ovakvo ponašanje, iako ispravno u pogledu konkurentnosti, nije ispravno u kontekstu logike same aplikacije.

Odabirom neke baze student poziva akciju OtvoriBazu kontrolera StudentController. Ova akcija kao argument ima naziv izabrane baze. Nakon što utvrdi da izabrana baza postoji, server treba da kreira kopiju baze za korisnika. Kako su sve korisničke baze na istom hijerarhijskom nivou tj. čuvaju se kao baze podataka na serveru, imena kopija baza moraju biti različita od originala i jedinstvena na nivou aplikacije. Imajući u vidu da se za jednu bazu i jednog korisnika kreira samo jedna kopija, kao i da svi korisnici imaju jedinstveno korisničko ime na nivou aplikacije, jedno validno ime baze kopije je ime originala na koje je nadovezano korisničko ime. Nakon što formira ime, StudentController poziva metodu CreateDatabaseCopy klase DatabaseController i prosleđuje joj ime izabrane baze i formirano ime njene kopije. Na slici 4.2 prikazan je deo implementacije metode CreateDatabaseCopy. Metoda koristi Transact-SQL upite kako bi najpre kreirala rezervnu kopiju, a zatim uradila oporavak rezervne kopije u novu bazu na serveru. Poslednji korak je brisanje rezervne kopije sa servera. Primetimo da komanda RESTORE DATABASE ima dodatnu klauzulu MOVE. Datoteka rezervne kopije ima ekstenziju .bak. Ovi fajlovi pored samog sadržaja baze podataka čuvaju i njene metapodatke. Između ostalog, tu je i ime same baze odnosno njenih logičkih fajlova. Bez klauzule MOVE, iako smo naveli novo ime baze, doći će do greške jer logički fajlovi već postoji na serveru (logički fajlovi originala). Deo koda pre samog RESTORE DATABASE upita iz fajla rezervne kopije čita imena logičkih fajlova kako bi se kasnije koristili u MOVE klauzuli.

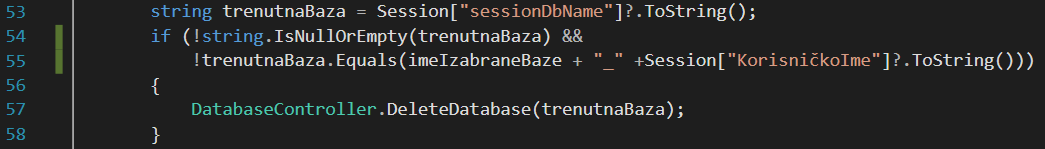
  
*Slika 4.2 Deo implementacije akcije CreateDatabaseCopy, DatabaseController.cs*

* + 1. Odjavljivanje

Rešenje iz prethodnog odeljka uvelo je kreiranje kopija baza za svakog korisnika. Na ovaj način otklonjen je problem deljenja podataka. Sa druge strane koristi se znatno veći broj resursa servera (procesora i memorije). Međutim, ono što možemo da ostvarimo jeste minimalno potrebno zauzeće memorije. Treba obezbediti da u svakom trenutku server čuva samo onoliko baza podataka koliko je neophodno za pravilno funkcionisanje aplikacije. Drugim rečima, u svakom trenutku, na serveru ne sme biti više baza od broja prijavljenih studenata (ne računajući bazu „BazaAplikacije“).

Da bismo ovo postigli neophodno je obrisati kopiju baze sa kojom student radi u situacijama kada student menja bazu nad kojom radi ili kada se student odjavljuje.

Treba voditi računa da pri promeni baze, student ponovo može izabrati istu bazu. U ovom slučaju kopiju baze ne treba brisati. Dakle, brisanje kopije baze u prvoj situaciji ne treba raditi odmah nakon povratka na stranicu za izbor baze, već tek nakon što student izabere novu bazu, pod uslovom da je novoizabrana baza različita od one nad kojom student radi. Ova logika implementirana je u akciji OtvoriBazu kontrolera StudentController. Kao i korisničko ime, ime baze nad kojom student trenutno radi se takođe čuva u objektu sesije.

  
*Slika 4.3 Deo implementacije akcije OtvoriBazu, StudentController.cs*

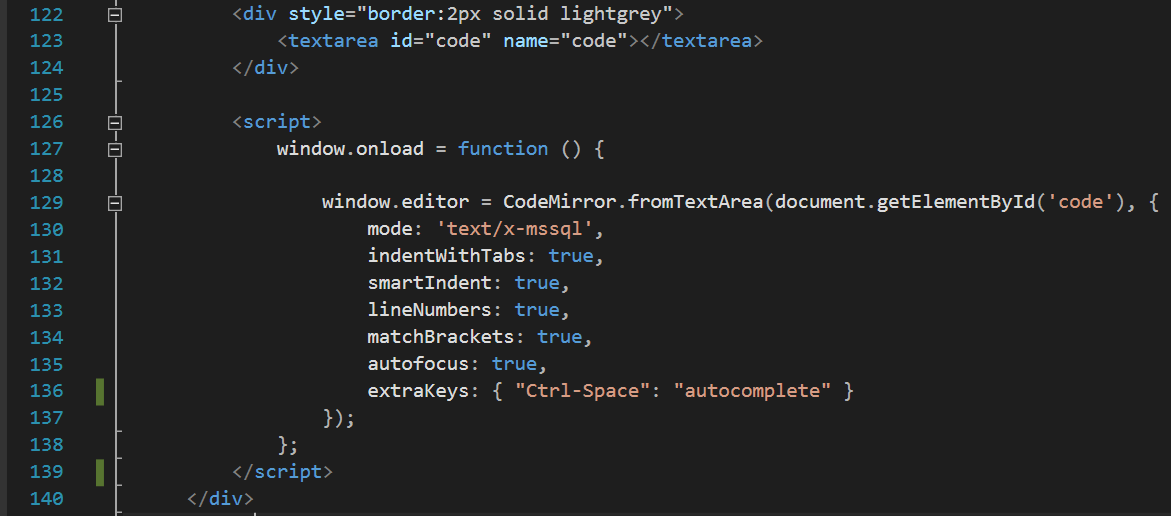
Student se može odjaviti iz aplikacije eksplicitno, pritiskom na dugme „Odjavi se“ ili implicitno, zatvaranjem veb pretraživača. U oba slučaja treba dealocirati objekat sesije i obrisati kopiju baze nad kojom student trenutno radi. Ova logika implementirana je u metodi SignOut kontrolera HomeController. U slučaju eksplicitnog odjavljivanja akcija se vezuje direktno za događaj pritiska na dugme „Odjavi se“. Detektovanje zatvaranja prozora pretraživača je nešto složenije. Ono što će se sigurno desiti pri zatvaranju prozor a jeste takozvani „unload“ stranice. Svojstvo „onbeforeunload“ HTML elementa „body“ omogućava da navedemo akciju koja treba da se izvrši pre samog „unload“ događaja. Tako da bi naizgled, rešenje bilo da na svim stranicama elementima „body“ dodamo svojstvo „ onbeforeunload="/Home/SignOut" “. Problem sa ovim rešenjem je što se „unload“ dešava pri svakom zatvaranju prozora, ali pored toga dešava se i pri svakom učitavanju nove stranice (tačnije pri izbacivanju stare). Da bi se SignOut izvršavao samo pri zatvaranju prozora, umesto direktnog poziva „unload“ događaj će zvati JavaScript funkciju „unload“ koja će pozvati SignOut samo ukoliko je uzrok „unload“ događaja zatvaranje prozora. JavaScript funkcija „unload“ i logika za detektovanje uzroka događaja implemenitrane su u „browserClose.js“.

  
*Slika 4.4 browserClose.js*

Pomenuta logika zasniva se ne postavljanju i ispitivanju vrednosti „flag“ promenljive isBrowser. Promenljiva isBrowser je globalna JavaScript promenljiva koja se inicijalno, po učitavanju stranice postavlja na „true“ vrednost. Ukoliko je vrednost ove promenljive „true“ u trenutnku događaja „onbeforeunload“ funkcija „unload“ će pozvati akciju BrowserClose kontrolera HomeController. Ova akcija zadužena je za odjavljivanje korisnika i brisanje kopije baze nad kojom je korisnik radio. Kako se ova akcija ne bi pozivala pri promeni stranice, u takvim situacijama isBrowser se prethodno postavlja na „false“. Korisnik može promeniti stranicu slanjem forme ili klikom na dugme koje vodi na novu stranicu. Slanje forme uvek vodi ka novoj stranici pa svako slanje forme obara isBrowser na flase. Sa druge strane akcija pridružena dugmetu ne mora voditi ka novoj stranici. Kako bi se napravila razlika, onim dugmadima koja vode ka novim stranicama pridružena je klasa redirectClass. Događaj click na bilo kom elementu koji ima ovu klasu takođe obara isBrowser na false. Ovim je u potpunosti obezbeđeno detektovanje zatvaranja prozora veb pretraživača.

* + 1. Izvršavanje upita

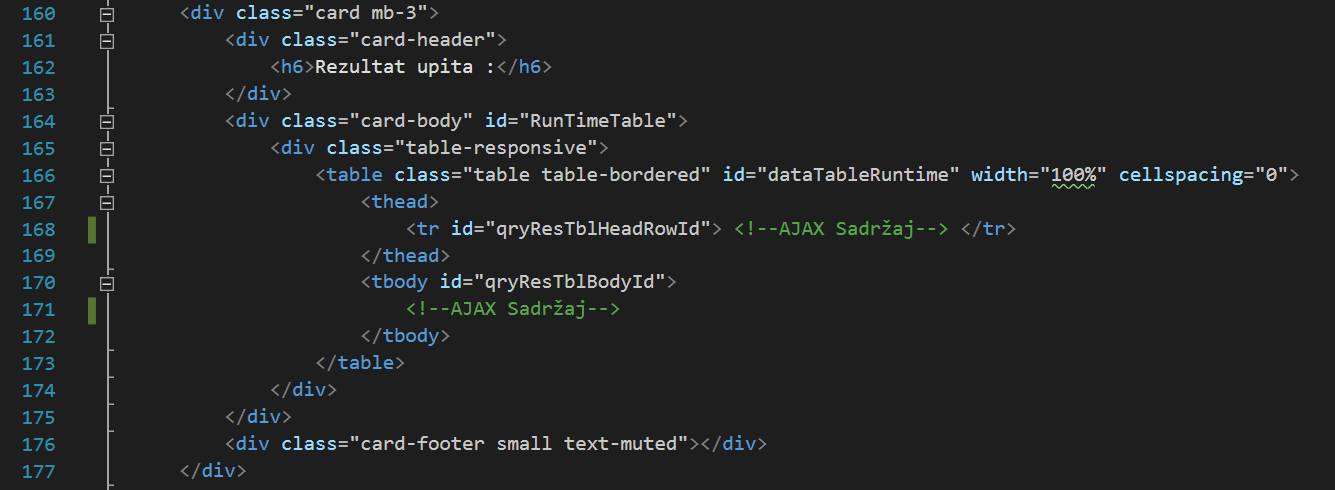
Prvi korak u izvršavanju upita jeste njegovo pisanje. Suštinski upit nije ništa drugo nego običan tekst i za unos upita dovoljno je jedno obično teksutalno polje. Međutim, tekstualno polje kao takvo nema mogućnost prepoznavanja ključnih reči i predlaganja istih. Kodiranje ovakve kontrole bi praktično zahtevalo kreiranje kompletnog leksera i parsera za MSSQL što prevazilazi obime ovog rada. Pokazuje se da za implementacijom tako kompleksne kontrole nema potrebe, obzirom da postoji nekoliko vrsta takvih open-source kontrola. Većina njih podržava veliki broj jezika. U ovom radu opredelio sam se za „CodeMirror“, budući da on omogućava da pored standardnog SQL-a prilagodi sintaksu MSSQL-u. Kreiranje ove kontrole prikazano je na 4.5.

  
*Slika 4.5 Deo implementacije pogleda StudentBazaView.cshtml – MSSQL editor*

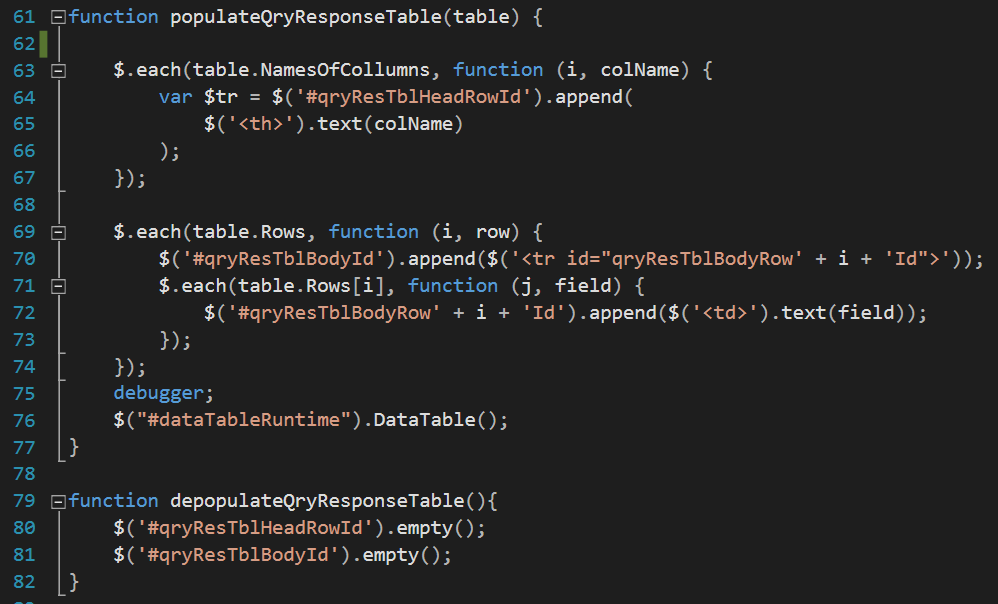
Kontrola se najpre definiše kao obično tekstualno polje <textarea>. Skript definisan ispod kontrole izvršava se kada se učita kompletna veb stranica. Skript koristeći CodeMirror biblioteku za JavaScript od običnog tekstualnog polja kreira moćan editor. Metoda fromTextArea koja se u tu svrhu poziva kao drugi argument prima konfiguracioni objekat editora. U njemu se definišu razne opcije editora od kojih je svakako najznačajnija mode kojom smo definisali da se radi o editoru za MSSQL. Pored nje postoje i opcije poput extraKeys, koje za određenu kombinaciju tastera na tastaturi okidaju navedenu akciju. Ovde smo konfigurisali da kombinacija tastera Ctr i Space okida automatsko dovršavanje ključnih reči.

Samo izvršavanje upita spada u jednostavnije funkcionalnosti. Razlog za to je što C# pruža relativno dobar i jednostavan API za izvršavanje MSSQL upita. Najpre je potrebno ostvariti konekciju ka bazi. Obzirom da u objektu sesije čuvamo ime baze nad kojom student trenutno radi, tu vrednost treba upotrebiti u stringu za ostvarivanje konekcije. Konekciju zatim otvaramo i iz nje kreiramo objekat komande. Tekstualni MSSQL upit se može proslediti kao parametar metodi za kreiranje objekta komande ili se može dodeliti odgovarajućem svojstvu već kreiranog objekta. Ovako kreirana i inicijalizovana komanda spremna je za izvršavanje. Ovde dolazimo i do jedine komplikacije. Sam API izvršavanje omogućava preko dve metode objekta komande. Prva metode je ExecuteReader i ona omogućava izvršavanje SELECT upita. Druga metoda zove se ExecuteNonQuery i ona omogućava izvršavanje INSERT, UPDATE ili DELETE upita. Očigledno je da izbor metode zavisi od samog sadržaja upita i da je u nekom trenutku potrebno parsirati sadržaj upita i odrediti o kojoj vrsti upita se radi. Pritiskom na dugme „Izvrši“ poziva se JavaScript funkcija executeSQL definisana u studentBaza.js. Funkcija vrši parsiranje i u zavisnosti od toga o kakvom se upitu radi poziva executeSelectSQL ili executeEditSQL. Ove dve funkcije takođe su definisane u studentBaza.js, a jedina razlika u njihovoj implementaciji je akcija kontrolera koju pozivaju. Obe pomenute akcije izvršavaju MSSQL upit, ali jedna koristi ExecuteReader, a druga ExecuteNonQuery. Parsiranje je moglo i da se izvrši na serverskoj strani. Tada bi se uvek pozivala ista akcija kontrolera, a onda bi u njenoj implementaciji prvo radili parsiranje, a zatim išli u jednu od dve grane od kojih bi jedna koristila ExecuteReader, a druga ExecuteNonQuery. U konkretnom slučaju nema značajne razlike jer je sama obrada jako male kompleksnosti. Međutim kroz čitavu aplikaciju teži se ka tome da se što više obrade prebaci na klijenta, kako bi server bio maksimalno rasterećen što je učinjeno i u ovom slučaju.

Ono što treba napomenuti je da, kao i u slučaju poruke o grešci pri prijavljivanju, pri izvršavanju upita nema potrebe za ponovnim učitavanje cele stranice. Izvršavanjem upita menja se samo deo stranice koji prikazuje rezultate upita i jedino taj sadržaj treba ažurirati. Dakle, još jednom rešenje su AJAX pozivi iz pomenutih JavaScript funkcija executeSelectSQL i executeEditSQL ka odgovarajućim akcijama kontrolera. Jedan deo odgovora od servera je ponovo tekstualna poruka i tu nema nikakve razlike u odnosu na poruku o grešci kod prijavljivanja korisnika. Pored tekstualne poruke, u slučaju SELECT upita rezultat može biti i cela tabela. U veb programiranju česti su slučajevi gde se sadržaj neke tabele popunjava rezultatom AJAX poziva. Međutim, ova situacija je specifična po tome što sama struktura tabele (njene kolone) nisu poznate i predstavljaju deo rezultata koji se dobija od servera. Dakle, potrebno je na osnovu rezultata AJAX poziva izmeniti strukturu stranice i izgraditi tabelu. Deo sadržaja koji je nepromenljiv (zaglavlja i okvir tabele) je ugrađen u stranicu pri učitavanju.

  
*Slika 4.6 Deo implementacije pogleda StudentBazaView.cshtml – tabela rezultata upita*

Ostatak sadržaja biće kreiran nakon svakog AJAX poziva za izvršavanje upita u delu označenom kao <!--AJAX Sadržaj-->. U svrhu kreiranja ovog sadržaja uvedena je pomoćna JavaScript funkcija populateQueryResponseTable definisana u studentBaza.js. Takođe, na istom mestu definisana je znatno jednostavnija funkcija depopulateQueryResponseTable. Njena svrha je da prilikom izvršavanje novog upita, obriše tabelu koja je bila rezultat prethodnog upita.

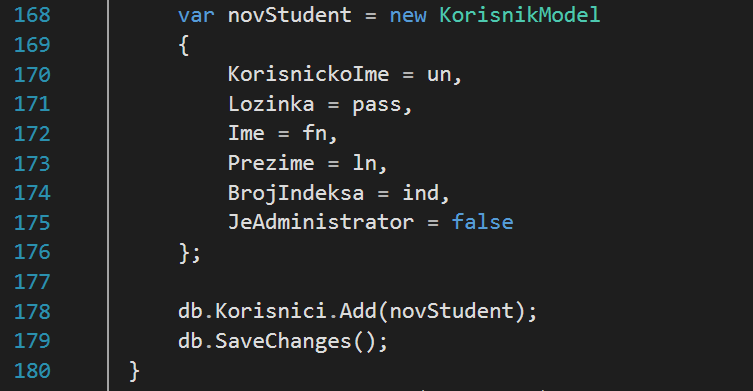
  
*Slika 4.7 Implementacija metoda populateQryResponseTable i depopulateQryResponseTable*

Funkcija populateQryResponseTable najpre popunjava zaglavlje tabele. Iterira kroz sva imena kolona koja je dobila kao deo odgovora servera i za svaku kolonu u <thead> element (koji referencira vrednoću njegovog id atributa) kreira po jedan <th> element čiji je sadržaj naziv kolone.  
Redovi se popunjavaju tako što se iterira kroz sve redove u rezltatu i najpre se za svaki red kreira novi <tr> element unutar elemnta <tbody>. Svakom <tr> elementu se postavlja jedinstvena vrednost atributa id tako što se u nju uključuje indeks iteracije. Zatim se unutar pomenute iteracije iterira kroz sva polja reda tabele, referiše tekući red i unutar njega kreiraju <td> elementi – po jedan za svako polje reda. Za iteraciju i kreiranje elemenata koriste se jQuery funkcije each i append.

Funkcija depopulateQryResponseTable referencira zaglavlje i telo tabele i za njih poziva jQuery funkciju empty koja uklanja sve elemente unutar elementa za koji je pozvana.

* + 1. Dodavanje studenta i baze

Obe funkcionalnosti realizovane su preko forme. Pri dodavanju studenta jedino će doći do promene u tabeli „Student“ baze „BazaAplikacije“. Kako nam je struktura ove tabele unapred poznata ispunjeni su svi uslovi za korišćenje Entity framework-a. Deo koda koji implementira ovu promenu dat je na sledećoj slici.

  
*Slika 4.8 Deo implementacije akcije UploadStudent, AdminController.cs*

Promenljive un, pass, fn, ln, ind predstavljaju vrednosti unesene na formi. Više nego očigledno je da je upotrebom Entity framework-a znatno olakšana interakcija sa bazom – nema potrebe za otvaranjem konekcije ka bazi, kreiranje komande, dileme da li pozivati executeReader ili executeNonQuery i najbitnije uopšte nema potrebe za poznavanjem SQL-a jer se on kao takav i ne koristi. Nažalost, kao što je već pomenuto, iako elegantan ovaj pristup zahteva poznavanje strukture tabela baze u trenutku pisanja koda pa je zbog specifičnosti same aplikacije njegova primena bila moguća samo u nekolicini slučajeva.

Na sličan način, prilikom postavljanja nove baze na server se popunjava novi ulaz u tabeli „Baze“. Dodatno, na server se postavljaju dva nova fajla. U direktorijumu Content čuva se slika UML modela baze, a u folderu App\_Data privremeno se čuva .bak datoteka baze koju postavljamo. Nakon toga baza se postavlja na server, a .bak datoteka briše. Poslednja dva koraka realizovana su na gotovo identičan način kao i poslednja dva koraka u kreiranju kopije baze u odeljku 4.1.3

1. Zaključak

Neminovno je da će tehnologija napredovati i samim tim će se internet dalje razvijati. Brzine protoka biće sve veće, a veze stabilnije. Već sada možemo videti koliko svetski softverski giganti ulažu u povećanje pokrivenosti internetom. Svesni su da na taj način uvećavaju svoje tržište. Sve ovo ukazuje na značaj koji će veb aplikacije imati u budućnosti.

Samo u sferi obrazovanja potencijal koje ovakve aplikacije imaju je praktično neograničen. Aplikacija „Vežbaj MSSQL“ je samo demonstrativan primer kako korišćenje internet tehnologija može doprineti kvalitetu obrazovanja.

Kada je reč o ovoj veb aplikaciji ostaje da se vidi kako će biti prihvaćena od strane korisnika. Sigurno je da ima mesta za poboljšanja. Uzimajući u obzir da se identične manipulacije baze mogu uraditi na više načina, jedna korisna funkcionalnost bila bi mogućnost da se kreiraju zadaci koje bi korisnici rešavali, a onda imali mogućnost da kroz aplikaciju verifikuju ispravnost njihovih rešenja. Performanse su još jedan aspekt koji može da zahteva poboljšanje. Kreiranje kopije baze je zahtevna operacija i treba videti kako se aplikacija ponaša sa većim brojem korisnika.

Cilj ovog rada je da demonstrira mogućnosti korišćenja veb aplikacija u svrhe obrazovanja. Nažalost, ta mogućnost je u velikoj meri zapostavljena. Da bi jedna ovakva aplikacija dostigla nivo na kom bi se koristila kao nastavno sredstvo, pored programerske veštine potreban je i veliki nivo poznavanja same materije, kao i najboljeg načina za prenošenje te materije onome ko je izučava. Ovo možda deluje kao kompleksan zadatak, ali rezultati koje bi šira primena ovakvih aplikacija donela bi sigurno opravdali uložen trud.

1. Literatura

* Laslo Kraus, „Programski jezik C# sa rešenim zadacima“, Akademska misao, Beograd 2016
* Douglas Crockford, „JavaScript: The Good Parts“, Yahoo Press, decembar 2008
* Elektronska forma: Rick Anderson: „Getting Started with ASP.NET MVC 5“  
  <https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started>
* Elektronska forma: „jQuery API“   
  <http://api.jquery.com/>
* Elektronska forma: „Entity framework Code First to a New Database“ <https://msdn.microsoft.com/en-us/library/jj193542(v=vs.113).aspx>
* Elektronska forma: „Bootstrap Documentation“ <https://getbootstrap.com/docs/4.0/getting-started/introduction/>
* Elektronska forma: Bootstrap tema „Sb-admin“   
  <https://startbootstrap.com/template-overviews/sb-admin/>
* Elektronska forma:   
  CodeMirror „User manual and reference guide“  
  <http://codemirror.net/doc/manual.html>   
  CodeMirror „Basic usage“  
  <http://codemirror.net/doc/manual.html#usage>

CodeMirror „Configuration“  
<http://codemirror.net/doc/manual.html#config>