



Univerzitet u Novom Sadu
Tehnički fakultet „Mihajlo Pupin“
Zrenjanin



SEMINARSKI RAD
Predmet: Programski prevodioci

**Upoređivanje gramatike C# i Java" - SEMANTIKA DOMENA:
Evidencija fudbalskih igrača - tip 5/A**

Predmetni nastavnik:
doc. dr Ljubica Kazi

Autor rada
Aleksa Cakić SI 23/17

Zrenjanin 2020.

Sadržaj

1	Uvod.....	4
2	Kratki pregled C# programskog jezika.....	4
3	Kratki pregled MSSQL-a.....	5
4	Kratki pregled PostgreSQL-a.....	7
5	Zadatak.....	10
5.1	C# Radno okruženje.....	10
5.2	JAVA Radno okruženje.....	11
6	Korisničko uputstvo.....	12
6.1	Dizajn korisničkog interfejsa.....	14
7	Opis implementacije.....	19
7.1	C# - implementacija.....	19
7.1.1	Baza podataka.....	19
7.2	C# - ASP.NET implementacija.....	20
7.2.1	Ključni delovi koda sa objašnjenjem.....	29
7.3	JAVA implementacija.....	34
7.3.1	Baza podataka.....	34
7.3.2	Implementacija Java back-enda – Spring.....	37
8	Greške.....	43
8.1	Run time greške.....	43
8.1.1	Run time greške u C#.....	44
8.1.2	Run time greške u Javi.....	45
8.2	Logičke.....	45
8.2.1	Logičke greške u C#.....	45
8.2.2	Logičke greške u Javi.....	46
8.3	Sintaksičke.....	47
8.3.1	Sintaksičke greške u C#.....	47
8.3.2	Sintaksičke greške u Javi.....	47
8.4	Izgled poruka kada kompajler naleti na grešku.....	48
8.4.1	JAVA.....	48
8.4.2	C#.....	48
8.5	Izgled poruka kada se otkloni greška.....	49
8.5.1	JAVA.....	49
8.5.2	C#.....	49
9	Pravila I BNF/EBNF prikaz.....	50
9.1	Run Time greške.....	50
9.1.1	Java.....	50
9.1.2	C#.....	50
9.2	Leksičke greške.....	50
9.2.1	Java (slika 38).....	50
9.2.2	C# (slika 39).....	51
9.3	Semantičke greške.....	51
9.3.1	Java.....	51
9.3.2	C#.....	51
10	Zaključak.....	52

11 Izvorni kod aplikacije.....	52
12 Literatura.....	53

1 Uvod

Microsoft je od 2014 godine menjajući kompajler u C# programski jezik; znači danas C# programski jezik kompajlira kompajler za C# programski jezik; omogućio programerima da kompajler kodnog imena „Roslyn“ kao kompajler nije više zatvorena crna kutija, već je open-source kod. Ali je isto tako time omogućio da se prave još bolje promene na samom programskom jeziku, pa tako danas imate male promene u kodu koje prave velike promene i umanjuju pisanje koda ali će ih biti sve više. Već sad možete izguglati neke promene koje Microsoft već najavljuje pored mnogobrojnih promena koje su ove godine uvedene.

Za razliku od C# programskog jezika, C++ programski jezik se smatra nešto bržim što danas i nije toliko relevantno s obzirom da su kompjuteri danas mnogo brži, osim možda u nekim segmentima kad su u pitanju programiranje igrica, ali ništa toliko zastrašujuće što se ne može nadomestiti. Sa C++ programskim jezikom možete da programirate i na Linux operativnom sistemu i na OS X platformi jer C++ programski jezik komunicira direktno sa operativnim sistemom i kompajlira odmah mašinski kod, dok je C# zavisao od Microsoft .Net Framework-a koji je posrednik između vaše aplikacije i operativnog sistema. Međutim vi u C++ ne možete da programirate na primer WPF – Windows Presentation Foundation aplikacije dok je danas programirati Windows Forms zastarela tehnologija i pored toga što se mora znati radi starijih projekata. Sa C# programskim jezikom možete definitivno da uradite više i da kodirate mnogo jednostavnije i lakše nego sa C++ programskim jezikom i to uvek imajte na umu.

2 Kratki pregled C# programskog jezika

Sada već davne 1995. godine je kompanija Sun Microsystems lansirala programski jezik Java, a koji se definiše kao objektno - orijentisani. Kasnije je kompanija Oracle otkupila pomenutu, te tako i postala zvaničan vlasnik ove tehnologije.

Princip na kome je ovaj programski jezik kreiran je primarno usmeren na jezik Oberon, ali i na mnoge druge, dok mu je sintaksa nalik C i C++ programskim jezicima. Ipak, ovaj tip programskog jezika ima značajno strožiji princip prevođenja, a potpuno je nezavisan od platforme na kojoj se radi. Posebna pažnja prilikom njegovog kreiranja je usmerena na upravljanje memorijom, a koja je kod Java programskog jezika u velikoj meri jednostavnija. Smatra se da je razlog za to vezan za veliku popularnost C programskog jezika.

Svrha primene ovog jezika je na prvom mestu vezana za davanje konkretnih instrukcija određenom uređaju, a kako bi on bio u mogućnosti da izvrši određene komande.

Upravo se Java programski jezik smatra jednim od najpopularnijih, a veruje se da je osnovni razlog za to vezan za činjenicu da se njegovo korišćenje uopšte ne naplaćuje. Budući da se korišćenje mnogih drugih programskih jezika gotovo uvek naplaćuje, sasvim je jasno i zbog čega upravo Java uživa toliku popularnost.

Druga prednost primene ovog programskog jezika se odnosi na činjenicu da se komande koje navodi Java, praktično rečeno mogu izvršavati na gotovo bilo kom računaru, jer je ona kreirana tako da uopšte ne zavisi od platforme koja se koristi na konkretnom tipu uređaja.

Svakako je prednost i ta što se baš Java koristi kada je potrebno pristupiti razvoju različitih vrsta aplikacija. Uz primenu tog programskog jezika se mogu praviti takozvane GUI aplikacije (Graphical User Interface), kao i apleti aplikacije (Applets), ali isto tako i one koje su poznate kao konzolne. Kada se karakteristike ovog programskog jezika uporede sa drugima, stiže se jasan utisak da je Java izuzetno jednostavna za korišćenje, pa se i to svakako može svrstati u jednu od mnogobrojnih prednosti koje se na njenu primenu odnose.

Zanimljivo je navesti i podatak da naziv ovog programskog jezika vodi poreklo upravo od istoimenog ostrva, a koje se nalazi na teritoriji Indonezije. Iako je Java primarno programski jezik, činjenica je da se ovim pojmom označava isto tako i soterska platforma, koja je zadužena da pokreće aplikacije koje su u ovom programskom jeziku izrađene.

Neretko se događa da se pojam Java i JavaScript mešaju, ali za to ne postoji razlog, sobzirom na to da su u pitanju dva potpuno drugačija programska jezika, čije karakteristike i funkcije se svakako razlikuju. Činjenica je da ovaj programski jezik ima brojne prednosti, ali naravno da su prisutne i određene mane, odnosno nedostaci kada je u pitanju njegova primena. Najznačajniji nedostatak se odnosi na brzinu njenog rada, uzevši u obzir da se ova tehnologija relativno sporo pokreće, a u poređenju sa drugima koje su joj na neki način slične.

3 Kratki pregled MSSQL-a

Bez obzira koji programski jezik učite ili sa kojim programskim jezikom programirate iz hobija ili poslovno; od svakog programera se očekuje da poznaje rad sa bazama podataka kao i osnove strukturiranja relacionih baza podataka. Baza podataka vam je najjednostavnije rečeno kolekcija

podataka smeštena u elektronskom formatu. Na engleskom jeziku se baza podataka kaže database ili skraćeno db. Inače prema Wikipedia-iji; baza podataka je organizovana kolekcija podataka za brzo pretraživanje i pristup; koja zajedno sa sistemom za održavanje i administraciju, organizovanje i memorisanje tih podataka čine sistem baze podataka. U školama će vam reći da je baza podataka kolekcija podataka koja se zapisuje u SQL server. Sve su ove definicije tačne ali je vama najvažnije da shvatite da je dobro organizovana baza podataka rešenje pola vašeg programerskog posla. Sve poznate baze podataka poput SQL, MySQL ili Oracle Database baza podataka; koriste isti standard za rad sa podacima i sve podržavaju SQL programski jezik. Kod Microsoft-a je to T-SQL i on je važan aspekt bez obzira na pojavu upitnog integrisanog jezika LINQ-a koji naveliko danas menja način programiranja ali definitivno nije zamena. T-SQL je jezik baza podataka i ne možete ga preskočiti već ga morate znati. Ali zato kad jednom savladate rad sa bazama podataka, lako će te moći koristiti sve vrste baza podataka jer funkcionišu na istim standardnim principima. Ne morate vi biti stručnjak za baze podataka ali neke osnovne stvari morate poznavati kako bi ste uopšte mogli da koristite baze podataka u vašim programima, sajtovima, servisima ili sistemima. U školama računara se uglavnom rad sa bazama podatka ne svrstava u C# programski jezik, ali se uči uporedo. Tako ću vam i ja uporedo sa postovima C# programskog jezika pisati i postove za rad sa Microsoft SQL Server 2016 Express serverom.

(Microsoft SQL Server 2016)

Pre su programeri uglavnom koristili tekstualne ili binarne datoteke da skladište informacije koje koristi njihov program. Međutim na taj način podaci su se samo gomilali, ponavljali; zauzimali su dosta memorije; mnoge kolone su ostajale prazne, svaka datoteka je u suštini predstavljala samo jednu tabelu i zato se javila velika potreba za programima i serverima koji su pored skladištenja podataka mogle da uvedu pre svega organizaciju u podacima ali i mnoštvo tehnoloških robusnih novina koje danas serveri za baze podataka nude. Prva baza podataka sa kojom ste se možda susretali kroz Microsoft Office jeste Microsoft Access baza podataka. I ona takođe obuhvata tabele, poglede, uskladištene procedure, funkcije i druge objekte koje su neophodni za pravljenje sistema podataka ali navedena baza podataka je desktop aplikacija i može biti veoma problematična kad su u pitanju mnoštvo objekata, tabela, relacija i podataka i pristupanjem istih preko mreže. Zato je pametnije odmah koristiti neku od edicija

Microsoft SQL Server baza podataka. Već više od deceniju postoje razne verzije i edicije Microsoft SQL Server-a , često se koriste starije verzije i one se stalno menjaju poput:

- SQL Server Express Edition
- SQL Server Workgroup Edition
- SQL Server Developer Edition
- SQL Server Standard Edition
- SQL Server Enterprise Edition
- SQL Server Mobile Edition

Naravno većina edicija se razlikuje pored tehničkih alata, ograničenja i mogućnosti; tako isto i po godinama izdanja. Microsoft SQL Server 2005 Express Edition i Microsoft SQL Server 2016 Express Edition se takođe razlikuju iako se te razlike često ne vide. Microsoft SQL Server 2016 Express je odličan izbor pre svega za početnike i zato što je besplatan dok su vam za druge edicije potrebne licence i mogu vam reći da one nisu ni malo jeftine. Zato moj izbor za vas je definitivno Microsoft SQL Server 2016 Express. Dobra vest je da Microsoft SQL Server 2016 Express pored 1 GB RAM-a, jednog procesora, nudi ograničenje skladištenja podataka na 10 GB. U ranijim verzijama ograničenje je bilo 4 GB. Sada dolazi sa Advanced Services i nećete imati komplikacije sa pravljenjem dijagrama.

4 Kratki pregled PostgreSQL-a

PostgreSQL ili jednostavnije Postgres je vrsta objektno-orijentisanih relacionih sistema za upravljanje bazama podataka (SUBP), pod open source licencom (otvoreni kod). Smatra se jednom od najpouzdanijih baza podataka. Najčešće koristi za web aplikacije i web baze podataka. Reklo bi se da je najveći konkurent MySQL-u.

PostgreSQL je bazično razvijen za rad na UNIX platformama, ali je portovan i na Linux, Windsows, macOS.

Prva zvanična verzija je objavljena 29. januara 1997. godine.

Inicijalno je razvijen u programskom jeziku C, ali takođe ima podršku za integraciju sa programskim jezicima kao što su Python, Perl, .NET, C++, Java, PHP, Ruby on Rails i drugi.

Postgres je nasljednik Ingres-a koji je bio jedan od sistema baza podataka, razvijen između 1977. i 1985. godine. Zvanično, autor PostgreSQL-a je Michael Stonebraker, profesor na Kalifornijskom univerzitetu u Berkliju (UCB). Stonebrakerova ideja je bila da izgradi napredniju verziju Ingres-a koja je robusnija, uz bolje performanse. Stonebraker i njegove kolege u UCB-u su osam godina (1986-1994) razvijali sistem Postgres baze podataka. Stonebrakerove kolege Andrei Iu i Jolli Chen su dodatno poboljšali razvijeni sistem zamjenom POSTQUEL upitnog jezika (query language) sa popularnijim i najčešće korišćenim SQL.

Ta poboljšana verzija nazvana je Postgres95. Nakon toga, 1996. godine, Postgres95 je po prvi put ušao u softversku industriju i postao jedan od najrobusnijih i najčešće korišćenih servera otvorenog programskog koda (open source).

Osobine i prednosti PostgreSQL-a

Open Source SUBP (DBMS – Database Management System) – Prva glavna prednost korišćenja Postgres-a je to što je open source i može se prilagoditi prema zahtevima developera. Ova mogućnost prilagođavanja je izuzetno korisna u razvoju velikih aplikacija.

Velika razvojna zajednica (iliti velik community) – Postgres je na tržištu već više od 15 godina i njegova zajednica je u ovom trenutku izuzetno velika, što samim tim znači dobru podršku i pomoć pri rešavanju problema vezanih za rad i uporebu.

Isplativost – Postgres je izuzetno ekonomičan i ne zahteva mnogo obučavanja korisnika kako bi se naučilo kako koristiti i programirati za ovu bazu podataka. Takođe, zahtevi za održavanje i podešavanje Postgres baze podataka su relativno mali u odnosu na druge sisteme za upravljanje bazama podataka.

Portabilnost – Dobra stvar u Postgresu je to što je portabilan i prenosiv sa gotovo svim glavnim platformama i programskim jezicima. Ova baza podataka je idealna za aplikacije namenjene višestrukim platformama.

Alatke za razvoj i GUI – Server Postgres baze podataka ne zahteva obimne konfiguracije komandne linije. Razvijen je nekoliko alata i GUI interfejsa koji vam mogu pomoći u jednostavnoj instalaciji i upravljanju serverom baze podataka.

Pouzdanost i stabilnost – Postgres je svetski priznat kao najsigurnija i stabilnija baza podataka. Šanse da se uništi baza podataka su minimalne i čak i ako se baza podataka sruši, postoje načini i funkcije koje vam omogućavaju da obnovite i vratite podatke.

Nedostaci PostgreSQL-a

Performanse – za jednostavne operacije čitanja i brisanja, PostgreSQL postiže lošije rezultate u odnosu na, na primer, MySQL.

Popularnost – definitivno da nije toliko zastupljen, u odnosu na druge sisteme baza podataka i samim tim nema tako jaku podršku.

Hosting – nije toliko zastupljen kod provajdera i pružaoca hosting usluga. Instalacija i pripremanje projekta

Prvi primer projekta je u C#.

Izrada projekta je vrlo laka i brza, i ne treba biti uopšte upoznat sa tehnologijom kako bi se uradio kompletan projekat, čime se implicira lakoća. Ovo je možda i najbolja tehnologija za apsolutno početnike da se upoznaju sa Web servisima, ali definitivno ne za dalji razvoj.

Prateći slike, projekat se dalje razvijao:

5 Zadatak

Zatak je uporediti gramatike dve naizgled slična jezika kroz primer aplikacije koja evidentira igrače i timove. Uz pomoć C#-a i frejmvorka ASP.NET i Jave i javinog Spring frejmvorka, u kombinaciji sa Angular 9 klijentskom stranom, treba predstaviti podatke.

Akcent ovog zadatka je poređenje gramatike i rada oba jezika uz slobodu programera da ubaci frejmvorke radi modernizacije.

U ovom seminarskom radu će se objasniti šta čini ova dva jezika toliko različitim, ali ključnim za internet programiranje, kako se ponašaju, kako reaguje na greške, kako te greške mogu da se izbegnu i zdrave navike kako bi se dostavio čist kod.

5.1 C# Radno okruženje

VISUAL STUDIO COMMUNITY 2019

Visual Studio je integrisano programsko okruženje, programirano od strane kompanije Majkrosoft. Visual Studio se koristi za programiranje računarskih igara, programa (Metro UI, desktop), veb-sajtova, veb-servisa i veb-aplikacija na Microsoft Windowsu. Visual Studio koriste Majkrosoftovi programski softveri:

- Windows Forms;

- Windows API;

- Windows Presentation Foundation;

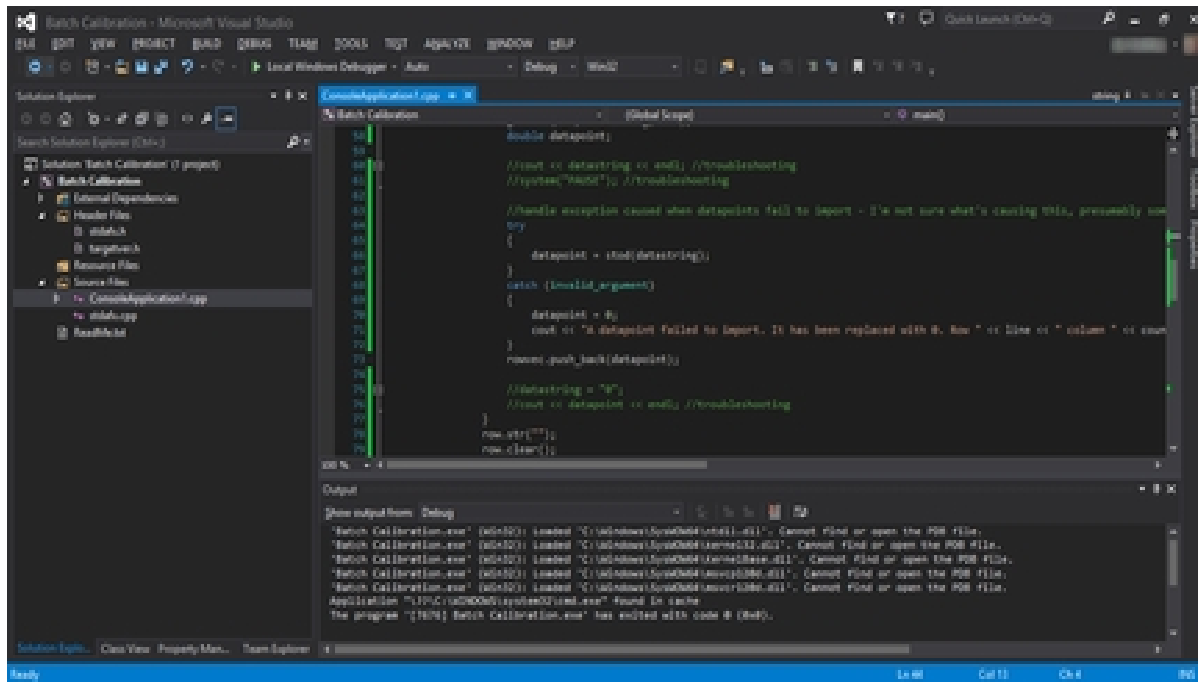
- Windows Store;

- Microsoft Silverlight;

Visual Studio podržava 36 različitih programskih jezika. Ugrađeni jezici uključuju C, C++ i C++/CLI (putem Visual C++), VB.NET (putem Visual Basic .NET), C# (putem Visual C#), F# (kao za Visual Studio 2010) i TypeScript (kao za Visual Studio 2013).

Podrška za druge programske jezike kao što su Python, Ruby, Node.js, i M dostupna je putem jezičkih usluga koje se instaliraju odvojeno. Visual Studio takođe podržava XML/XSLT, HTML/XHTML, JavaScript i CSS, Java (i J#).

Microsoft pruža i besplatnu verziju programa Visual Studio pod nazivom Community Edition koja podržava dodatke i dostupna je bez ikakvih troškova.

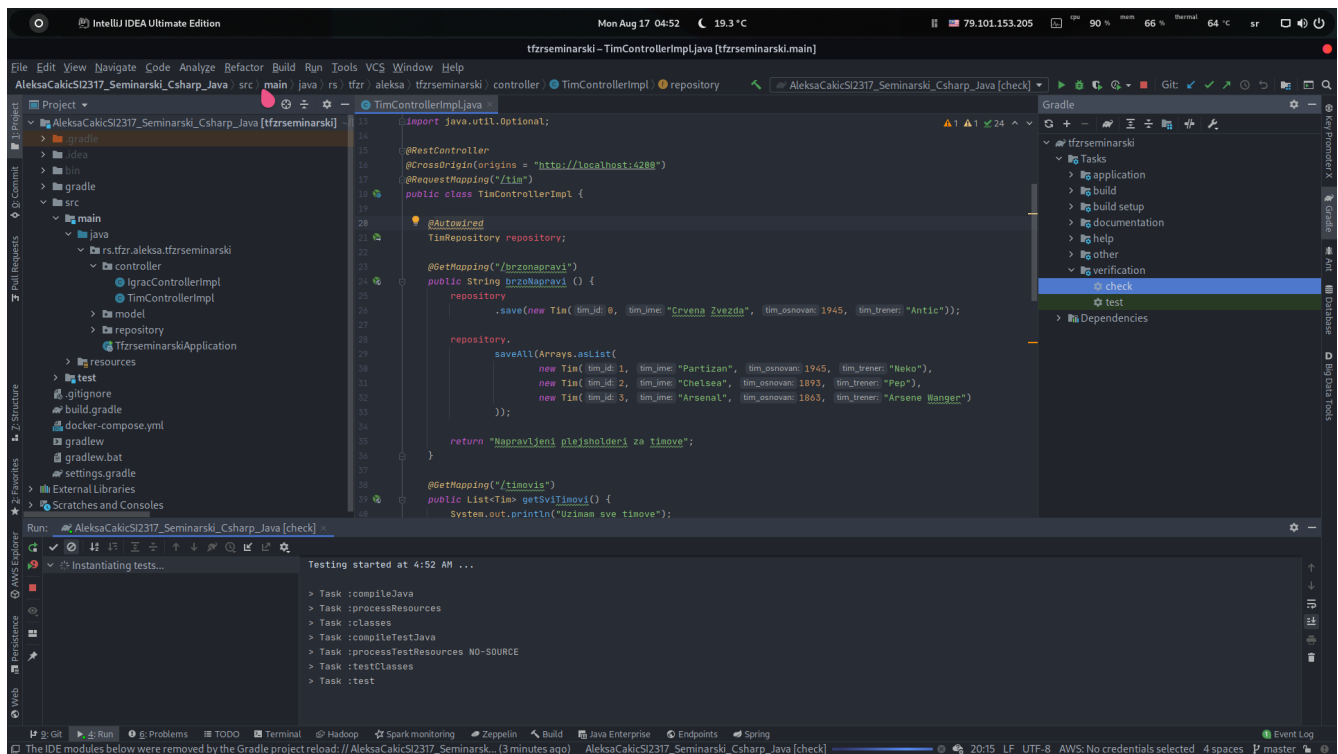


Slika 1: Izgled VS IDE-a

5.2 JAVA Radno okruženje

IntelliJ IDEA 2020 by JetBrains

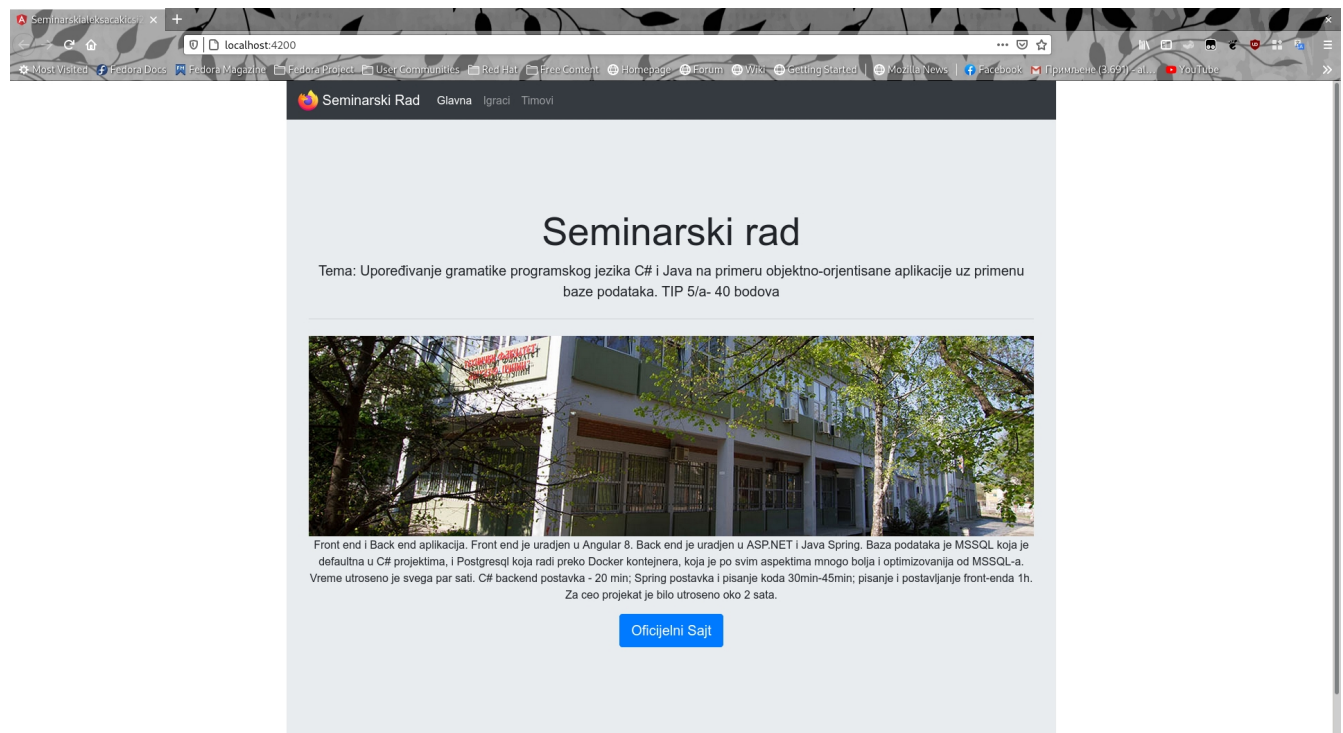
IntelliJ IDEA je alat koji predstavlja integrisano razvojno okruženje (Integrated Development Environment-IDE) za kreiranje softvera. Razvoj savremenih aplikacija uključuje upotrebu više jezika, alata, okvira i tehnologija. Daje podršku za razvoj programa primenom različitih programskih jezika, pri čemu se najčešće koristi Java. Java kod se izvršava korišćenjem Java Virtuelne Mašine (JVM). JVM je jezgro Jave, to je apstraktna mašina koja postoji samo u memoriji. Java je jezik koji se prevodi i interpretira: (java izvorni kod-*.java fajl, nakon kompajliranja (kompajlerom javac) se prevodi u bajtkod -*.classfajl, bajtkod se pomoću java interpretera dešifruje i interpretira u izvršni kod).



Slika 2: Izgled IntelliJ IDE-a

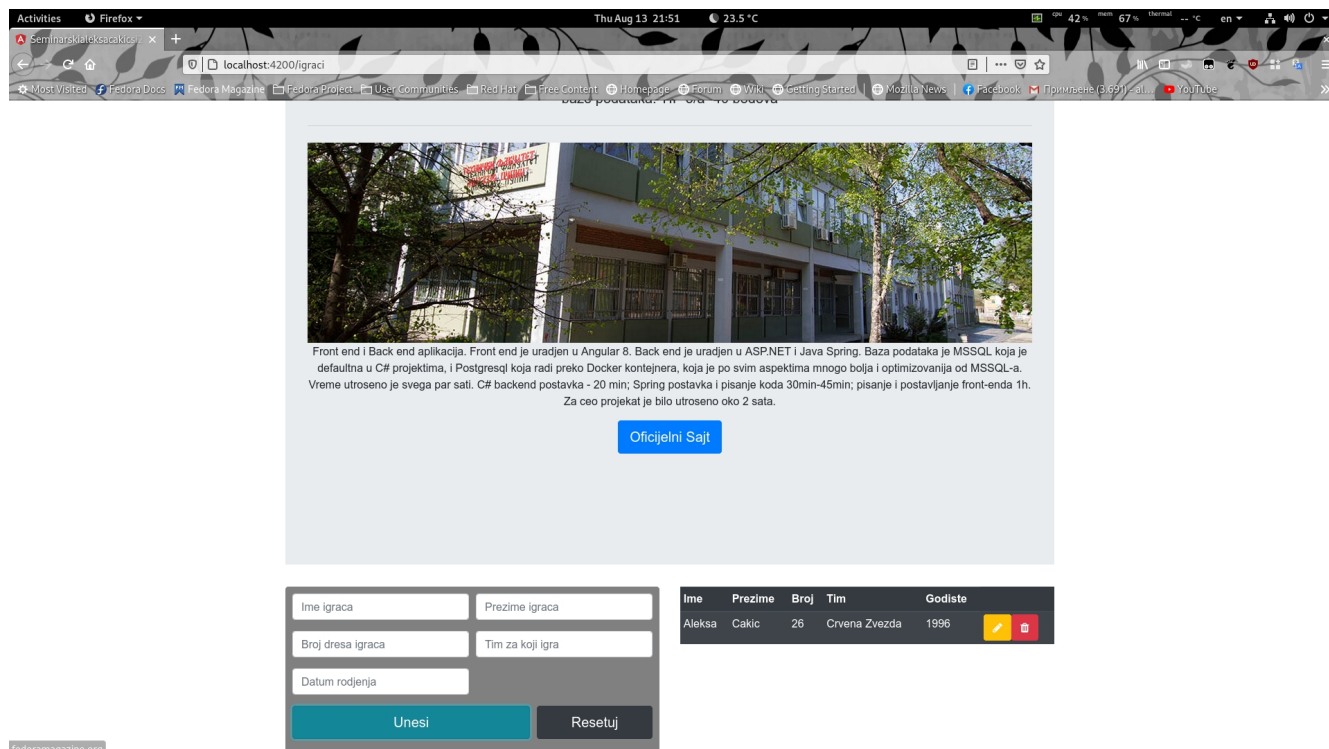
6 Korisničko uputstvo

Samo korisničko uputstvo je napravljeno sa namerom da bude intuitivno. U front-end delu aplikacije, programer se postarao da korisnik može da sleti na naslovnu stranu sa kratkim opisom zadatka. U navbaru, korisnik može navigovati sebe po delu stranice u zavisnosti da li želi da doda, menja ili briše igrače ili timove.



Slika 3: Izgled naslovne strane u Firefox browseru

Dalje, kada korisnik skroluje dole, može videti forme koje imaju plejsholder u sebi, te se tim untuitivnim gestom navode da unesu potrebni kveri.

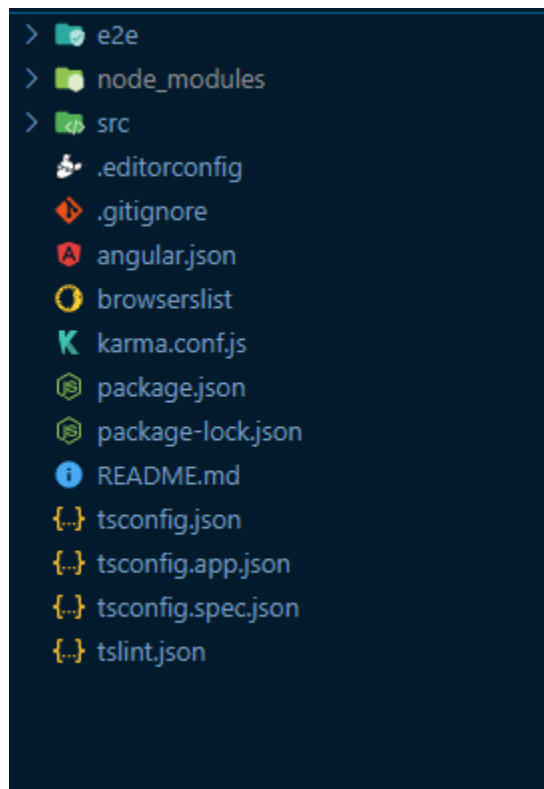


Slika 4: Unešen kveri i prikaz unešenog

6.1 Dizajn korisničkog interfejsa

*Dizajn korisničkog interfejsa je potpuno isti za obe aplikacije jer koriste isti front-end izvor.

- *Angular 9* je osnova za samo funkcionisanje front end dela aplikacije. Kao izuzetno moćan frejmwork daje osnovu za izuzetno kompleksne SPA aplikacije. Angular čine komponente I angular u ovom projektu predstavlja jedinstvo svih komponenti kako bi se prikazao podatak unešen I obrađen kroz jednu od dve back-end tehnologije.



Slika 5: Izgled Angular projekta

- *TypeScript* skripterski jezik koji se kompajluje u JavaScript. U suštini je jezik koji proširuje mogućnosti samog JavaScript-a. TypeScript je jezik koji je takođe razvio Microsoft I možda je čak I jedina dobra kreacija koja je ikada izašla kao produkt Microsofta. TS je open source I održavaju ga Google, Microsoft I ljudi širom sveta.

```

src > app > igraci > deljeno-igraci > igrac.service.ts > IgracService > Deletelgrac
1  import { Injectable } from '@angular/core';
2  import { Igraci } from '../deljeno-igraci/igraci';
3  import { Http, Response, Headers, RequestOptions, RequestMethod } from '@angular/http';
4  import { Observable } from 'rxjs/Observable';
5  import 'rxjs/add/operator/map';
6  import 'rxjs/add/operator/toPromise';
7
8  @Injectable({
9    providedIn: 'root'
10 })
11 export class IgracService {
12   OdabraniIgraci : Igraci;
13   ListaIgraca : Igraci[];
14   constructor(public http : Http) { }
15
16   Postigraci(igrac : Igraci) {
17     var body = JSON.stringify(igrac);
18     var headerpotion = new Headers({ "Content-Type" : "application/json" });
19     var requestoption = new RequestOptions({ method : RequestMethod.Post, headers: headerpotion });
20     // return this.http.post('https://localhost:44309/api/igracis', body, requestoption).map(x => x.json());
21     return this.http.post('http://localhost:8080/api/igracis/napravi', body, requestoption).map(x => x.json());
22   }
23   8080
24   Putigraci(id, igrac) {
25     var body = JSON.stringify(igrac);
26     var headerpotion = new Headers({ "Content-Type" : "application/json" });
27     var requestoption = new RequestOptions({ method : RequestMethod.Put, headers: headerpotion });
28     // return this.http.put('https://localhost:44309/api/igracis/' + id, body, requestoption).map(x => x.json());
29     return this.http.put('http://localhost:8080/api/igracis/' + id, body, requestoption).map(x => x.json());
30   }
31
32   GetIgraci() {
33     // this.http.get('https://localhost:44309/api/igracis').map((data : Response) => {
34     this.http.get('http://localhost:8080/api/igracis').map((data : Response) => {
35       return data.json() as Igraci[];
36     }).toPromise().then(x => {
37       this.ListaIgraca = x;
38     })
39   }
40
41   DeleteIgrac(id : number) {
42     // return this.http.delete('https://localhost:44309/api/igracis/' + id).map(res => res.json());
43     return this.http.delete('http://localhost:8080/api/igracis/' + id).map(res => res.json());
44   }
45 }
46

```

Slika 6: Izgled servisa pisanim TypeScriptom

- *HTML* 5 kostur prikaza cele aplikacije.


```

1  <!doctype html>
2  <html lang="en">
3
4  <head>
5      <meta charset="utf-8">
6      <title>Seminarskialeksacakicsi2317frontcsharp</title>
7      <base href="/">
8
9      <meta name="viewport" content="width=device-width, initial-scale=1">
10     <link rel="icon" type="image/x-icon" href="favicon.ico">
11     <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
12         integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZ15MYYYxFfc+NcPb1dKGj7Sk" crossorigin="anonymous">
13     <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
14 </head>
15
16 <body>
17     <app-root></app-root>
18     <!-- JS, Popper.js, and jQuery -->
19     <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"
20         integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+0rCXAkrfj"
21         crossorigin="anonymous"></script>
22     <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
23         integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo"
24         crossorigin="anonymous"></script>
25     <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"
26         integrity="sha384-0gVRvuATP1z7JjHLku0U7Xw704+h835Lr+6QL9UvYjZE3Ipu6Tp75j7Bh/kR0JkI"
27         crossorigin="anonymous"></script>
28 </body>
29
30 </html>
31

```

Slika 7: Izgled index.html-a

```

1 <div class="container" style="text-align: center;">
2
3 <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
4   <a class="navbar-brand" href="#">
5     
7     Seminarski Rad
8   </a>
9   <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
10     aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
11     <span class="navbar-toggler-icon"></span>
12   </button>
13   <div class="collapse navbar-collapse" id="navbarNav">
14     <ul class="navbar-nav">
15       <li class="nav-item active">
16         <a class="nav-link" href="/">Glavna<span class="sr-only">(current)</span></a>
17       </li>
18       <li class="nav-item">
19         <a class="nav-link" routerLink="/igraci" routerLinkActive="active">Igraci</a>
20       </li>
21       <li class="nav-item">
22         <a class="nav-link" routerLink="/timovi" routerLinkActive="active">Timovi</a>
23       </li>
24     </ul>
25   </div>
26 </nav>
27 <div class="jumbotron jumbotron-fluid">
28   <div class="jumbotron">
29     <h1 class="display-4">Seminarski rad</h1>
30     <p class="lead">Tema: Upoređivanje gramatike programskog jezika C# i Java na primeru objektno-orijentisane aplikacije uz primenu baze podataka. TIP 5/a- 40 bodova</p>
31     <hr class="my-4">
32     <picture>
33       
34     </picture>
35     <p>
36       Front end i Back end aplikacija. Front end je uradjen u Angular 8. Back end je uradjen u ASP.NET. Baza podataka je MSSql koja je defaultna u C# projektima.
37       Vreme utroseno je svega par sati. Sledeca aplikacija je Java.
38     </p>
39     <a class="btn btn-primary btn-lg" href="http://www.tfzr.uns.ac.rs" role="button" target="blank_">Oficijelni Saj</a>
40   </div>
41 </div>
42
43 <!-- ===== <app-igraci></app-igraci> ===== -->
44 </div>
45
46 <router-outlet></router-outlet>

```

Slika 8: Bootstrap klase za kreiranje tabele

- *Bootstrap 4* CSS frejmwor koji je spasao sate I sate Front-End developerima. Zasniva se na SCSS-u I može dodatno dograditi SASS-om. Izuzetno je modularan I lak za korišćenje. Ime bootstrap klase se treba staviti u HTML klasu I taj tag će povući Bootstrap-ov CSS (*slika 8*).
- CSS 3 skripta po kojoj će se HTML drugačije predstavljati na stranici.

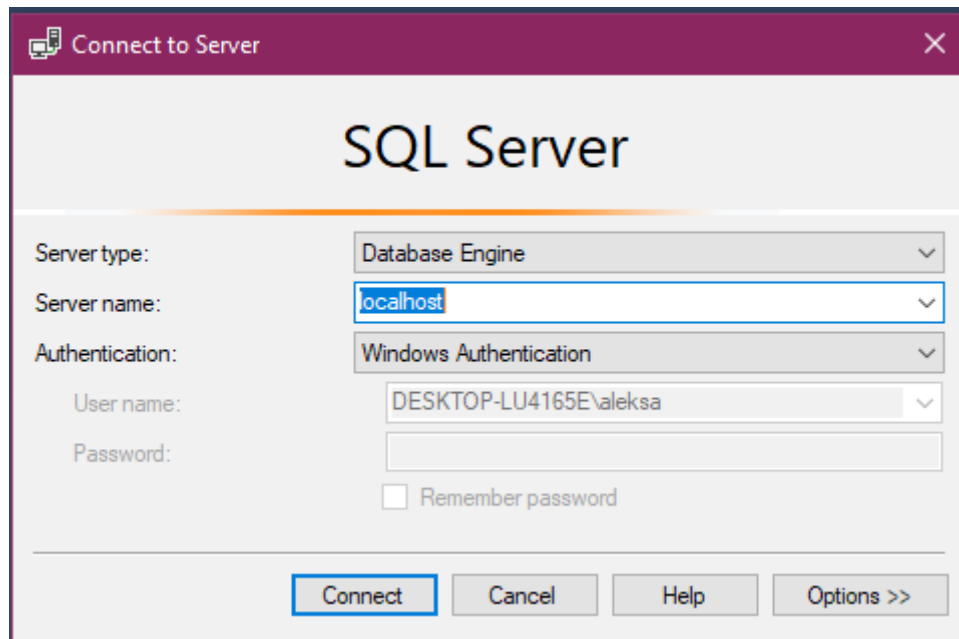
7 Opis implementacije

U sledećem odeljku će biti opisano kako se aplikacija implementirala I gradila. U slučaju replikacije, pratiti korake detaljno kako ne bi došlo do devijacije podataka, pada aplikacije ili grešaka tokom pokretanja ili rada.

7.1 C# - implementacija

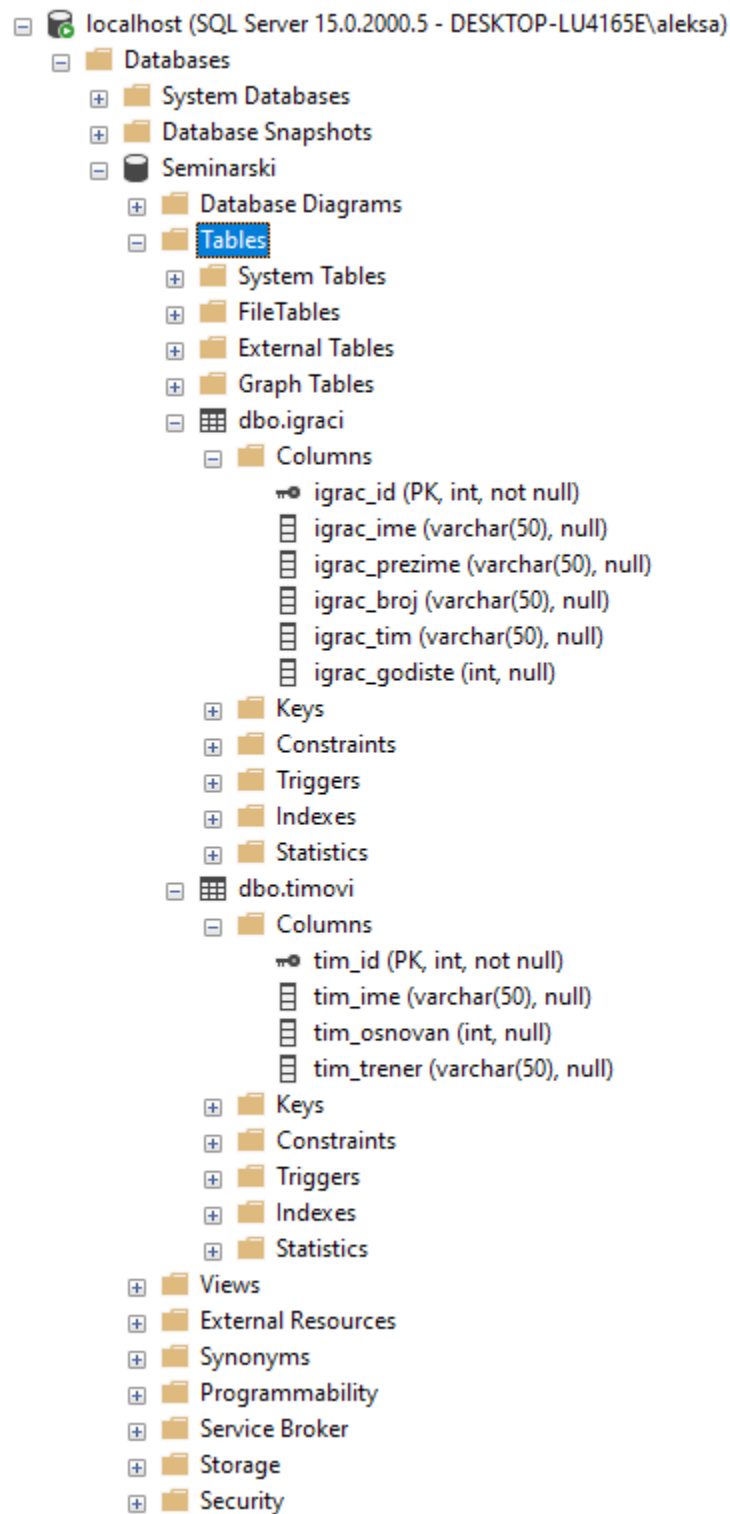
7.1.1 Baza podataka

Za ovaj deo, potrebno je imati instaliran Microsoft SQL Server na koji će se povezati sa SQL Menagement Studio-m. Za to nam treba ime računara (ili ime servera) na koji se povezuje I šifra (slika 9).



Slika 9: Login box

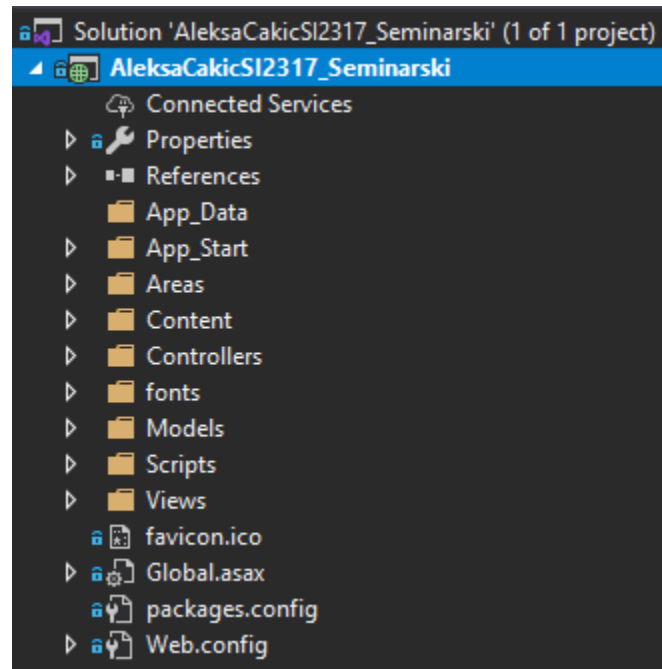
Potom, kada login uspe, sa leve strane u SMSS-u će biti formirana hijerarhije gde će se kreirati baza podataka I tabele u toj bazi podataka (slika 10).



Slika 10: Izgled baze

7.2 C# - ASP.NET implementacija

Nakon što se napravi baza podataka I poveže se sa serverom, u Visual Studiju se pravi ASP.NET aplikacija u kojoj će se VS-u I serveru reći da je veb prirode, te da treba da generiše komponente koje će se koristiti na vebu MVC Design Pattern-u (*slika 11*).



Slika 11: Arhitektura projekta u C#

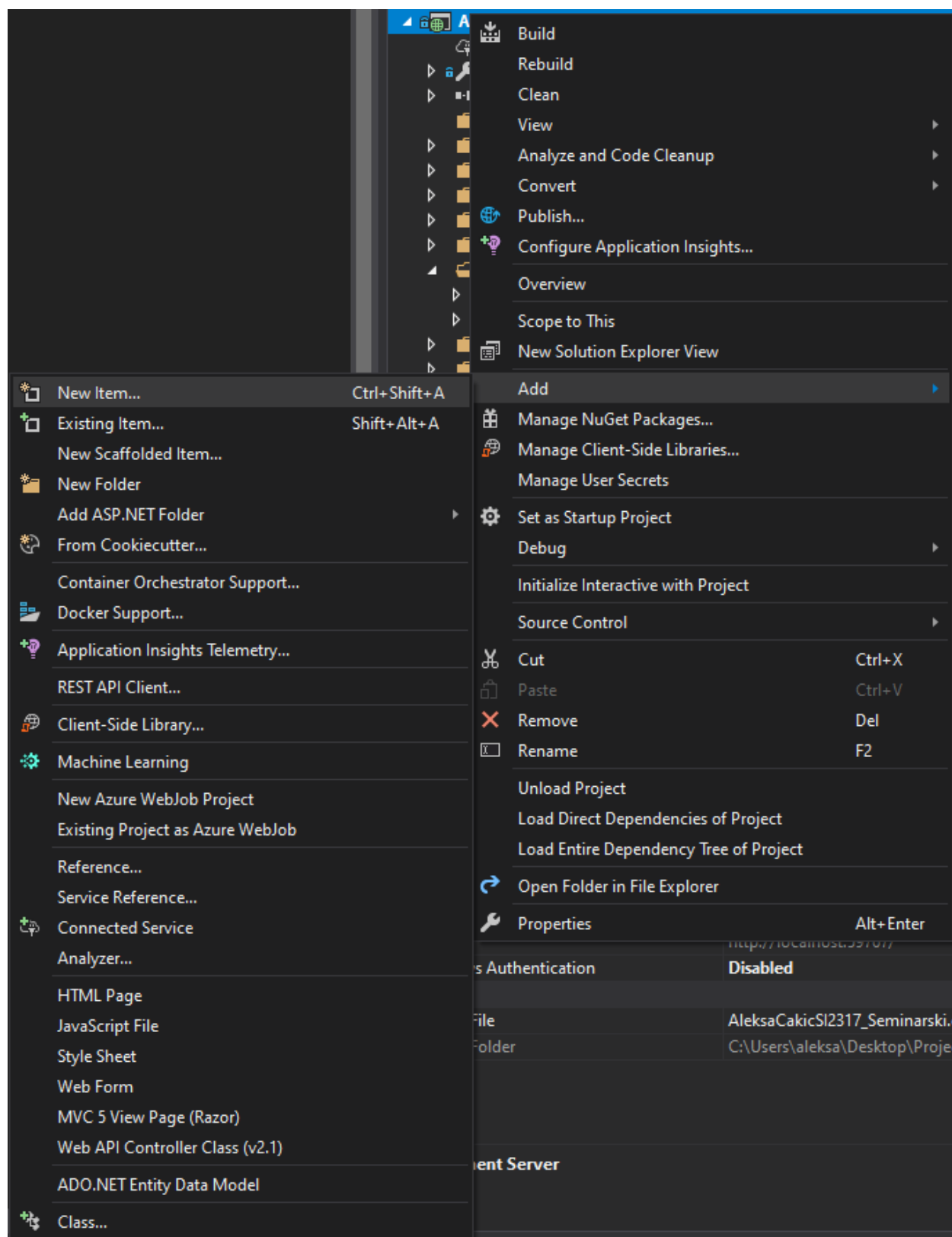
Kada je to gotovo, može se preći na veb konfiguraciju gde će se konfigurisati početni end point API-ja (*slika 12*).


```

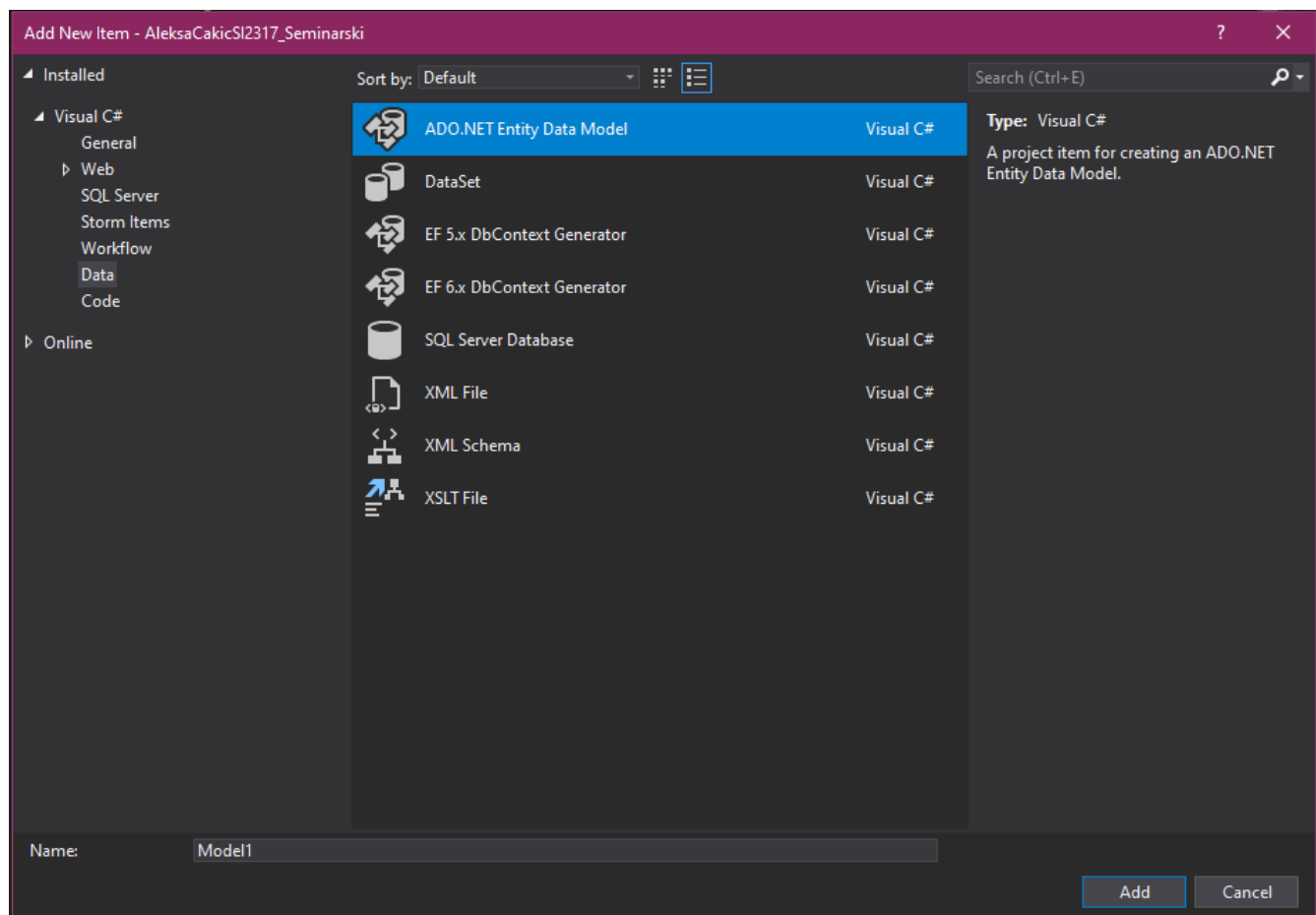
1  using System.Web.Http;
2  using System.Web.Http.Cors;
3
4  namespace AleksaCakicSI2317_Seminarski
5  {
6      1 reference
7      public static class WebApiConfig
8      {
9          1 reference
10         public static void Register(HttpConfiguration config)
11         {
12             // Web API configuration and services https://localhost:44309/
13             config.EnableCors(new EnableCorsAttribute(headers: "*", methods: "*", origins: "*"));
14
15             // Web API routes
16             config.MapHttpAttributeRoutes();
17
18             config.Routes.MapHttpRoute(
19                 name: "DefaultApi",
20                 routeTemplate: "api/{controller}/{id}",
21                 defaults: new { id = RouteParameter.Optional }
22             );
23         }
24     }

```

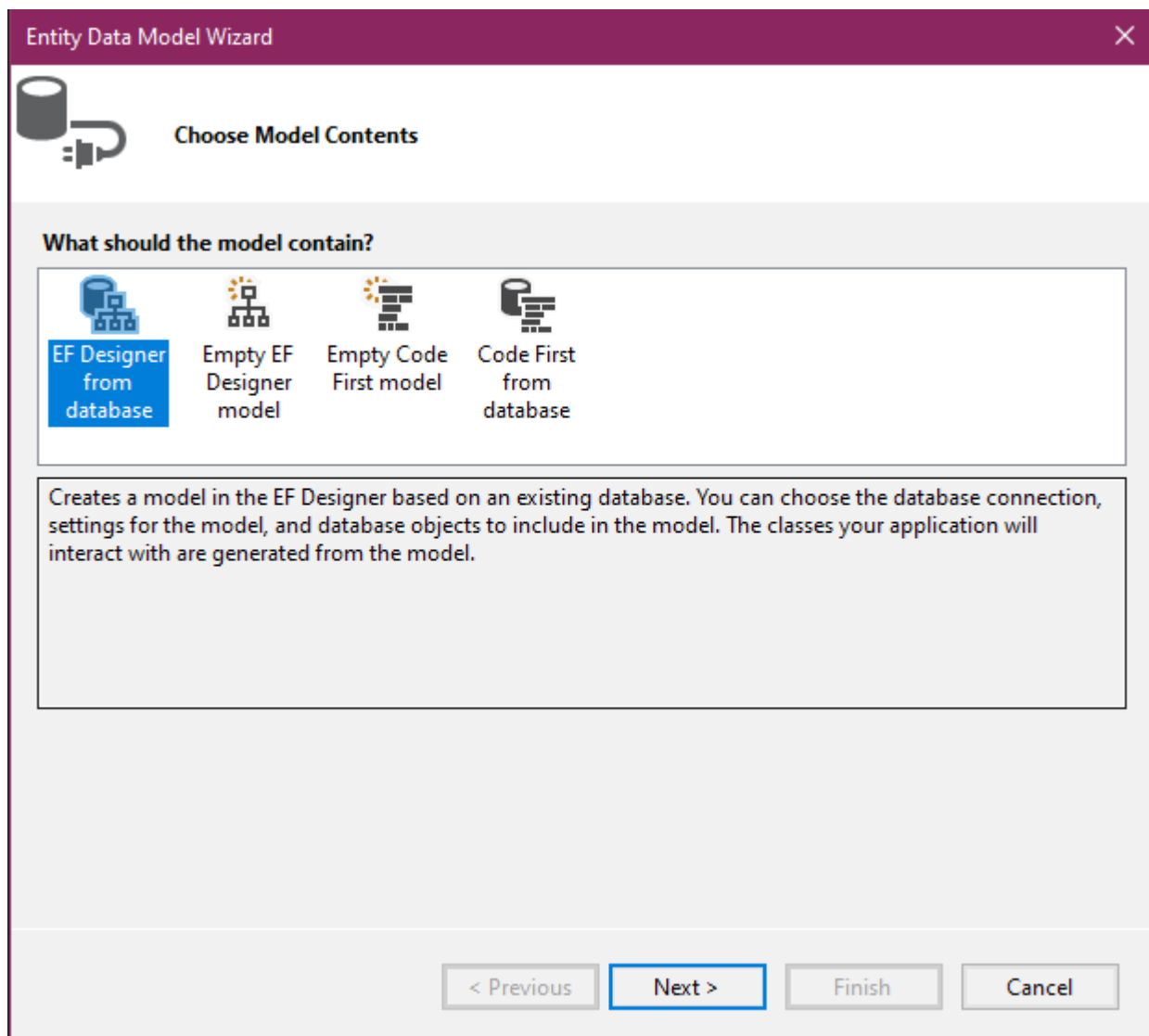
Slika 12: WebConfig fajl



Slika 13: Dodavanje Data Source-a




Slika 14: Dodavanje ADO Entity modela



Slika 15: Prvi korak uspostavljanja veze sa bazom

Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

desktop-lu4165e.Seminarski.dbo

New Connection...

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

☐ No, exclude sensitive data from the connection string. I will set it in my application code.

☐ Yes, include the sensitive data in the connection string.

Connection string:

metadata=res://*/Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;provider=System.Data.SqlClient;provider connection string="data source=localhost;initial catalog=Seminarski;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework"

☒ Save connection settings in Web.Config as:

SeminarskiEntities2

< Previous

Next >

Finish

Cancel

Slika 16: Unos tipa i mesta baze/servera

26

Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
[.NET Framework Data Provider for SQL Server] [Change...]

Server name:
[localhost] [Refresh]

Log on to the server

Authentication: [Windows Authentication]

User name: []

Password: []

☐ Save my password

Connect to a database

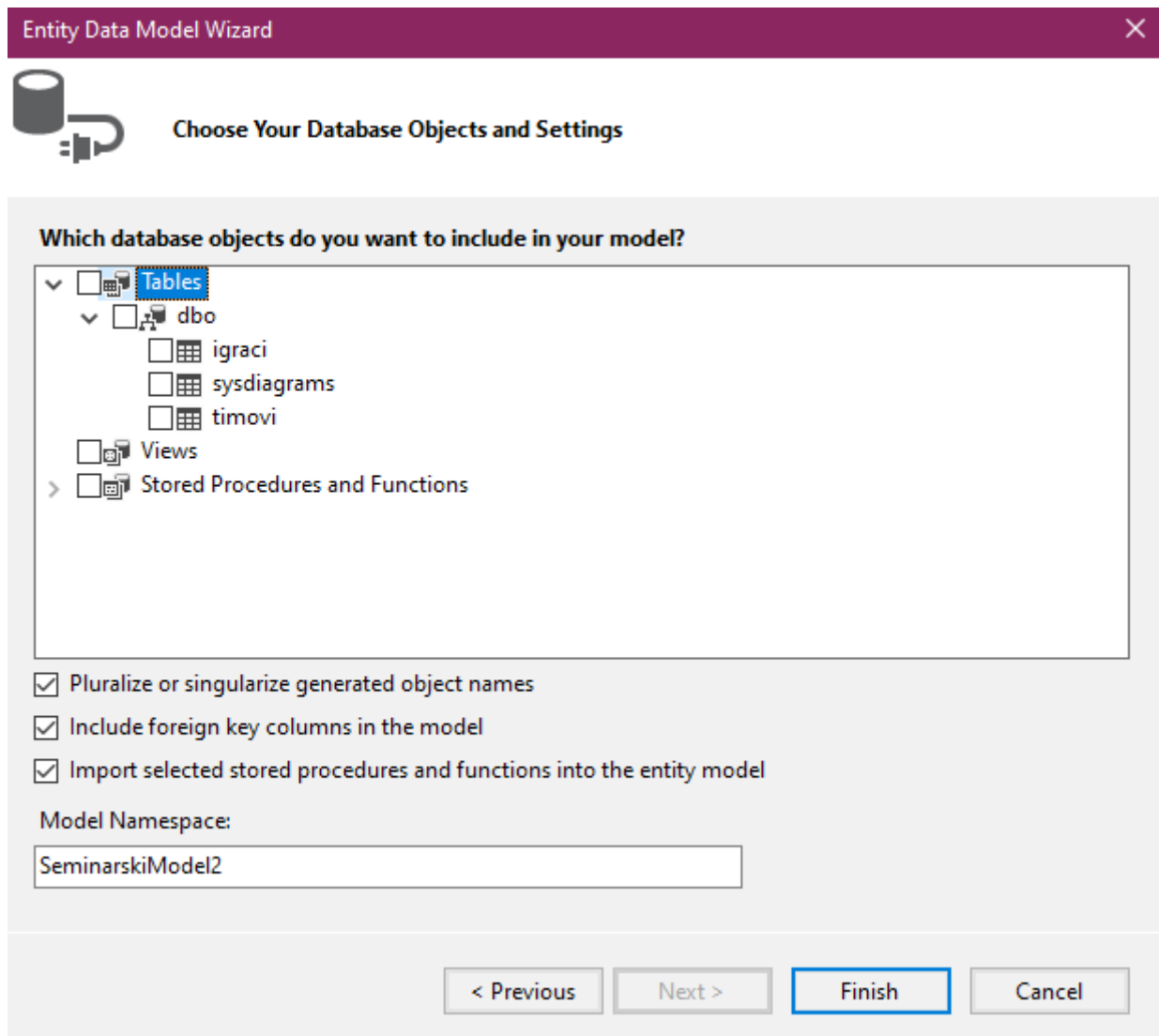
☒ Select or enter a database name:
[]

☐ Advanced...

master
model
msdb
Seminarski
tempdb

Test Connection [OK] [Cancel]

Slika 17: Odabir baze - Seminarski



Slika 18: Odabir tabela od kojih će se napraviti model

Nakon ovih komandi, u direktorijumu modela u VS Solution-u će se naći odabrani model. Modeli se mogu praviti pojedinačno ili mogu odjednom.

7.2.1 Ključni delovi koda sa objašnjenjem

Controller

Kontrolor predstavlja posrednika između prikaza podataka i modela. Sadrži glavne mehanizme za kontrolu toka programa, odnosno ponašanje same aplikacije i upravlja korisničkim zahtevima. On je programski najzahtevniji deo aplikacije.

Kontrolor interpretira ulazne podatke korisnika i prosleđuje ih do modela ili ih prikazuje korisniku. On sadrži deo aplikacijske logike i ima sposobnost da utiče na stanje modela, odnosno odlučuje kako model treba da se izmeni, kao rezultat korisničkih zahteva, kao i način na koji će se podaci prikazati.

U slučaju ove aplikacije, kontrolor uma ulogu *route*-vanja *CRUD* metoda (***POST***, ***PUT***, ***UPDATE***, ***DELETE***). U kontrolorima se u ovoj aplikaciji određuju kranji end-pointi kojima će zahtevi putovati i na koje će odgovarati istim (slika 19, 20, 21).

Uz ASP.NET-u koji je generisan, dolaze već predefinisane metode, te nema potrebe za njihovo detaljno objašnjavanje RESTful servisa.

HTTP metode

- GET –zahtev za dobijanje resursa (slika 19)
- POST –zahtev za kreiranjem/ažuriranjem resursa (slika 21)
- PUT –zahtev za kreiranjem/ažuriranjem resursa (slika 20)
- DELETE –zahtev za brisanjem resursa (slika 21)

```
// GET: api/igracis
0 references
public IQueryable<igraci> Getigracis()
{
    return db.igracis;
}
```

Slika 19: GET Metoda kontrolera - autokonfigurisana

```

// PUT: api/igracis/5
[ResponseType(typeof(void))]
0 references
public IHttpActionResult Putigraci(int id, igraci igraci)
{
    //if (!ModelState.IsValid)
    //{
    //    return BadRequest(ModelState);
    //}

    if (id != igraci.igrac_id)
    {
        return BadRequest();
    }

    db.Entry(igraci).State = EntityState.Modified;

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!igraciExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return StatusCode(HttpStatusCode.NoContent);
}

```

Slika 20: PUT metoda - autokonfigurisana

```

// POST: api/igracis
[ResponseType(typeof(igraci))]
0 references
public IActionResult Postigraci(igraci igraci)
{
    //if (!ModelState.IsValid)
    //{
    //    return BadRequest(ModelState);
    //}

    db.igracis.Add(igraci);
    db.SaveChanges();

    return CreatedAtRoute("DefaultApi", new { id = igraci.igrac_id }, igraci);
}

// DELETE: api/igracis/5
[ResponseType(typeof(igraci))]
0 references
public IActionResult Deleteigraci(int id)
{
    igraci igraci = db.igracis.Find(id);
    if (igraci == null)
    {
        return NotFound();
    }

    db.igracis.Remove(igraci);
    db.SaveChanges();

    return Ok(igraci);
}

```

Slika 21: POST i DELETE metode - autokonfigurisane

Model

Model sadrži glavne programske podatke, kao što su informacija o objektima iz baze podataka, sastoji se od skupa klasa koje modeliraju i podržavaju rešavanje problema kojim se aplikacija bavi. Obično je stabilna komponenta, koja traje koliko i sam problem. Sva poslovna logika aplikacije sadržana je u modelu. Postoji nekoliko načina konstrukcije kostura modela. Naime, programer može najpre da konstruiše model, definisanjem klasa i atributa unutar klasa, te kasnije, na osnovu klasa da formira bazu podataka.

U tu klasu modela se apstraktno predstavljaju podaci iz tabele baze kao što su ime, prezime, broj I tako dalje (slika 22).


```

1  //-----
2  // <auto-generated>
3  //   This code was generated from a template.
4  //
5  //   Manual changes to this file may cause unexpected behavior in your application.
6  //   Manual changes to this file will be overwritten if the code is regenerated.
7  // </auto-generated>
8  //-----
9
10 namespace AleksaCakicSI2317_Seminarski.Models
11 {
12     using System;
13     using System.Collections.Generic;
14
15     public partial class igraci
16     {
17         public int igrac_id { get; set; }
18         public string igrac_ime { get; set; }
19         public string igrac_prezime { get; set; }
20         public string igrac_broj { get; set; }
21         public string igrac_tim { get; set; }
22         public Nullable<int> igrac_godiste { get; set; }
23     }
24 }
25

```

Slika 22: Model Igraca

7.3 JAVA implementacija

7.3.1 Baza podataka

U primeru ove aplikacije, baza podataka I server su PostgreSQL. Postgres se u ovom primeru pokreće sa kontejner platforme koja se podiže uz pomoć jedne naredbe na jednom fajlu.

Nije potrebno imati Postgres instaliran da bi se pokrenuo server, ali je zato potreban Docker. Docker je kontejnerska platforma koja radi na nivou virtuelizacije. Nije isto što I virtuelna mašina.

U ovom primeru, baza I server se podužu tako što će se iščitati *docker-cmpose* fajl koji sadrži konfiguraciju baze podataka.

```
1  # Use postgres/example model.user/password credentials
2  # Author: Aleksa Cakic
3  version: '3.1'
4  >> services:
5  >    db:
6  >      image: postgres
7  >      restart: always
8  >      ports:
9  >          - "5432:5432"
10 >      environment:
11 >          POSTGRES_PASSWORD: root
12
```

Slika 23: yml fajl koji sadrži konfiguraciju

Na slici se vidi koji servis se podiže, kog je tipa taj servis, slika (image) kog softvera I tako dalje. Ovaj fajl određuje port, mesto I korisnike te baze.

Pokretanjem fajla uz pomoć komandne linije, inicilizira se proces (*slika 24*).

```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\aleksa\Desktop\Project\tfzrseminarski>docker-compose up
```

Slika 24: Pokretanje preko komandne linije

Kada se pokrene komanda, zid teksta procesa inicijalizacije će iskočiti. Baza je spremna za rad.

U nastavku, do kraja ovog dela uputstva će stajati uputstvo za pravljenje baze uz pomoć skripti jer će na svakom drugom računaru pri prvom pokretanju Dockera I Postgresa, biti podignut samo server bez baze podataka. Skripte je dovoljno samo executovati (*slika 25, 26, 27*).

```
1      -- Database: seminarski
2
3      -- DROP DATABASE seminarski;
4
5      CREATE DATABASE seminarski
6          WITH
7              OWNER = postgres
8              ENCODING = 'UTF8'
9              LC_COLLATE = 'en_US.utf8'
10             LC_CTYPE = 'en_US.utf8'
11             TABLESPACE = pg_default
12             CONNECTION LIMIT = -1;
13
14     COMMENT ON DATABASE seminarski
15         IS 'Baza za seminarski iz softverskih prevodioca';
```

Slika 25: Kreiranje baze

```

1  create table igrac
2  (
3      igrac_id      integer not null
4      |      constraint igrac_pkey
5      |      primary key,
6      igrac_broj    integer,
7      igrac_godiste integer,
8      igrac_ime      varchar(255),
9      igrac_prezime  varchar(255),
10     igrac_tim      varchar(255)
11 );
12
13 alter table igrac
14     owner to postgres;

```

Slika 26: Kreiranje igrača SQL skripta

```

1  create table tim
2  (
3      tim_id        integer not null
4      |      constraint tim_pkey
5      |      primary key,
6      tim_ime        varchar(50),
7      tim_osnovan    integer,
8      tim_trener     varchar(50)
9  );
10
11 alter table tim
12     owner to postgres;
13
14

```

Slika 27: SQL skripta za kreiranje tima

7.3.2 Implementacija Java back-enda – Spring

Veb sajt Springa (<https://projects.spring.io/spring-framework/>) definiše radni okvir Spring ovako: radni okvir Spring obezbeđuje sveobuhvatni model programiranja i konfiguracije za moderne poslovne aplikacije zasnovane na jeziku Java. Radni okvir Spring se koristi za spajanje poslovnih Java aplikacija. Njegova glavna svrha je da vodi računa o svim tehničkim vezama koje su potrebne da bi se povezali različiti delovi aplikacije. To omogućava programerima da se fokusiraju na srž svog posla – na pisanje poslovne logike.

Gradle

Gradle je softver za automatizovano bildovanje i podizanje aplikacija za više jezika. Kontrolise proces kompilacije, pakovanja i testiranja, deploymenta i puštanja. Koristi jezik zvani Groovy.

U primeru ove aplikacije, gradle je služio da ubacuje zavisnosti kao što su spring frejmwork, i dodatni alati (*slika 28*) potrebni za razvoj aplikacije.

```

1  plugins {
2      id 'org.springframework.boot' version '2.3.1.RELEASE'
3      id 'io.spring.dependency-management' version '1.0.9.RELEASE'
4      id 'java'
5  }
6
7  group = 'rs.tfzr.aleksa'
8  version = '0.0.1-SNAPSHOT'
9  sourceCompatibility = '1.8'
10
11  repositories {
12      mavenCentral()
13  }
14
15  ► dependencies {
16      implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
17      implementation 'org.springframework.boot:spring-boot-starter-web'
18      developmentOnly 'org.springframework.boot:spring-boot-devtools'
19      runtimeOnly 'org.postgresql:postgresql'
20      testImplementation('org.springframework.boot:spring-boot-starter-test') {
21          exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
22      }
23
24      // https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa
25      compile group: 'org.springframework.boot', name: 'spring-boot-starter-data-jpa', version: '2.3.1.RELEASE'
26      // https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa
27      compile group: 'org.springframework.data', name: 'spring-data-jpa', version: '2.3.1.RELEASE'
28
29  }
30
31  ► test {
32      useJUnitPlatform()
33  }
34

```

Slika 28: Izgled gradle.build fajla

U scope-u dependancies se nalaze implementacije, a compile je šta će ući u Ggradle-ov kompajl.

Spring Configuration

Spring je možda dovoljno moćan da radi *right-out-of-the-box*, ali naravno, za stabilnost I dodatne mogućnosti, Spring konfiguracioni fajl je potreban kako rekao Spring endžinu kako da radi I sa kime ima posla (slika 29). Vidi se odakle će spring vući bazu ***spring.datasource.url***, koji su korisničko ime pasvord baze podataka.

```
1  spring.datasource.url=jdbc:postgresql://localhost/seminarski
2  spring.datasource.username=postgres
3  spring.datasource.password=root
4  spring.jpa.generate-ddl=true
```

Slika 29: Konfiguracija Springa

Java

Java deo projekta prati Repository pattern, što znači da ima Data Object layer preko kojeg će koristiti servis. Sama postavka projekta je učinjena kako bi se svaka osoba uključena u projekat mogla snaći u arhitekturi (slika 30).


```

48     @GetMapping("/igracis")
49     public List<Igrac> getSviIgraci() {
50         System.out.println("Uzimam sve igrace");
51
52         List<Igrac> igraci = new ArrayList<>();
53         repository.findAll().forEach(igraci::add);
54
55         return igraci;
56     }
57
58     @PostMapping(value = "/igracis/napravi")
59     public Igrac postIgrac(@RequestBody Igrac igrac) {
60         Igrac _igrac =
61             repository
62                 .save(new Igrac(
63                     igrac.getIgrac_id(), igrac.getIgrac_ime(), igrac.getIgrac_prezime(), igrac.getIgrac_broj(), igrac.getIgrac_tim(), igrac.getIgrac_godiste()
64                 ));
65
66         return _igrac;
67     }
68
69     @DeleteMapping("/igracis/{id}")
70     public ResponseEntity<String> obrisiIgraca(@PathVariable("id") int igrac_id) {
71         System.out.println("Briše se igrač sa id: " + igrac_id + "... ");
72
73         repository.deleteById(igrac_id);
74
75         return new ResponseEntity<>("Igrac je uspešno obrisani!", HttpStatus.OK);
76     }

```

Slika 31: Kontroler igrača

Kroz Springove anotacije se može postaviti putanja (end-point) na koji će front end moći da peca funkcije preko HTTP metoda. Ovde se na slici 31 se vidi tipičan Spring kod koji bi završio u bilo kojoj aplikaciji. GET metoda će sve entrije iz baze spakovati u objekat I vratiti front-endu za prikaz.

POST metoda preko repozitorija ubacuje novi kveri u bazu I zbog toga se poziva metoda *.save()* kako bi se kveri uspešno izvršio. DELETE metoda je intuitivna: odabrani ID igrača će obrisani iz baze. Ista logika je upotrebljena I za timove.

Model

```
1      package rs.tfzr.aleksa.tfzrseminarski.model;
2
3      import org.springframework.beans.factory.annotation.Autowired;
4
5      import javax.persistence.*;
6      import java.io.Serializable;
7
8      @Entity
9      @Table(name = "igrac")
10     public class Igrac implements Serializable {
11
12         @Id
13         @Column(name = "igrac_id")
14         @GeneratedValue(strategy = GenerationType.AUTO)
15         private int igrac_id;
16
17         @Column(name = "igrac_ime")
18         private String igrac_ime;
19
20         @Column(name = "igrac_prezime")
21         private String igrac_prezime;
22
23         @Column(name = "igrac_broj")
24         private int igrac_broj;
25
26         @Column(name = "igrac_tim")
27         private String igrac_tim;
28
29         @Column(name = "igrac_godiste")
30         private int igrac_godiste;
```

Slika 32: Model Igrača

Model u javi ima mnogo veće značenje. U tom modelu se uz pomoć Spring anotacija mapira polje sa mesto u tabli. Time spring zna kako da istruktuje repository gde da raspodeli resurse (podatke). Povezivanjem podatka anotacijom **@Column** možemo proslediti mesto baze polju u modelu. **@Id** bi trebao odmah naznačiti da je zadužen za bilo koji tip identifikacije – od intedžera do UUID-a, a reč

“bilo koji” je namerno iskorištena kako bi se objasnio *@GeneratedValue*. Naime, argument koji ta anotacija prima “bandluje” tip podatka koji mu se prosleđuje.

8 Greške

Greške su sastavni deo života, kako od stila, tako i do IT-a. U programiranju možemo da klasifikujemo greške u 3 kategorije:

1. Run Time
2. Logičke
3. Sintaksičke

I u daljem tekstu će biti kroz primere objašnjeno kako se ponašaju u ovim dvema backend tehnologijama.

8.1 Run time greške

Greške koje nastaju za vreme rada programa (runtime errors) – jesu greške koje kompajleri/interpreteri ne mogu da uoče ali se one ipak ispoljavaju u toku rada tj. korišćenja programa. Ove greške, ukoliko se ne „uhvate“ posebnim programskim strukturama u kodu (najčešće try-catch metoda), mogu da izazovu prestanak rada programa uz gubitak svih podataka i međurezultata neke obrade koji su postojali do momenta pojave greške. Tipični primeri ovakvih grešaka u kodu se pojavljuju kada se jednoj promenljivoj numeričkog tipa pokuša da dodeli vrednost tekstualnog tipa koja ne može da podleže konvertovanju u broj (osim u slučaju ako sam tekst nije i broj). Tu je zatim tipična greška deljenja nekog broja sa nulom, pokušaj povezivanja na server baze podataka kada su parametri konekcije loše zadati ili kada je server neaktivan i sl. Runtime greške, ukoliko postoji dobra praksa programiranja, mogu da budu uhvaćene, ubeležene (log sistem) i preduhitrene na način koji neće obustaviti dalji rad programa.

8.1.1 Run time greške u C#

```
// PUT: api/timovis/5
[ResponseType(typeof(void))]
public IHttpActionResult Puttimovi(int id, timovi timovi)
{
    //if (!ModelState.IsValid)
    //{
    //    return BadRequest(ModelState);
    //}

    if (id != timovi.tim_id)
    {
        var a = id / 0;
        return BadRequest();
    }

    db.Entry(timovi).State = EntityState.Modified;

    try
    {
        db.SaveChanges();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!timoviExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return StatusCode(HttpStatusCode.NoContent);
}
```

Slika 33: Run time greška

Run time greška u ovom kodu se nalazi u prvoj nezakomentarisanom if upitu gde se promenljiva `a` deli sa nulom, time zbunjuje program i pravi Run time grešku.

8.1.2 Run time greške u Javi

```
package rs.tfzr.aleksa.tfzrseminarski;

public class Greske {
    public static int promenljiva1 = 10;
    public static int getPromenljiva2 = promenljiva1/0;

    public int printuj() {
        System.out.print(getPromenljiva2);
        return getPromenljiva2;
    }
}
```

Slika 34: Run time greška u Javi

Ista situacija kao uslici 33 (slika 33), samo što je na ovom primeru malo lepše predstavljeno jer je pisano u javi. Naime, u primeru se vide dve nepromenljive *promenljiva1* I *getPromenljiva2*. Vrenost nepromenljive *getPromenljiva2* je vrednost promenljive 1 podeljeno sa 0. Naravno, postaramo se da ta greška pozove I ispiše, samo što neće ispisati rezultat već zid teksta koji će usmeravati na gomilu javinih funkcija koje su bile afektivne u procesu.

8.2 Logičke

Aplikacija je uspešno kompajlirana, prilikom rada ne prijavljuje nikakve greške niti prestaje sa radom. Sve je u redu, ali vaš program “samo” ne radi ono što treba da radi. Vraća pogrešne rezultate, ne upisuje podatke i tome slično. Ovo su verovatno greške koje je najteže otkriti jer su posledice lošeg dizajna aplikacije.

8.2.1 Logičke greške u C#

Logička greška predstavljena na slici 35 se dešava u else if bloku. Naime, do greške dovodi nekompatibilnost tipova liste timova I kontrolera, koja nije instancirana I nije istog tipa. Kompajler može da negoduje u toku greške, a može u potpunosti da je ignoriše. Recimo da su oba tipa podatka ista, da je kompajler odradio svoje I uneo taj boolean podatak, umesto očekivanog, rezultat je pogrešan, ali je program I dalje radio I kompajlovao se.

```
// TODO - Ovde se nalazi LOGICKA GRESKA
// GET: api/timovis/5
[ResponseType(typeof(timovi))]
public IHttpActionResult Gettimovi(int id)
{
    timovi timovi = db.timovis.Find(id);
    if (timovi == null)
    {
        return NotFound();
    }
    else if (timovi == Controllers.igracisController)
    {
        Console.WriteLine("Logička greška ovde!");
    }

    return Ok(timovi);
}
```

Slika 35: Primer logičke greške

8.2.2 Logičke greške u Javi

```
int aleksa = 5 + 4 * 3 / 2;
int aleksa2 = (5 + 4) * (3 / 2);
int aleksa3 = (5 + 4) * (3 / 2);
int aleksa4 = (5 + (4 * 3) / 2);

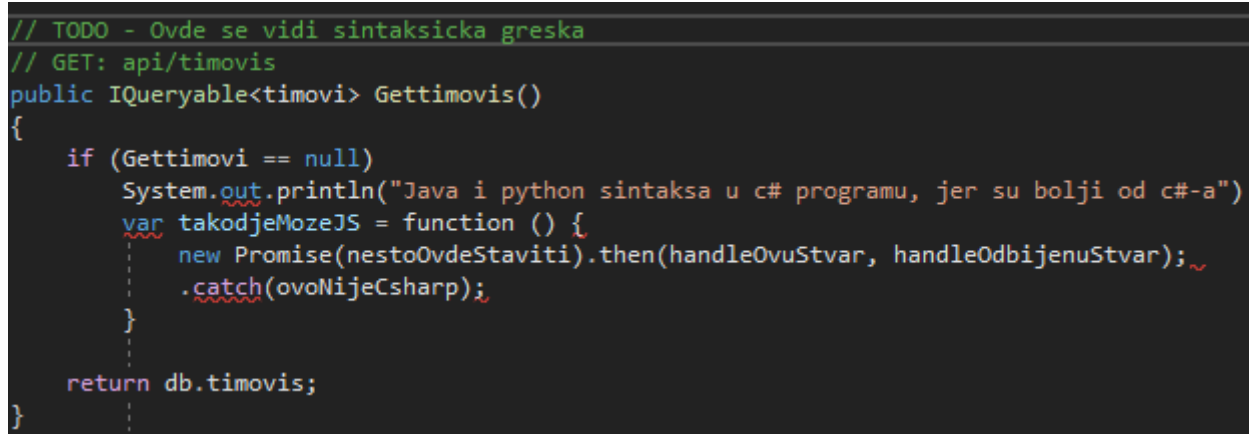
public void logickaGreksa() {
    System.out.println("Aleksa: " + aleksa + " // ");
    System.out.println("Aleksa2: " + aleksa2 + " // ");
    System.out.println("Aleksa3: " + aleksa3 + " // ");
    System.out.println("Aleksa4: " + aleksa4 + " // ");
}
```

Slika 36: Primer logičke greške u javi

U ovom primeru (slika 36) možemo videti nekoliko promenljivih tipa intidžera gde će svaki od njih dati različit rezultat, osim druge I treće. Java zna redoslede računskih operacija, ali nije to uvek slučaj za programera. Pre kodiranja neke metode ili matematičke operacije, programer mora dobro ovratiti pažnju kako piše kod. U prvoj promenljivoj vidimo rešenlje 11, drugoj I trećoj 13.5, a u poslednjoj takođe 11.

8.3 Sintaksičke

8.3.1 Sintaksičke greške u C#

A screenshot of a code editor showing C# code with several syntax errors. The code is for a GET endpoint 'api/timovis'. It starts with a comment '// TODO - Ovde se vidi sintaksicka greska' and another '// GET: api/timovis'. The method signature is 'public IQueryable<timovi> Gettimovis()'. Inside, there's an 'if (Gettimovi == null)' block. The first error is 'System.out.println' which is Java syntax, not C#. The second error is 'var takodjeMozeJS = function () {' which is JavaScript syntax. The third error is 'new Promise(nestoOvdeStaviti).then(handleOvuStvar, handleOdbijenuStvar);' which is also JavaScript syntax. The fourth error is '.catch(ovoNijeCsharp);' which is JavaScript syntax. The code ends with 'return db.timovis;' and a closing brace. The errors are highlighted with red squiggly lines.

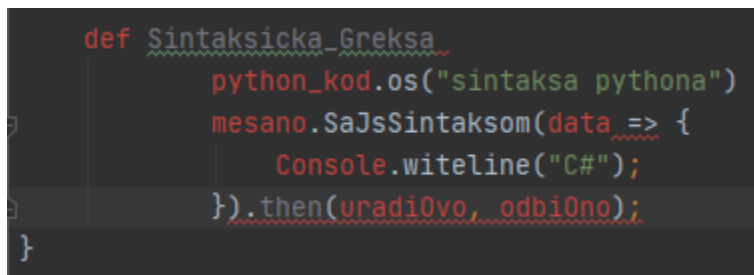
```
// TODO - Ovde se vidi sintaksicka greska
// GET: api/timovis
public IQueryable<timovi> Gettimovis()
{
    if (Gettimovi == null)
        System.out.println("Java i python sintaksa u c# programu, jer su bolji od c#-a")
        var takodjeMozeJS = function () {
            new Promise(nestoOvdeStaviti).then(handleOvuStvar, handleOdbijenuStvar);
            .catch(ovoNijeCsharp);
        }

    return db.timovis;
}
```

Slika 37: Primer sintaksičke greške

Primeri sintaksičkih grešaka variraju. Na primeru u slici 37 vidi se nepoznata sintaksa, amalgamacija `System.out.println()` java, i prosleđivanje funkcije kao rezultat promenljivoj.

8.3.2 Sintaksičke greške u Javi

A screenshot of a code editor showing Java code with syntax errors. The code starts with 'def Sintaksicka_Greksa' which is Python syntax. It then has 'python_kod.os("sintaksa pythona")' which is also Python syntax. The next line is 'mesano.SaJsSintaksom(data => {' which is a mix of Java and JavaScript. The following line is 'Console.witeline("C#");' which is Java syntax. The code ends with '}).then(uradiOvo, odbiOvo);' which is JavaScript syntax. The errors are highlighted with red squiggly lines.

```
def Sintaksicka_Greksa
{
    python_kod.os("sintaksa pythona")
    mesano.SaJsSintaksom(data => {
        Console.witeline("C#");
    }).then(uradiOvo, odbiOvo);
}
```

Slika 38: Primer sintaksičke greške u Javi

Umesto klasičnih grešaka kao što je gutanje slova ili izostavljanje znaka “ ; ”, ovde se takođe može videti nenativna sintaksa Javi. Možemo videti deklaraciju metode u stily Python programskog jezika, pokušaj pristupanja otvorenim `os` lib funkcija, gde se linija ispod pretvara u amalgamaciju C# koji izvršava *promise* funkciju koja je nativno u JavaScript jeziku.

8.4 Izgled poruka kada kompajler naleti na grešku

8.4.1 JAVA

```
12 errors

FAILURE: Build failed with an exception.

* What went wrong:
Execution failed for task ':compileJava'.
> Compilation failed; see the compiler error output for details.

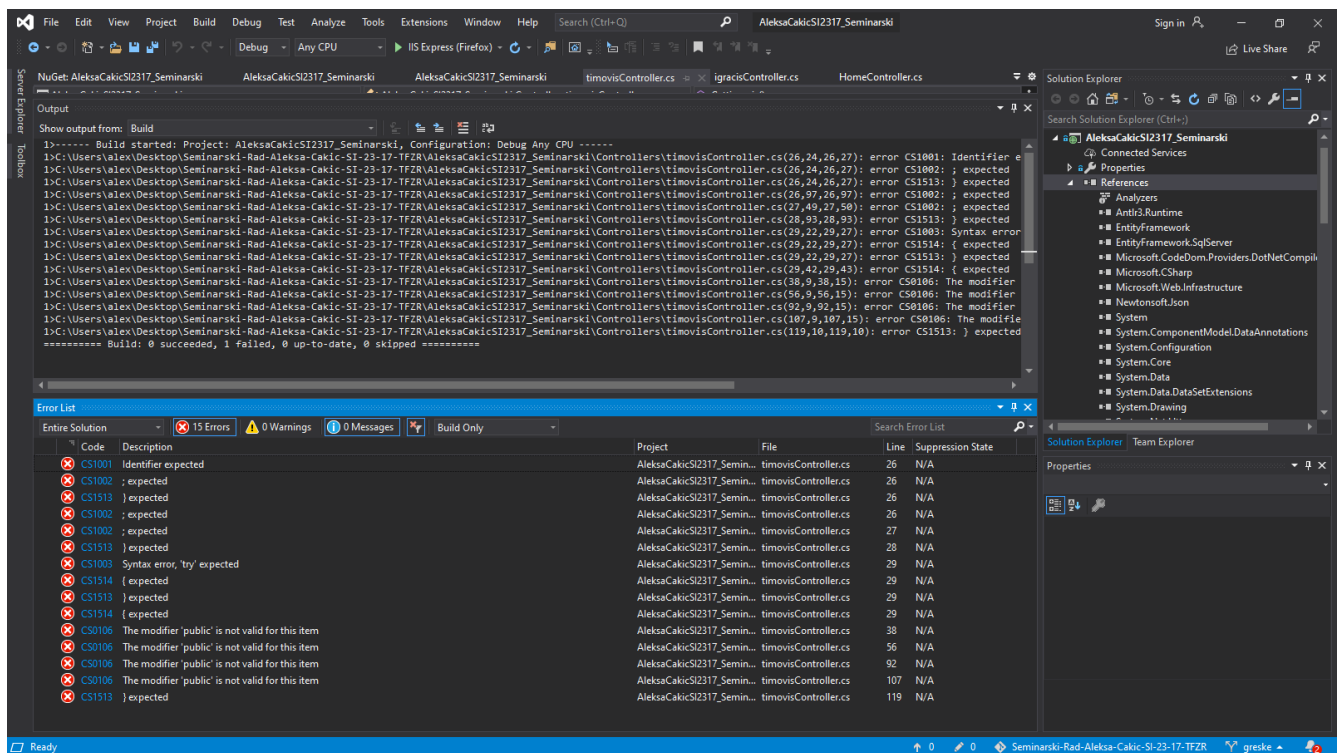
* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug option to get more log output. Run with --scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in 1s
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.4.1/userguide/command\_line\_interface.html#sec:command\_line\_warnings
1 actionable task: 1 executed
3:44:07 AM: Task execution finished ':classes'.
```

Slika 39: Neuspešan kompajl

8.4.2 C#



Slika 40: Neuspešan kompajl

9 Pravila I BNF/EBNF prikaz

Za opis programskih jezika često se koriste kontekstno-slobodne gramatike. Bekus-Naurova forma (BNF) je konvencija za zapisivanje pravila kontekstno-slobodnih jezika. Proširena Bekus-Naurova forma (EBNF) dodaje određene sintaksičke izraze BNF notaciji i omogućava jednostavniji zapis pravila gramatike.

9.1 Run Time greške

9.1.1 Java

Napravljena greška u EBNF kodu:

```
<int> <identifikator1> ::= cifra{cifra}<separator>  
<int> <identifikator1> ::= <identifikator1> / {0}<separator>  
<cifra> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

Kako bi se izbegla runtime greška, nije dozvoljeno deklarirati promenljivu tipa int i dodeliti joj vrednost : broj podeljen nulom, kao što je prikazano na slici broj 34.

9.1.2 C#

Napravljena greška u EBNF kodu

```
<var> <identifikator> ::= cifra{cifra} / 0<separator>  
<cifra> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

Isti slučaj kao i u prethodnom primeru, samo što se tip promenljive deklarira kao var što jeziku C# daje veliku fleksibilnost i dinamiku u odnosu na Javu. Time se stopa grešaka može povećati. Ali slučaj predstavljen kao bilo koja promenljiva ne bi mogla da se podeli sa brojem 0 kao što se vidi na slici 33.

9.2 Leksičke greške

9.2.1 Java (slika 38)

Napravljena greška u EBNF kodu

```
<def> <Sintaksička_greška> ::= <telo metode><separator>
```

Pravilo:

U javi se metoda ne deklarise na dati način, javi treba prvo dati do znanja koji je modifikator pristupa u pitanju, povratni tip, zatim ime praćeno zagradama.

**<public | protected | private | static | abstract | final | synchronized | native><void |
><SintaksičkaGreška><parametri> ::= <telo metode>**

9.2.2 C# (slika 39)

Napravljena greška u EBNF kodu

Greška koje se ovde desila je jezička kojoj je cilj bio da izazove pometnju. Naime, JavaScript sintaksa nije podržana u C#-u, te pravilan EBNF ove funkcije bi bio:

**<public | protected | private | static | abstract | final><void |
><SintaksičkaGreška><parametri> ::= <**

<nativna print naredba><separator>

**<var><ime promenljive> ::= <(impliciranje
tipa)><naziv_metode>([identifikator])<separator>**

Mora se uzeti u obzir povratni tip metode. Ako metoda nije tipa void, neće bit dalje greške.

9.3 Semantičke greške

9.3.1 Java

Greška

<def><ime metode> “:” ::=<telo metode>

Pravilo

Java jezik u deklarisanju svojih metoda mora da ima kvrgaste zagrade, ne dve tačke.

**<public | protected | private | static | abstract | final | synchronized | native><void |
><ImeMetode><parametri> ::= “{” <telo metode> “}”**

9.3.2 C#

Greška

<var><identifikator>::=<function> <telo funkcije>...

Pravilo

Ako se metoda proseduje kao vrednost nekoj promenljivoj, ta metoda mora imati povratni tip I mora biti ravno pozvana. Metoda mora imati dve zagrade I separator na kraju.

<var><identifikator>::=<metoda sa odgovarajućim povratnim tipom>” (); ”

10 Zaključak

Ovaj seminarski rad je bio zanimljiv projekat za nekoga ko je više radio Javu. Oba jezika su sintaksički izuzetno slična, neki bi rekli skoro isti. Naravno, svaki jezik ima zanimljive native funkcije kao serializable fields u C#-u. Oba jezika su dobra za nešto. C# možda ima prednost je ga održava majkrosoft kome je cilj dominacija svetom.

Gledajući kroz frejmvorke korištene, vidi se kako se oba jezika ponašaju MVC arhitekturi I koji jezik kako komunicira sa bazom. Esencijalno, sami jezici bi imalo skoro isti kod kad bi bili bare-bone, ali kroz kombinaciju ASP.NET I Spring frejmvorka, nastala je mala paralela.

C# I ASP.NET su imali najlakšu konfiguraciju baze dok je konfiguracija baze u Springu išla kroz kod. Ovde se vidi razlika: jedan jezik ima brzo pravljenje baze, sigurno za male projekte koji bi mogli puno da skladište uz pomoć SQL Servera, dok sa druge strane imamo Spring I Postgres koji nude moć konfigurisanja I detaljnog upravljanja. Svakako, oba jezika imaju podršku za obe vrste servera I baze, C# kroz Nugget pakete I Java kroz Spring, Spark, Gradle I Maven.

U oba slučaja jezika vidimo sličnost sintakse, ali različite ličnosti.

C# će I dalje ostati dete majkrosofta I biće jedino kompatibilan sa windows platformom jer koristi biblioteke istog operativnog sistema, dok se javina moć samo više naglašava, neprikosnovenost u servisima I mogućnosti da nauči programera kako funkcionišu serveri.

Iako je ovaj seminarski rad veoma polarizovan, profesionalno neprofesionalno urađen, autor je hteo da izrazi svoje nezadovoljstvo određenom tehnologijom održene kompanije trudeći se da udovolji temi I zadatku seminarskog rada.

11 Izvorni kod aplikacije

Izvorni kod aplikacije se nalazi na sledećem linku na GitHub repozitorijumu. Korišćenje koda je slobodno. Kod treba klonirati, obrisati skriveni .git fajl I prepravljati po volji.

U README.md fajlu piše kako konfigurisati front end I pokretanje istog.

12 Literatura

1. <https://cs.au.dk/~amoeller/RegAut/JavaBNF.html>
2. <http://www.externsoft.ch/download/csharp.html>
3. *Spring in Action, 5th Edition* – Craig Walls
4. *Cloud Native Java: Designing Resilient Systems with Spring Boot, Spring Cloud, and Cloud Foundry* – Josh Long & Keny Bastani
5. *ASP.NET Core MVC 2. 2017* - Adam Freeman