

Web aplikacija za ostvarivanje suradnje samostalnih programera na projektima

Krpan, Vatroslav

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Virovitica University of Applied Sciences / Veleučilište u Virovitici**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:165:644760>

Rights / Prava: [In copyright](#)/[Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2025-02-22**

Repository / Repozitorij:



[Virovitica University of Applied Sciences Repository - Virovitica University of Applied Sciences Academic Repository](#)



VELEUČILIŠTE U VIROVITICI
Stručni prijediplomski studij Računarstva

VATROSLAV KRPAN

WEB APLIKACIJA ZA OSTVARIVANJE SURADNJE SAMOSTALNIH
PROGRAMERA NA PROJEKTIMA
ZAVRŠNI RAD

VIROVITICA, 2023

VELEUČILIŠTE U VIROVITICI
Stručni prijediplomski studij Računarstva

WEB APLIKACIJA ZA OSTVARIVANJE SURADNJE SAMOSTALNIH
PROGRAMERA NA PROJEKTIMA
ZAVRŠNI RAD

Predmet: Projektiranje informacijskih sustava

Mentor:

Marko Hajba, mag. math., pred.

Student:

Vatroslav Krpan

VIROVITICA, 2023.



OBRAZAC 1b

ZADATAK ZAVRŠNOG RADA

Student/ica: VATROSLAV KRPAN JMBAG: 0165081903

Imenovani mentor: Marko Hajba, mag. math., pred.

Imenovani komentor: -

Naslov rada:

Web aplikacija za ostvarivanje suradnje samostalnih programera na projektima

Puni tekst zadatka završnog rada:

Izraditi web aplikaciju koja služi samostalnim programerima za pronalazak suradnika za rad na projektima. Za izradu aplikacije potrebno je koristiti .NET 7, Blazor Server programski okvir i Entity Framework Core 7, a za rad s bazom podataka SQL Server. Bazu je potrebno napuniti testnim podacima.

Korisnici mogu biti registrirani korisnici (programeri) i administratori. Web aplikacija treba imati različite funkcionalnosti:

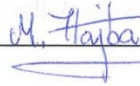
- Omogućiti registraciju i prijavu korisnika.
- Administratori brinu o održavanju funkcionalnosti aplikacije, primaju prijedloge i rješavaju probleme koje mogu prijaviti ostali korisnici.
- Registrirani korisnici na početnoj stranici imaju prikaz projekata predloženih po vještinama koje je korisnik odabrao prilikom registracije. Mogu pokrenuti novi projekt unosom nužnih informacija o njemu i vršiti administraciju projekata kojima su vlasnici.
- Registrirani korisnici mogu pretraživati projekte i prijavljivati se za suradnju na njima, ako je to moguće. Vlasnik projekta odlučuje o prijavama na njegove projekte, a komunikacija se vrši putem sustava za razmjenu poruka integriranom u aplikaciju.
- Korisnici međusobno mogu ocjenjivati suradnike na projektu, što je vidljivo na njihovom profilu. Na profilu korisnika vidljiva je i druga statistika njegova rada, npr. ukupan broj projekata na kojima je sudjelovao, broj završenih projekata, uspješnost i sl.

Osim programskog rješenja, u pisanom dijelu završnog rada opišite ukratko korištene tehnologije te detaljno opišite arhitekturu sustava i odabrane procese i/ili funkcije unutar same aplikacije. Prilikom opisivanja, osim neformalnih koristite i neke od formalnih metoda koje poznajete.

Datum uručenja zadatka studentu/ici: 31.07.2023.
Rok za predaju gotovog rada: 08.09.2023.

Mentor:

Marko Hajba, mag. math., pred.



Dostaviti:

1. Studentu/ici
2. Povjerenstvu za završni rad - tajniku

Web aplikacija za ostvarivanje suradnje samostalnih programera na projektima**Sažetak**

Tema rada je web aplikacija koja omogućava samostalnim programerima umreženje i povezivanje s drugim kolegama koji se bave kreiranjem programskih rješenja u raznim programskim jezicima i programskim okvirima. Aplikacija je osmišljena kako bi pomogla samostalnim programerima prilikom pronalaska posla te im omogućila ostvarivanje suradnje s drugim programerima na otvorenim projektima. Prilikom izrade aplikacije korišten je Blazor programski okvir koji omogućava brz i fleksibilan razvoj aplikacije. Glavna svrha ove aplikacije je povezati korisnika sa sadržajem (projektima) koji bi po određenim parametrima odgovarao upravo njemu. Prilikom stvaranja korisničkog računa programer ima mogućnost unošenja svojih vještina te sam sebi za pojedinu vještinu dodjeljuje ocjenu (počevši od 1- posjeduje temeljno znanje vještine do 5-odlično poznavanje vještine). Korisnik ima mogućnost stvaranja svog projekta te na njemu uz druge informacije može navesti očekivane vještine i stupanj znanja od suradnika koji bi se prijavili na projekt. Aplikacija zatim korisniku nudi mogućnost pretrage projekata po nekoliko različitih kriterija, pregled informacija pojedinog projekta te slanje zahtjeva za suradnju na projektu. Vlasnik projekta prihvaća suradnike, ažurira stanje izrade projekta te se nakon dovršenog projekta otvara mogućnost recenzije suradnika na tom projektu. U ovom radu opisane su tehnologije korištene za izradu aplikacije, arhitektura aplikacije i baze podataka te funkcionalnosti specifične za temu aplikacije.

Ključne riječi: Blazor, C#, projekt, suradnja, web aplikacija

Web Application for Facilitating Collaboration Among Independent Programmers on Projects

Abstract

The theme of the thesis is an application solution that enables independent programmers to connect with other colleagues involved in creating software solutions in various programming languages and frameworks. The application is designed to assist independent programmers in finding jobs and facilitate collaboration with other programmers on open projects. For the technical implementation of the solution, the Blazor framework was used, which allows for fast and flexible application development. The main purpose of this application is to connect users with projects that match their specific criteria. During account creation, skills should be chosen and rated on a scale 1 to 5, where 1 represents basic knowledge and 5 represents excellent proficiency, by a programmer. Users can create their own projects and include information about the expected skills and proficiency level of collaborators who would apply for the project. The application then offers the user the ability to search for projects based on several different parameters, view information about each project, and send collaboration requests for a project. The project owner accepts collaborators, updates the project's progress, and after the project is completed, there is an option to review the collaborators on that project. The technologies used for developing the application, the architecture of the application and the database, as well as the functionalities specific to the theme of the application will be described.

Keywords: *Blazor, C#, collaboration, project, web applications*

Sadržaj

1. Uvod.....	1
2. Korištene tehnologije	3
2.1. C# programski jezik.....	3
2.1.1. Općenito	3
2.1.2. .NET programski okvir.....	4
2.1.3. ASP.NET Core	4
2.1.4. Web API framework	5
2.1.5. Entity Framework Core.....	7
2.1.6. LINQ.....	7
2.2. Blazor programski okvir.....	9
2.2.1. Modeli posluživanja Blazor aplikacija	11
2.2.2. Blazor WebAssembly.....	11
2.2.3. Blazor server	13
2.2.4. SignalR	14
2.3. Microsoft SQL Server	15
3. Arhitektura aplikacije.....	17
3.1. Arhitektura klijentskog dijela.....	19
3.2. Arhitektura poslužiteljskog dijela.....	20
3.3. Arhitektura baze podataka	21
4. Funkcionalnosti aplikacije.....	22
4.1. Stvaranje korisničkog računa	22
4.2. Pretraživanje projekata	24
4.3. Stvaranje novog projekta	25
4.4. Prikaz pojedinog projekta	26
4.5. Sustav za komunikaciju	27
5. Zaključak	33
6. Popis literature	35

1. Uvod

Web aplikacija „DevGuru“ polazi od ideje da se sve više programera okreće prema radu udaljenom od sjedišta tvrtke (rad od kuće) te je takav način rada postao bolja praksa od fizičkog dolaska na radno mjesto sa fiksnim radnim vremenom. Kako se tržište rada u području razvoja programskih podrška za razne probleme mijenja, nastala je potreba za digitalnim rješenjima koja bi pojedincu nudila mogućnost pronalaska posla i samostalnog zapošljavanja na nekom radnom mjestu. Internet je, pogotovo u današnje vrijeme, postao glavni medij za komunikaciju i povezivanje ljudi u svijetu što je otvorilo nove mogućnosti zapošljavanja i obavljanja posla iz udobnosti svog doma.

U procesu prilagodbe na takav način rada nastala je potreba za alatima koji bi programerima širom svijeta omogućili pronalazak adekvatnih radnih mjesta. Stoga je svrha ove aplikacije da samostalnim programerima na najefektivniji način omogući pokretanje projekata te definiranje cilja i traženih zahtjeva za izradu projekta. Sukladno tome olakšava se pronalazak odgovarajućih suradnika za potrebe sudjelovanja na projektima. Isto tako se pojedinom programeru nudi mogućnost da na što lakši način dođe do vlasnika malih projekata te se prijavi na traženo radno mjesto.

Primarna tehnologija korištena za izradu aplikacije je programski okvir Blazor koji koristi serverski model posluživanja, a kao glavni viši procesorski jezik korišten je C#. Na strani poslužitelja za izradu razgranate strukture Web API-a (engl. *Application Programming Interface*) korišten je .NET 7 programski okvir koji uz dodatak Entity Framework Core 7 programskog okvira i relacijske baze podataka Microsoft SQL omogućuje efikasno korištenje resursa te stvaranje složene arhitekture programskog koda. Korištenjem navedenih tehnologija postignuto je jednostavno rješenje složenog programskog zahtjeva čime se ubrzao proces izrade web aplikacije.

U ovom radu koji se sastoji od tri dijela opisać će se tehnologije koje su korištene za izradu ove aplikacije, a to su: .NET, C#, Blazor Server, Signal R, Microsoft SQL Server te Entity Framework. Nakon toga detaljno će biti opisana struktura baze podataka i način izvođenja. Zatim se u sljedećem dijelu opisuje programska arhitektura API-a i Blazor Server

projekta. U zadnjem poglavlju bit će opisani načini na koje je programsko rješenje realizirano te glavne funkcionalnosti aplikacije.

2. Korištene tehnologije

U ovom poglavlju bit će opisana svaka tehnologija koja je korištena u izradi praktičnog dijela rada. U glavnim poglavljima bit će navedene i općenito opisane tehnologije, dok će se u potpoglavljima razrađivati detalji vezani za pojedinu tehnologiju.

2.1. C# programski jezik

2.1.1. Općenito

C# je suvremen, objektno orijentiran programski jezik koji uključuje enkapsulaciju, nasljeđivanje i polimorfizam kao osnovne značajke objektno orijentiranog jezika te se bazira na tipovima podataka. C# omogućuje programeru izgradnju različitih vrsta sigurnih i robusnih aplikacija koje se izvode u .NET programskom okviru. C# spada u obitelji C jezika i lako je shvatljiv programerima koji su imali prethodno iskustvo u jezicima kao što su C, C++, Java i JavaScript. C# također je komponentno orijentirani jezik što bi značilo da se kroz razvoj aplikacije implementacijom raznih C# funkcionalnosti olakšava programeru stvaranje komponenti koda koje se mogu ponovo koristiti [1][2].

C# nudi mnoštvo značajka za izgradnju održivih aplikacija kao što je *Garbage Collection* koja automatski oslobađa memoriju zauzetu nedostupnim ili nekorištenim objektima. Rukovanje iznimkama je značajka koja omogućava strukturiran pristup otkrivanju grešaka i njihovog rješavanja. C# nudi mogućnost nulabilnih tipova podataka dodjeljivanjem null vrijednosti memorijskoj lokaciji objekta. Velik dio C# programskog jezika veže se za lambda izraze koji se prenose iz okruženja funkcijskog programiranja. Oni su osnovna sintaksa integriranih upita, tj. LINQ (engl. *Language Integrated Query*) izraza sa kojima je moguće manipulirati i upravljati svim vrstama setova podataka podržanih u C# programskom jeziku. Također velika značajka vrlo bitna za razvoj web stranica je asinkrono programiranje koje se implementira pomoću `async – await` ključnih riječi. Svi tipovi podataka u C#, uključujući i primitivne tipove kao što su `int` ili `double` nasljeđuju tip podatka `object` preko kojega se omogućava izvršavanje osnovnih operacija skladištenja, prijenosa ili manipulacije objektima. C# podržava korisnički definirane referencijalne tipove, generičke vrijednosne

tipove, delegate i dinamičke tipove podataka što povećava performanse izvođenja aplikacije. Jezik je podložan verzioniziranju što osigurava sigurnu i stabilnu okolinu za razvoj jer se na taj način popravljaju greške i nadograđuju mogućnosti koje jezik pruža [1][2].

2.1.2. .NET programski okvir

Kada se spominje C# programski jezik mora se uzeti u obzir da on ne dolazi kao samostalan proizvod već mu je potrebna specifična okolina za pokretanje. .NET Framework je platforma koju je razvio Microsoft, a služi kao temeljni pokretač servisa te konzolnih, web i Windows aplikacija napisanih u C# programskim jezikom. Osmišljen je kako bi osigurao konzistentnu objektno orijentiranu okolinu pogodnu za razvoj aplikacija koje se mogu zapisivati i pokretati lokalno, preko weba ili na daljinu [3].

Sastoji se od zajedničkog jezika za izvođenje CLR (engl. *Common Language Runtime*) koji pruža mogućnost upravljanja memorijom i široke biblioteke klasa kao što je ADO.NET ili WCF koje programerima omogućuju pozivanje i izvođenje kompleksnih dijelova koda. CLR je temelj .NET programskog okvira koji je zadužen za stanje memorije, izvođenje zadataka na nitima, provjeru sigurnosti koda, kompajliranje i druge systemske servise. Implementacija CLR-a vrši se preko sučelja naredbenog retka CLI (engl. *Command Line Interface*) koji kao višeplatformski alat za razvoj, nadogradnju i pokretanje dolazi uz .NET programski okvir. Bitno je spomenuti da se kroz .NET programski okvir jezici koji su pogodni toj okolini (C#, F#, VB.NET...) prilikom kompajliranja prevode u međukod koji se naziva CIL (engl. *Common Intermediate Language*). Na taj način osigurava se jednako izvođenje koda na svim vrstama procesora te se na taj način programeru olakšava razvoj aplikacije [3].

2.1.3. ASP.NET Core

ASP.NET Core je višeplatformski programski okvir otvorenog koda fokusiran na poboljšanje performansa, a koristi se za izgradnju modernih aplikacija povezanih na Internet. Core je novija, nadograđena verzija prethodnika ASP.NET koja ujedinjuje prethodno zasebne programske okvire MVC (engl. *Model-View-Controller*) web programski okvir i okruženje za izgradnju API-a. ASP.NET Core nudi mogućnosti izgradnje web aplikacija i usluga na

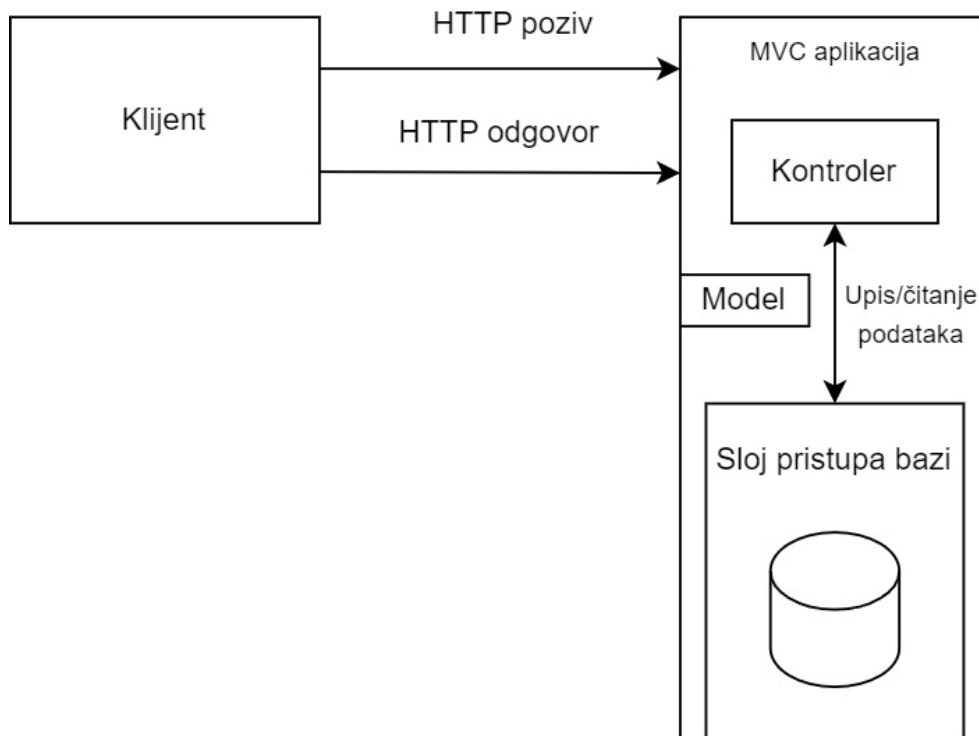
programskoj arhitekturi pogodnoj za testiranje. Omogućuje pisanje Razor sintakse na klijentskoj strani što uveliko olakšava stvaranje interaktivnog korisničkog sučelja za više različitih scenarija u aplikaciji. Sjedinjenjem web API i MVC tehnologije dobiva se okruženje u kojem je fokus na produktivnosti i smislenosti koda što se implementira preko model binding svojstva da se podatci iz HTTP zahtjeva u parametre metode mapiraju automatski i validacije modela podataka koju je moguće izvesti na klijentskoj i poslužiteljskoj strani [4].

Određena verzija ASP.NET Core-a zahtijeva korištenje podržane inačice razvojnog okruženja Visual Studio. Trenutno najnovija verzija sa postojećom podrškom je .NET 7, a zahtijeva Visual Studio 2022 verziju okruženja za pokretanje aplikacija, no već se krajem studenog 2023. godine očekuje nova verzija .NET 8.

2.1.4. Web API framework

Prema definiciji API u računalnom programiranju označava sučelje koje sadržava skup definicija potprograma, protokola i alata za izradu aplikacija bez korisničkog sučelja. Može se reći da je API neka vrsta sučelja koje ima skup ugrađenih funkcija koje programerima omogućuju pristup određenim HTTP (engl. *Hypertext Transfer Protocol*) značajkama [5].

U ASP.NET Core programskom okviru velik dio svodi se na Web API, jer upravo taj programski okvir predstavlja glavne ulazne i izlazne točke aplikacije koja se razvija. Preko njega se HTTP pozivima prenose podatci potrebni za prikaz na klijentskoj strani ili za upis i obradu korisničkih unosa te zatim unos u bazu podataka. Ovaj programski okvir omogućava automatsko serializiranje podatkovnih članova klase u pravilno formatirani JSON (engl. *JavaScript Object Notation*) oblik bez dodatnog postavljanja konfiguracijskih parametara. Uz jednostavan i brz način postavljanja također se nudi mogućnost prilagođavanja postavki serializacije za krajnje točke koje zahtijevaju kompleksnije operacije. Za komunikaciju se koristi HTTP koji radi na aplikacijskom sloju OSI modela [5].



Slika 1. Dijagram Web API arhitekture [12]

Priloženi dijagram na slici 1. opisuje način komunikacije klijenta i Web API-a te se može podijeliti u tri koraka. Prilikom slanja zahtjeva sa klijentske strane generira se HTTP poziv koji u sebi može ili ne mora sadržavati podatke formatirane u JSON obliku. Nakon toga zahtjev se zaprima na strani kontrolera te se izvršava dio koda koji je zadužen za metodu kontrolera koja je pozvana. U većini slučajeva podatci se sa API kontrolera šalju na sloj za obradu podataka te se dohvaćaju ili upisuju u bazu podataka. Nakon izvršenih akcija kontroler vraća odgovor klijentu u JSON formatu koji u sebi sadrži vrijednosti u obliku atribut-vrijednost.

2.1.5. Entity Framework Core

Entity Framework Core je službena platforma koju je izradio Microsoft te je nova izvedba starijeg Entity Frameworka koji se nakon šeste verzije počeo isporučivati usuglašeno sa verzijama .NET Frameworka. Entity Framework Core objavljen je široj javnosti po principima otvorenog koda te je zaslužan za mapiranje objekata proslijeđenih iz koda u bazu podataka ili obrnuto. To je nadogradnja na ADO.NET koji pruža programeru automatiziran mehanizam za pristupanje kontekstu baze podataka i pohranu podataka. Napravljen je s idejom da bude univerzalan ORM (engl. *Object Relational Mapping*) alat koji bi funkcionirao sa .NET objektima i svim vrstama relacijskih baza podataka kao što su MS SQL Server, SQLite, PostgreSQL, MySQL, OracleDB te mnoge druge. Postoje dva glavna načina na koje se Entity Framework Core može postaviti a to su: Code-First pristup i Database-First pristup. Prilikom izrade praktičnog dijela rada korišten je Database-First pristup što bi značilo da su se modeli u programskom kodu stvarali po definiciji tablica iz baze podataka. EF Core po odabranoj bazi podataka i njezinoj SQL shemi stvara kontekst baze u C# kodu i po njemu generira osnovne modele podataka po stupcima iz tablica. Na taj način se sva definirana pravila u bazi podataka, kao što su polja koja ne smiju imati null vrijednost, prenose u C# klase. Ovaj način još se naziva *Scaffolding* ili obrnuti inženjering te se vrlo lagano uz nekoliko obaveznih parametara izvršava naredbom Scaffold-DbContext. Code-first je pristup kojim se kroz C# kod stvaraju osnovne klase čije je ponašanje potrebno detaljno opisati. Pokretanjem prve migracije stvara se kontekst baze podataka te ga se može izmijeniti ili nadopuniti kroz modelBuilder koristeći lambda izraze. Nakon primjene migracije stvaraju se tablice i njihove veze po definiciji konteksta u C# kodu [6].

2.1.6. LINQ

LINQ je naziv za skup tehnologija koje omogućavaju pisanje integriranih upita sličnih onima u SQL-u u programski jezik C#. LINQ omogućava manipulaciju svim vrstama kolekcija u C# programskom jeziku bez obzira na to nalaze li se u kolekciji primitivni ili korisnički stvoreni tipovi podataka. Upiti se tako mogu koristiti za pretvaranje bilo kojih LINQ podržanih izvora u tok podataka neke druge vrste. Tako za primjer LINQ možemo

koristiti za prihvaćanje podataka iz SQL toka te ih prebaciti u XML izlazni tok podataka. LINQ sintaksa lako se uči, jer se sastoji od već poznatih jezičnih izraza vezanih za C#. Programer koji je navikao pisati SQL upite lako će se priviknuti na LINQ jer se njegov osnovni izgled upita veže na SQL. Koristeći sintaksu upita nad kolekcijom moguće je obaviti filtriranje, izmjenu redoslijeda po nekim uvjetima ili grupiranje po atributima uz minimalnu količinu koda. Važno je napomenuti da se upit neće okinuti sve dok se iteracijom ne pređe preko upita, kao što je npr. sa foreach petljom [7].

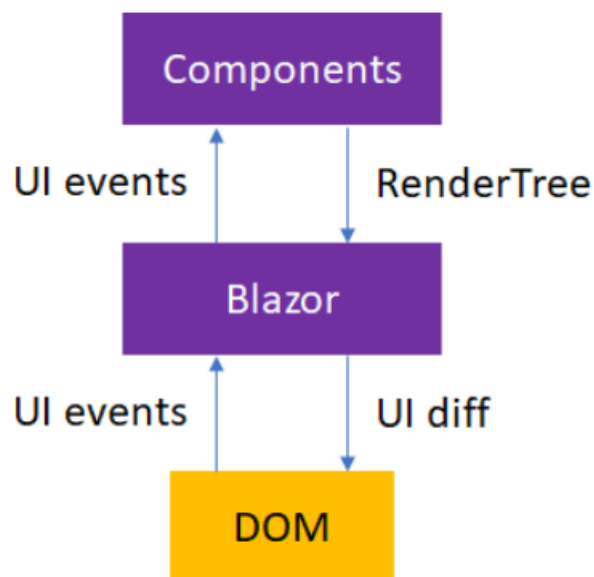
Isječak programskog koda 1: LINQ izrazi napisani različitim sintaksama

```
1. List<string> gradovi = new List<string>() { "Osijek", "Virovitica", "Zagreb", "Split",  
2.                                     "Koprivnica" };  
3. List<string> rezultat;  
4.  
5. //Query sintaksa  
6. rezultat = (from g in gradovi  
7.             where g.ToLower().Contains('a')  
8.             select g).ToList();  
9.  
10.  
11. //Method sintaksa  
12. rezultat = gradovi.Where(x => x.Contains('a')).ToList();
```

Isječak programskog koda 1. prikazuje listu u koju su zapisani nazivi gradova, te LINQ upite preko kojih se dohvaćaju svi gradovi koji u svojem imenu sadrže znak 'a'. Prvi način je napisan *query* sintaksom, a *method* sintaksom moguće je napisati cijeli LINQ izraz u jednoj liniji.

2.2. Blazor programski okvir

Blazor je klijentski web programski okvir koji je razvio Microsoft po prirodi sličan JavaScript klijentskim okvirima kao što su Angular ili React. Temeljna karakteristika mu je da programerima omogućava izgradnju bogatih i interaktivnih komponenti koristeći C# programski jezik umjesto JavaScripta. Mogućnost pisanja klijentske i poslužiteljske strane koda u jednom dokumentu (Razor komponenti) olakšava proces izrade aplikacije i daje veću preglednost koda Blazor rukuje interakcijama korisnika i renderira potrebne dijelove korisničkog sučelja. Za razliku od standardnih pristupa webu koji funkcioniraju na principima zahtjeva i odgovora, Blazor koristi događaje koji nisu dio standardnih HTTP poziva. Ovisno o odabranom modelu posluživanja Blazor će komponente, koje su temelj izgradnje Blazor aplikacije, renderirati na klijentskoj ili na poslužiteljskoj strani [8][9].



Slika 2. Prikaz stvaranja DOM-a iz razor komponenti [13]

Blazor aplikacija tvori se od komponenti koje u osnovi predstavljaju .NET klase kao komade korisničkog sučelja koji se mogu koristiti za višekratnu upotrebu. Svaka komponenta ima svoju klasu koja nasljeđuje `ComponentBase` klasu te se na taj način definiraju. U komponenti je moguće pisati HTML (engl. *HyperText Markup Language*), CSS (engl. *Cascading Style Sheets*) i C# kod koristeći Razor sintaksu što omogućava puno lakše pristupanje podacima. Imaju nastavak `.razor` te se zbog toga komponente u Blazoru nazivaju i Razor komponente. Svaka razor komponenta ima svoj vijek trajanja te za sebe održava svoje stanje i definira svoju logiku za renderiranje što može uključivati i renderiranje drugih komponenti. U komponenti moguće je definirati događaje koji će nakon korisničke interakcije pokrenuti renderiranje i obnavljanje podataka u komponenti. Nakon renderiranja komponente podatci se ne prebacuju direktno u DOM (engl. *Document Object Model*) već se po hijerarhiji slažu u `RenderTree`. Na taj način Blazor prati prijašnje stanje komponenti i po novim podacima učitanim u `RenderTree` renderira komponente u kojima je došlo do promjena [8][9].

Isječak programskog koda 2. Prikaz koda napisan razor sintaksom

```
1. @page "/counter"
2.
3. <PageTitle>Counter</PageTitle>
4.
5. <h1>Counter</h1>
6.
7. <p role="status">Current count: @currentCount</p>
8.
9. <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
10.
11. @code {
12.     private int currentCount = 0;
13.
14.     private void IncrementCount()
15.     {
16.         currentCount++;
17.     }
18. }
19.
```

U isječku koda 2. prikazana je `Counter.razor` komponenta koja se generira prilikom otvaranja novog Blazor projekta. Svaka razor komponenta u osnovi sastoji se od tri dijela: direktive, razor markup i poslužiteljski kod. U direktivama, koje se uvijek nalaze na vrhu razor komponente, uključuju se imenski prostori, ubacuju se servisi, definira se putanja do

komponente te se može definirati parametar koji komponenta prima preko URL-a. Drugim riječima, u direktivama se definiraju svi potrebni elementi za normalno funkcioniranje komponente. U središnjem dijelu koda nalaze se HTML elementi, odnosno sadržaj stranice koji se prikazuje na klijentskom dijelu. Taj dio koda sličan je standardnom pristupu izgradnji web stranica, no uz razor sintaksu moguće je koristiti podatkovne članove komponente uz znak „@“ što omogućava direktno pristupanje i izmjenu sadržaja na stranici bez osvježavanja web preglednika. Tako je u trećem dijelu koji se naziva logika komponente, koja zapravo predstavlja klasu razor komponente, definiran podatkovni član *currentCount* i metoda *IncrementCount*. U sadržaju stranice u paragrafu ispisuje se trenutno stanje broja koji je inicijalno postavljen na 0. Ispod njega nalazi se gumb koji na događaj klika poziva metodu *IncrementCount* u kojoj se varijabla *currentCount* povećava za 1. Klikom na gumb mijenja se vrijednost varijable, Blazor zatim registrira promjenu, renderira komponentu, izmjenjuje *RenderTree* i prikazuje novo stanje brojanika. Na taj način postiže se interakcija korisnika i izmjena podataka prikazanih u korisničkom sučelju bez osvježavanja stranice [8][9].

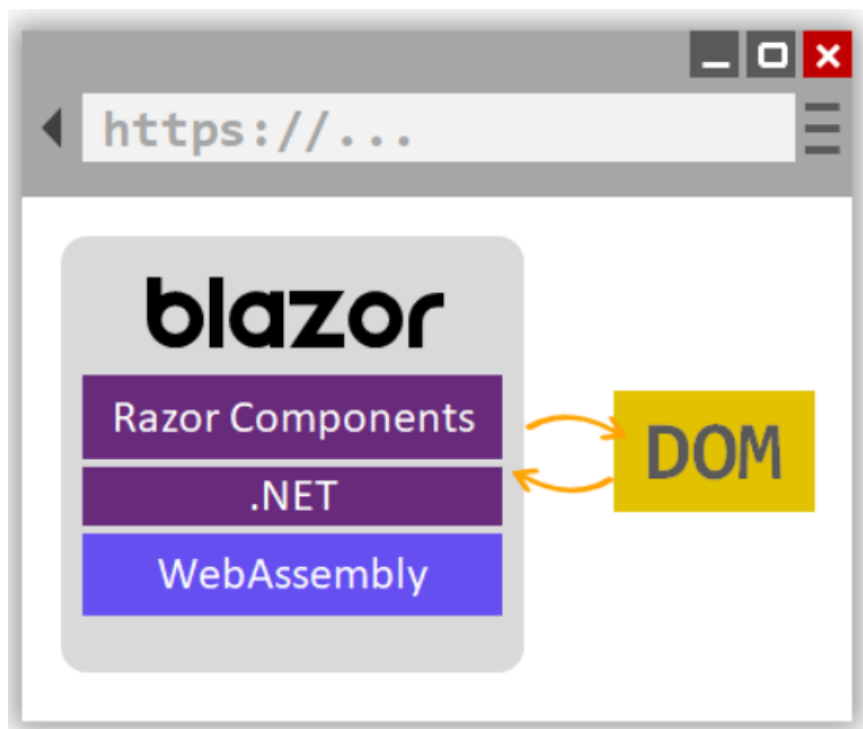
2.2.1. Modeli posluživanja Blazor aplikacija

Ideja Blazor aplikacija inicijalno je krenula od klijentske strane. Osmišljen je kao SPA (engl. *Single-page application*) programski okvir kojemu je cilj pokretati .NET aplikacije u web pregledniku koristeći *WebAssembly* i *Razor* sintaksu. Kombinacijom riječi *browser* i *Razor* dobiven je naziv *Blazor*. Tokom razvijanja ovog programskog okvira došao je i poslužiteljski model koji je proširio mogućnosti samog izvođenja i pokretanja aplikacije. Iako se modeli posluživanja razlikuju, mogućnost da se piše jedna baza koda koja bi posluživala oba modela stvorila je uniformiranu zajednicu i olakšala je rješavanje programskih izazova u razvoju aplikacija [9].

2.2.2. Blazor WebAssembly

Na klijentskoj strani posluživanja *Blazor* je omogućen *WebAssembly*jem. *WebAssembly* (*WASM*) je set binarnih instrukcija koji je stvoren kako bi pregledniku dao mogućnost prikazivanja kompajliranog koda više razine kao što je *C++*. *Blazor* aplikacije koje koriste *WebAssembly* pokreću se direktno u pregledniku na način da se .NET Core runtime kompajlira u *WASM*, a zatim se zajedno s aplikacijom kompajliranom u

Intermediate Language šalje na klijentsku stranu. Za pokretanje Blazor WASM aplikacije nisu potrebni dodatci za browser. Kada se aplikacija pokrene .NET runtime je usmjeren prema *assembly* kodu aplikacije nakon čega se učitava korijenska komponenta. Blazor izračunava *RnderTree* i njegove komponente te po tome osvježava stanje DOM-a što je vidljivo sa slike 3. Cijela aplikacija sa svim potrebnim kompajliranim .NET bibliotekama pokreće se u preglednikovom sandboxu što je okolina koja štiti jezgru preglednika od zlonamjernih akcija na klijentskoj strani. Prednost WebAssembly pristupa Blazoru je to što ne ovisi o ASP.NET Core web serveru, već je aplikaciju moguće posluživati preko CDN-a (engl. *Content Delivery Network*) ili preko statičnog hostinga kao što je GitHub Pages ili Azure Microsoftovog servisa. Blazor WebAssembly aplikacija također može raditi i bez povezanosti s internetom jer se cijeli sadržaj aplikacije nalazi u memoriji preglednika. U trenutku ponovnog povezivanja na mrežu Blazor će osvježiti podatke nakon što uspostavi vezu sa serverom pomoću SignalR-a ili pozivom na API. [9].

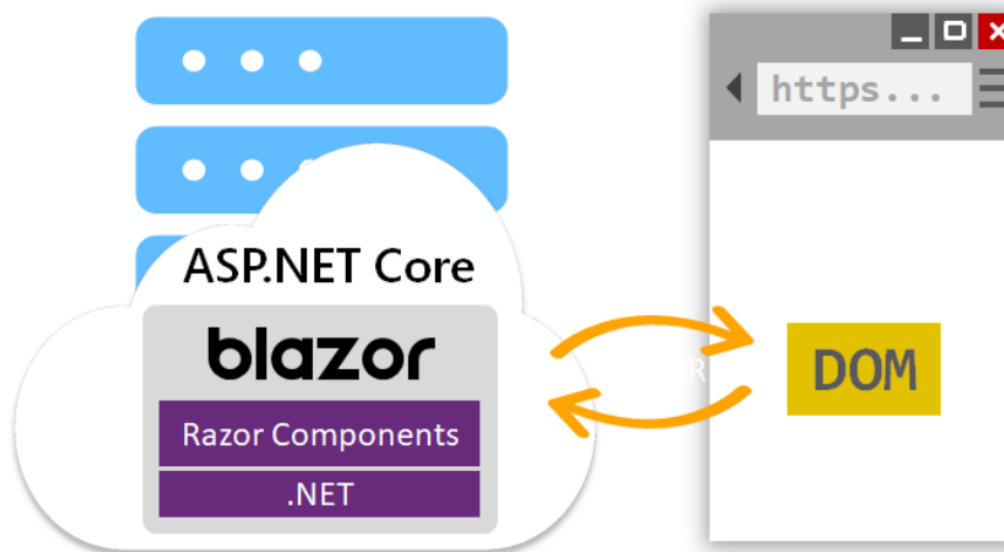


Slika 3. Komunikacija Blazora i web preglednika u WebAssembly načinu izvođenja [9]

2.2.3. Blazor server

U prijašnjim odlomcima spominjala se RenderTree apstrakcija koja predstavlja stablo svih učitanih komponenti, tj. sadržaja web stranice. Blazor programski okvir inicijalno gradi RenderTree prilikom prvog pokretanja aplikacije, a zatim se nakon svake interkcije ili promjene korisničkog sučelja generira novi RenderTree te se uspoređuje sa prethodnom verzijom. Sukladno tome same komponente ne moraju se pokretati u procesu osvježavanja korisničkog sučelja, te se zapravo ti procesi ne moraju niti pokretati na istom domaćinu [9].

U Blazor Server aplikacijama komponente se pokreću na poslužitelju umjesto preglednika koji se nalazi na klijentskoj strani kao što je to slučaj kod WebAssembly-a. Događaji se okidaju na klijentskoj strani, a poslužitelj se o tome obavještava u stvarnom vremenu. Za to je zadužena Microsoftova biblioteka SignalR koja stvara sigurnosni tunel između poslužitelja i klijenta. Kada se okinutim događajima izmjeni korisničko sučelje komponentama se na poslužiteljskoj strani putem SignalR-a dostavljaju informacije o tome što se promijenilo. Zatim se na poslužitelju računa novo stanje RenderTree-a i uspoređuje sa prijašnjim stanjem. Izmijenjeno stanje RenderTree-a se zatim primjenjuje na DOM i dolazi do izmjena korisničkog sučelja bez osvježavanja web preglednika [9].

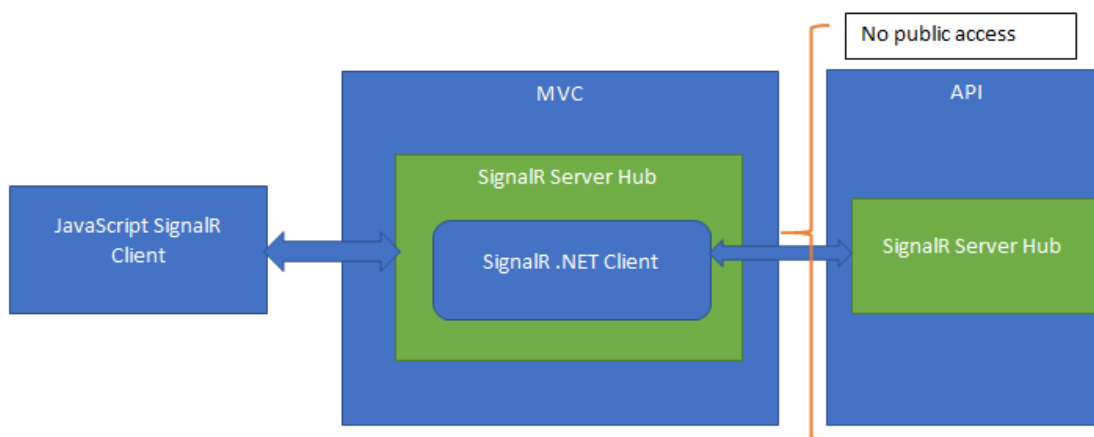


Slika 4. Komunikacija Blazora na poslužitelju i web preglednika [9]

Prednost Blazor Server aplikacija je to što ne ovise o klijentskoj strani što dovodi do boljih performansa jer se svi resursno zahtjevni zadatci obavljaju na poslužitelju. Blazor Server aplikacije imaju širi spektar uređaja na kojima se mogu prikazivati, pa tako i na web preglednicima koji ne podržavaju WebAssembly. Količina podataka koja se preuzima sa poslužitelja na web preglednik je znatno manja za razliku od WebAssembly-a što dovodi do bržeg učitavanja web stranice. Mana Blazor servera je u tome što zahtjeva stalnu povezanost sa internetom, tj. poslužiteljem. Prilikom gubitka veze aplikacija prestaje funkcionirati [9].

2.2.4. SignalR

SignalR je biblioteka otvorenog koda koju je razvio Microsoft. Temeljni zadatak joj je da programeru pojednostavi implementaciju funkcionalnosti koje djeluju u stvarnom vremenu. Daje pojednostavljenu mogućnost stalne veze između klijenta i poslužitelja pa se tako programeru šire mogućnosti za razvoj web aplikacije. Primjenjuje se u aplikacijama koje zahtijevaju push notifikacije za obavijesti ili kanala za komunikaciju u stvarnom vremenu. Također može naći primjenu u nadzornim aplikacijama, aplikacijama sa sadržajem kojem pristupa više korisnika u isto vrijeme ili aplikacijama koje koriste GPS koordinatne sustave za praćenje. SignalR pruža API za stvaranje poslužitelj-klijent poziva koji se nazivaju RPC (engl. *Remote Procedure Calls*). Tim pozivom sa poslužitelja, koji radi na .NET Core kodu, se dohvaćaju funkcije napisane na klijentskoj strani [10].



Slika 5. Prikaz komunikacije servera sa klijentom putem SignalR-a [14]

Postoji više podržanih platformi za pisanje klijentske strane koja „sluša“ promjene na poslužitelju tako da biblioteka SignalR ne ovisi isključivo o C# programskom jeziku već se klijentski dio može pisati JavaScriptom ili TypeScriptom. SignalR po mogućnostima klijenta i poslužitelja automatski odabire način za prijenos podataka a to mogu biti: WebSocket, Server-Sent događaji ili Long Polling [10].

SignalR biblioteka bazira se na hubovima. Hub označava kanal provodnik koji omogućava klijentu da poziva metode sa poslužitelja i obrnuto. SignalR osigurava otpremanje podataka do druge strane bez dodanog konfiguriranja kanala. SignalR omogućava slanje različitih tipova podataka te podržava model binding. Proces razmjene podataka između klijenta i poslužitelja funkcionira tako da klijent šalje poruku koja sadrži naziv metode koja je napisana na poslužitelju i potrebne parametre koje metoda prima. Podatci koji su zaprimljeni sa strane poslužitelja se slažu po postavkama zadanog protokola. Nakon obavljenog poziva poslužitelj obavlja zadatke napisane u metodi [10].

2.3. Microsoft SQL Server

Općenito, SQL Server je relacijski sustav za upravljanje bazama podataka (RDBMS) koji podržava širok spektar funkcionalnosti obrade, ugradnje poslovne logike i analize sadržaja baze podataka. Relacijske baze podataka temelje se na vezama između podataka, a glavna gradivna jedinica je relacija tj. tablica. Uz OracleDB i IBM-ovu DB2 bazu podataka, Microsoft SQL Server je jedno od najpopularnijih okruženja za razvoj i održavanje baze podataka [11].

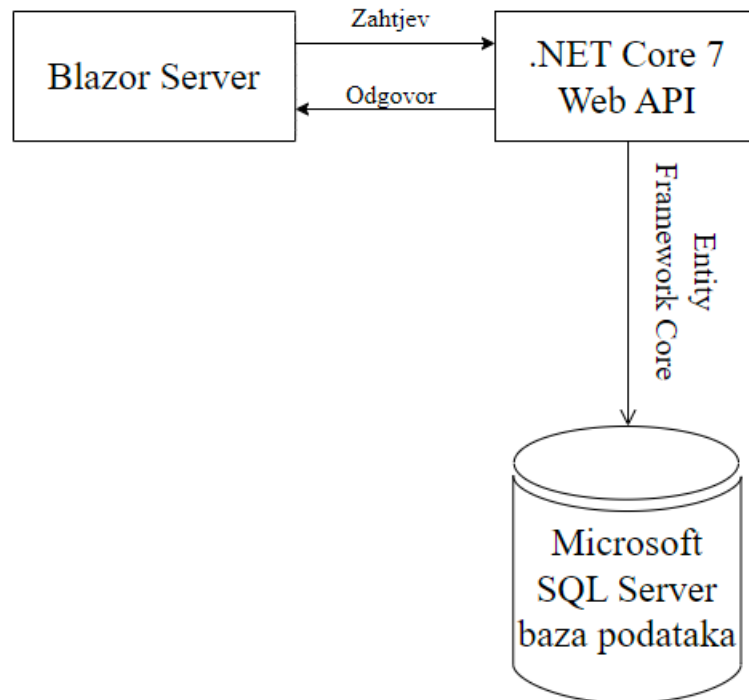
SQL Server Management Studio (SSMS) glavni je alat za upravljanje Microsoft SQL Server-om koji je prvi puta izdan uz SQL Server 2005. Sveobuhvatan je alat namijenjen programerima baza podataka jer svojim grafičkim sučeljem uvelike olakšava rad sa bazama podataka. Alat u sebi sadrži preglednik za uređivanje SQL skripti i grafičko sučelje za hijerarhijski prikaz objekata i operacija koje se mogu obavljati nad njima. Glavna značajka SSMS-a je Object Explorer koji programeru omogućava lakše snalaženje i pretraživanje po stvorenim bazama podataka i njihovim objektima. Ovaj alat nudi razne mogućnosti od kojih

su neke: vizualni prikaz objekata u bazi, editor za pisanje upita uz uključen IntelliSense, stvaranje grafičkih dijagrama baze podataka, skriptiranje baze i mnoge druge [11].

Microsoft SQL Server podržava naprednije SQL jezika koje se naziva Transact SQL ili T-SQL. Razvili su ga Microsoft i Sybase za potrebe logički kompleksnih zadataka i naprednijih pretraživanja tablica i sadržaja u bazi podataka. Kao jednu od najkorištenijih mogućnosti treba spomenuti procedure koje predstavljaju komade logike koji se mogu integrirati u upit te tako preciznije odrediti traženi set podataka. Procedure mogu primiti parametre poslane iz programskog koda. U procedurama moguće je pisati petlje, okidati upite unutar procedure te spremati rezultate određenih upita u varijable te i kasnije koristiti [11].

3. Arhitektura aplikacije

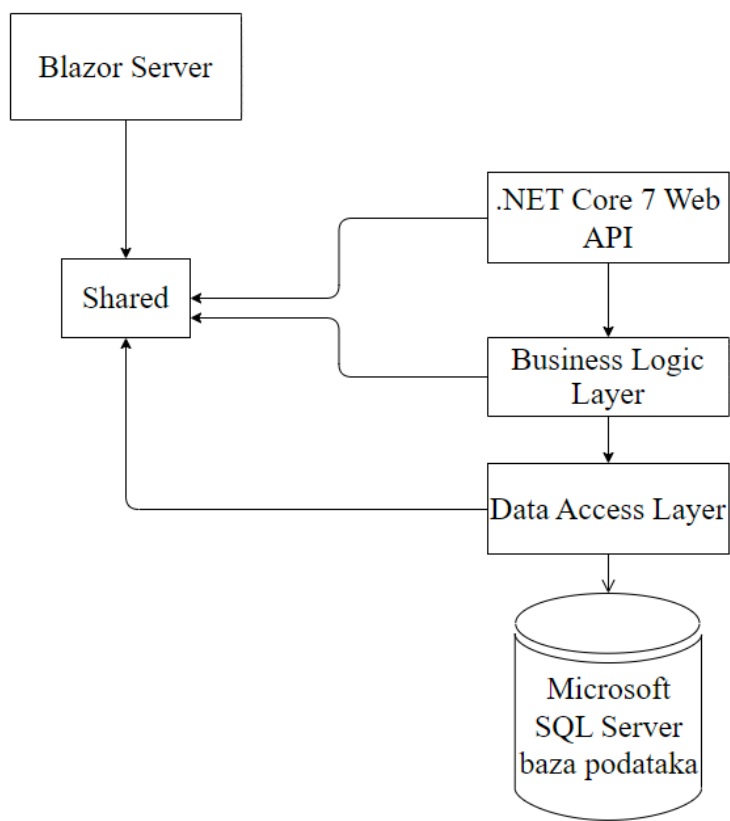
U ovom poglavlju detaljno će se opisati arhitektura programskog rješenja, slojevi od kojih se aplikacija sastoji te na koji način su oni povezani. Zasebno će se opisati svaki sloj i njegova svrha te će se opisati struktura baze podataka i povezanosti tablica. Za izradu praktičnog dijela ovog rada korištena je troslojna arhitektura koja omogućava odvajanje slojeva za poslovnu logiku i pristup bazi podataka od klijentskog dijela kojemu korisnik može pristupiti. Ovaj način stvaranja aplikacije olakšava debugiranje tokom razvoja te omogućuje testiranje zasebnih slojeva. Na slici 6. može se vidjeti grafički prikaz glavnih dijelova sustava.



Slika 6. Shema arhitekture aplikacije DevGuru

Aplikacija DevGuru logički se sastoji od tri dijela: prezentacijskog dijela koji predstavlja Blazor Server projekt, poslužiteljskog dijela koji se tvori od Web Api projekta uz slojeve poslovne logike i sloja pristupa bazi te od trećeg dijela kojeg predstavlja baza podataka. Prezentacijski dio predstavlja Frontend, tj. korisničko sučelje i sve što je potrebno

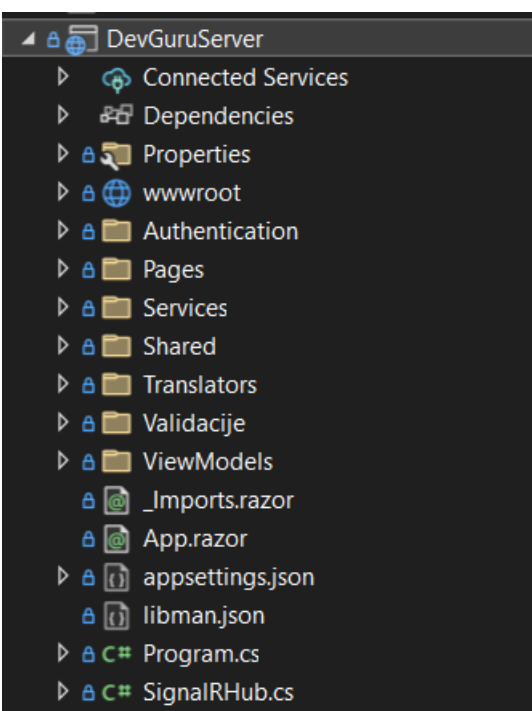
za normalan rad i interakciju s sadržajem koji se prikazuje u web aplikaciji. Backend dio, tj. REST Web API zadužen je za prikupljanje, obradu i prosljeđivanje podataka od korisničkog sučelja do baze podataka i obratno. Backend dio aplikacije sastavljen je po načelima i pravilima višeslojne arhitekture. Sastoji se od samog Web API projekta koji u sebi sadržava kontrolere zaslužne za dohvat i slanje podataka, BLL (engl. *Business Logic Layer*) sloja što je tip Class Library projekta na kojem se obavlja dio logike vezane za sadržaj aplikacije i obrađuje podatke te ih prilagođava formatu pogodnom za slanje na API vrata ili prema DAL (engl. *Data Access Layer*) sloju koji je zadužen za komunikaciju s bazom podataka. Na DAL-u, koji je također Class Library tip projekta, se nalaze metode za pristup kontekstu baze podataka, prevoditelji tipova podataka i modeli stvoreni po tablicama iz baze podataka za što je odgovoran Entity Framework Core. U programskom rješenju također se nalazi SharedLib Class Library projekt koji u sebi sadržava DTO (engl. *Data Transfer Object*) modele kojima se pristupa sa Frontenda i Backenda prilikom prijenosa podataka sa API-a.



Slika 7. Prikaz relacija između projekata unutar Visual Studia

3.1. Arhitektura klijentskog dijela

Kao što je u uvodu rečeno, klijentski dio praktičnog dijela rada napravljen je koristeći Blazor Server programski okvir. Na slici 8. prikazane su mape i datoteke koje se nalaze unutar klijentskog dijela. Unutar mape Authentication nalazi se prilagođena klasa *UserSession* i *CustomAuthenticationStateProvider* klasa koja nasljeđuje klasu *AuthenticationStateProvider*, a pruža mogućnosti spremanja podataka trenutno prijavljenog korisnika u sesiju. Mapa Pages sadržava sve



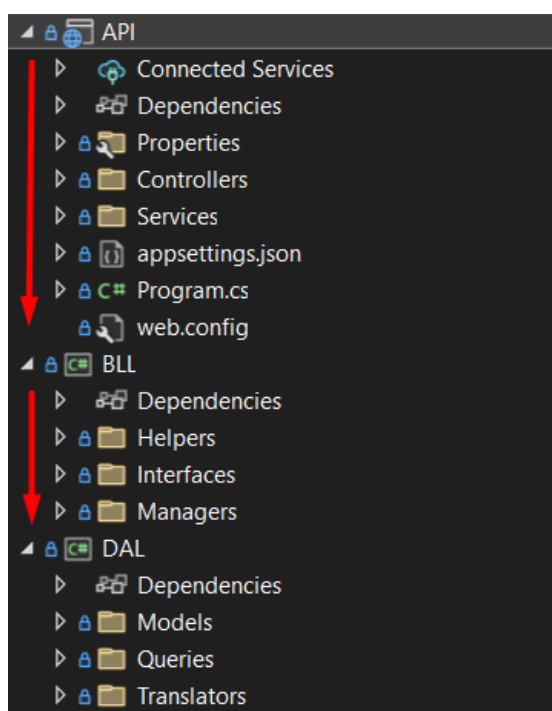
Slika 8. Arhitektura Blazor Server projekta

Mapa *Pages* u sebi sadržava sve *Razor* komponente, tj. sve prikaze koji se mogu naći na web stranici. Svaka razor komponenta u svojoj pozadini sadržava .cs dokument koji predstavlja klasu vezanu za tu komponentu. Mapa *Services* u sebi sadrži klase servisa koji služe za slanje zahtjeva na API te infrastrukturu sučelja koja se vežu na servise. U datoteci *Program.cs*, osim inicijalnog koda potrebnog za pokretanje aplikacije, nalaze se servisi koji su registrirani pomoću *Dependency Injection*-a u kontejner tipa *Singleton*. U mapi *Shared* nalaze se *Razor* komponente koje su dijeljene preko korištenja cijele aplikacije kao što je

navigacijski izbornik i podnožjem stranice. U *_Imports.razor* dokumentu definiraju se svi *Using* tagovi koji će se primijeniti u cijelom projektu.

3.2. Arhitektura poslužiteljskog dijela

Poslužiteljski dio sastoji se od Web Api projekta koji predstavlja ulazna i izlazna vrata Backenda. Klijentski dio koji prikazuje sadržaj na stranicama preko servisa poziva određene metode preko kojih se poziva API. Na slici 9. prikazani su Web API i Class Library projekti koji se vežu na njega te reference između projekata.



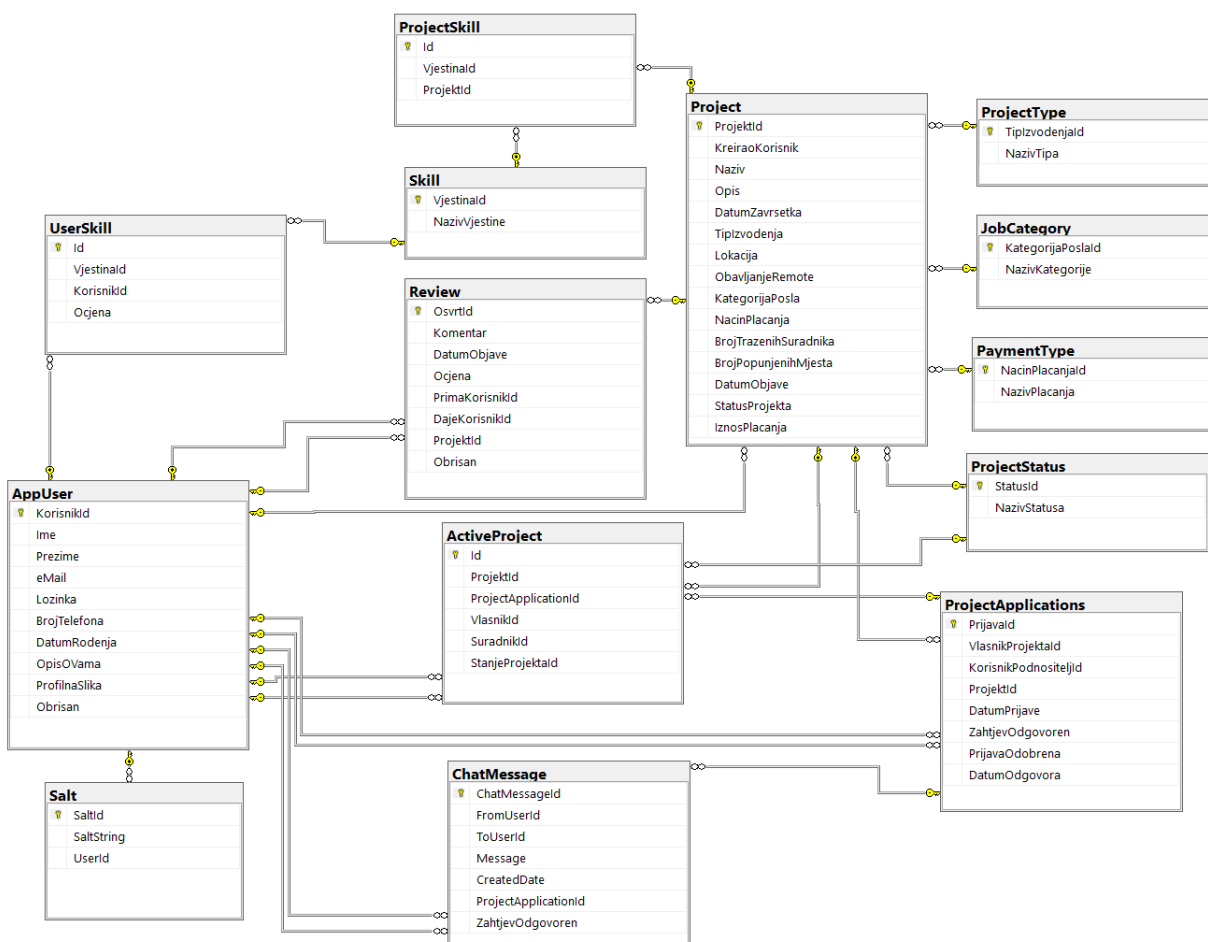
Slika 9. Arhitektura Blazor Server projekta

Zahtjev koji dođe na API vrata se preko *Managera*, koji su definirani na BLL-u i registrirani kao servisi u Web Api projektu, prosljeđuju na sloj odgovoran za obavljanje poslovne logike. Metode u sloju poslovne logike zatim pozivaju statične metode iz *Query* klasa koje se nalaze na sloju za komunikaciju. U ovom koraku iz konteksta baze podataka se dohvaćaju ili upisuju podatci, te se istim putem vraćaju nazad na BLL gdje se obavljaju sve operacije računanja, izmjene, sastavljanja ili prevođenja modela podataka u oblik pogodan za prosljeđivanje do API sloja. Preporuka je da se API kontroleri drže što urednijima, tj. da

sadrže isključivo logiku za prosljeđivanje podataka do nižih slojeva i odgovor na zahtjev, te da se sve operacije oko podataka vrše na drugim mjestima. Tako se npr. autentifikacija korisnika provodi na BLL-u uz korištenje pomoćne klase za *hash* i *salt* lozinke koje je korisnik unio tokom prijave ili registracije.

3.3. Arhitektura baze podataka

Na slici 10. prikazana je shema baze podataka na kojoj su prikazane sve tablice iz baze, te njihovi primarni i strani ključevi koji tvore relacije između tablica.



Slika 10. Shema baze podataka

4. Funkcionalnosti aplikacije

U ovome poglavlju biti će opisan tok izvođenja akcija i funkcionalnosti aplikacije. Veća poglavlja u aplikaciji, uz stvaranje korisničkog računa i prijavu, predstavljaju pregled svih projekata uz mogućnost pretraživanja po odabranim kriterijima, pregled pojedinog projekta, stvaranje novog projekta, pregled korisničkih računa i sustav za razmjenu poruka integriran u aplikaciju.

4.1. Stvaranje korisničkog računa

Prilikom pokretanja web stranice prikazuje se početna stranica s koje je moguće pregledati i po kriterijima pretražiti sve projekte. Korisniku se klikom na gumb Registracija nudi mogućnost stvaranja korisničkog računa ili se, ako posjeduje korisnički račun, klikom na gumb Prijava može prijaviti u sustav.

Kreiranje korisničkog računa razdvojeno je na tri dijela. U prvom dijelu od korisnika se traži unos podataka potrebnih za prijavu u sustav, a to su adresa elektroničke pošte i lozinka. Ukoliko se polja ostave praznim, lozinka kraća od 6 znakova ili se unese nevaljana adresa elektroničke pošte ispod polja za unos korisniku će se ispisati prikladne validacijske poruke. U drugom koraku se od korisnika traži unos osobnih podataka. Na ekranu se prikazuju polja u koja korisnik unosi svoje ime i prezime, datum rođenja, broj telefona za kontakt i opis koji se prikazuje na njegovom korisničkom profilu. Korisniku se također nudi opcija odabira profilne slike, no nije obavezan za stvaranje računa. U trećem dijelu registracije korisniku se prikazuje izbornik s vještinama koje može odabrati i ocijeniti svoje znanje pojedine vještine. Modal za ocjenjivanje vještina prikazan je na slici 11.

 Ocjenite svoje znanje!

Dodjelite ocjenu od 1-5 za svoje znanje **React.js** vještina



Napredno

ZATVORI

SPREMI

Slika 11. Prikaz modala za ocjenjivanje vještine

Nakon registracije korisniku se prikazuje početna stranica sa predloženim projektima koji se generiraju po vještinama koje si je korisnik dodijelio prilikom registracije. U isječku programskog koda 3. prikazuje se metoda kojoj se prosljeđuje lista identifikatora vještina koje korisnik ima registrirane na svom računu. Metoda po toj listi dohvaća projekte kojima je dodijeljen pojedini identifikator iz liste vještina korisnika, te se sa identifikatorom i nazivom projekta spremaju u *Dictionary*.

```
1. public static async Task<List<Project>> DohvatiSlicneProjektePoSkillID
   (DevGuruContext db, List<int> skillsID)
2. {
3.     try
4.     {
5.         List<Project> output = new();
6.
7.         var projekti = await db.Projects.Include(x => x.TipIzvođenjaNavigation)
8.             .Include(x => x.KategorijaPoslaNavigation)
9.             .Include(x => x.TipIzvođenjaNavigation)
10.            .Include(x => x.StatusProjektaNavigation)
11.            .Include(x => x.NacinPlaćanjaNavigation)
12.            .Include(x => x.ProjectSkills).ThenInclude(x => x.Vjestina)
13.            .Include(x => x.KreiraoKorisnikNavigation).ToListAsync();
14.
15.         foreach (var projekt in projekti)
16.         {
17.             var listaIntova = projekt.ProjectSkills.Select(x=>x.VjestinaId).ToList();
18.             foreach (int skillID in skillsID)
19.             {
20.                 if (!output.Contains(projekt) && listaIntova.Contains(skillID))
21.                 {
22.                     output.Add(projekt);
23.                 }
24.             }
25.         }
26.         return output;
27.     }
28.     catch (Exception ex)
29.     {
30.         throw new Exception(ex.Message);
31.     }
32. }
33.
```

4.2. Pretraživanje projekata

Prilikom otvaranja stranice korisniku se prikazuju svi projekti spremljeni u bazu poredani po datumu silazno. Na vrhu stranice nalaze se padajući izbornici pomoću kojih je moguće pretraživati projekte. Odabirom svojih najboljih vještina prema korisniku, olakšava se pronalazak odgovarajućih projekata. Također je moguće odabrati kategoriju posla, tip projekta ili način plaćanja. Ispod padajućih izbornika nalazi se polje za unos koje predstavlja traku za pretraživanje. Korisnik tako umjesto točnog naziva projekta može unijeti ključne riječi koje ga zanimaju te će mu se prikazati projekti koji mogu sadržavati sličan naziv ili opis. Na slici 12. prikazan je dio stranice za pretraživanje i primjer kartice projekta na kojoj su prikazane najbitnije informacije vezane za projekt kao što su naziv projekta, koji korisnik i kada ga je stvorio te tražene vještine za suradnju na projektu.

Odaberite vještine Sve

Odaberite kategoriju Sve

Odaberite tip projekta Sve

Odaberite način plaćanja Sve

Pretražite projekte po naslovu ili opisu

Korisnik: [Vatroslav Krpan](#) Datum objave: 30.8.2023.

Web aplikacija za administraciju pogona šećerane

Tražene vještine: .NET | .NET Core Web API | ASP.NET MVC

Slika 12. Prikaz filtera za pretraživanje i kartica projekta

4.3. Stvaranje novog projekta

U navigacijskom izborniku klikom na gumb *Stvori novi projekt* otvara se stranica sa formom za stvaranje novog projekta. Od korisnika se traži unos nekih osnovnih podataka o projektu kao što su naziv, opis ili traženi broj suradnika na projektu. Korisnik također iz padajućih izbornika odabire tip izvođenja projekta tj. tražene suradnje, kategoriju projekta tj. vrstu aplikacije koja će se razvijati i način plaćanja te iznos. Korisnik prilikom stvaranja projekta može odabrati datum nakon kojega će se prijave za projekt automatski zatvoriti ili može označiti da želi samostalno zatvoriti prijave na projekt što mu se omogućuje na stranici pregleda projekta. Drugim suradnicima se može omogućiti rad na daljinu, a moguće je unijeti i fizičku lokaciju izvođenja projekta. Iz popisa vještina potrebno je odabrati koje znanje vještina će se tražiti od suradnika prilikom izrade projekta. Forma za unos podataka o novom projektu vidljiva je na slici 13.

The image displays two parts of a web application interface. On the left is a form titled "Stvori novi projekt" (Create new project). It contains several input fields and checkboxes:

- "Naziv projekta" (Project name): "Web aplikacija za administraciju pogona šećerane"
- "Unesite datum" (Enter date): empty field with a calendar icon.
- Checkbox: "Želim samostalno zatvoriti prijave" (checked).
- Section: "DATUM DO KOJEGA SU PRIJAVE OTVORENE" (Date until applications are open).
- "Unesite lokaciju izvođenja projekta" (Enter project location): "Virovitica".
- Checkbox: "Remote obavljanje posla" (unchecked).
- "Odaberite tip izvođenja projekta" (Select project type): "Cijeli projekt".
- "Odaberite kategoriju projekta" (Select project category): "Web development".
- "Odaberite način plaćanja" (Select payment method): "Po dogovoru".
- "Unesite iznos plaćanja u eurima" (Enter payment amount in euros): "€ 0".
- "Unesite broj traženih suradnika na projektu" (Enter number of required team members): "5".

 On the right is a section titled "Popis vještina" (Skill list) with the instruction "Odaberite vještine koje trebate od svojih suradnika na ovom projektu" (Select skills you need from your team members on this project). It contains a list of skills with checkboxes:

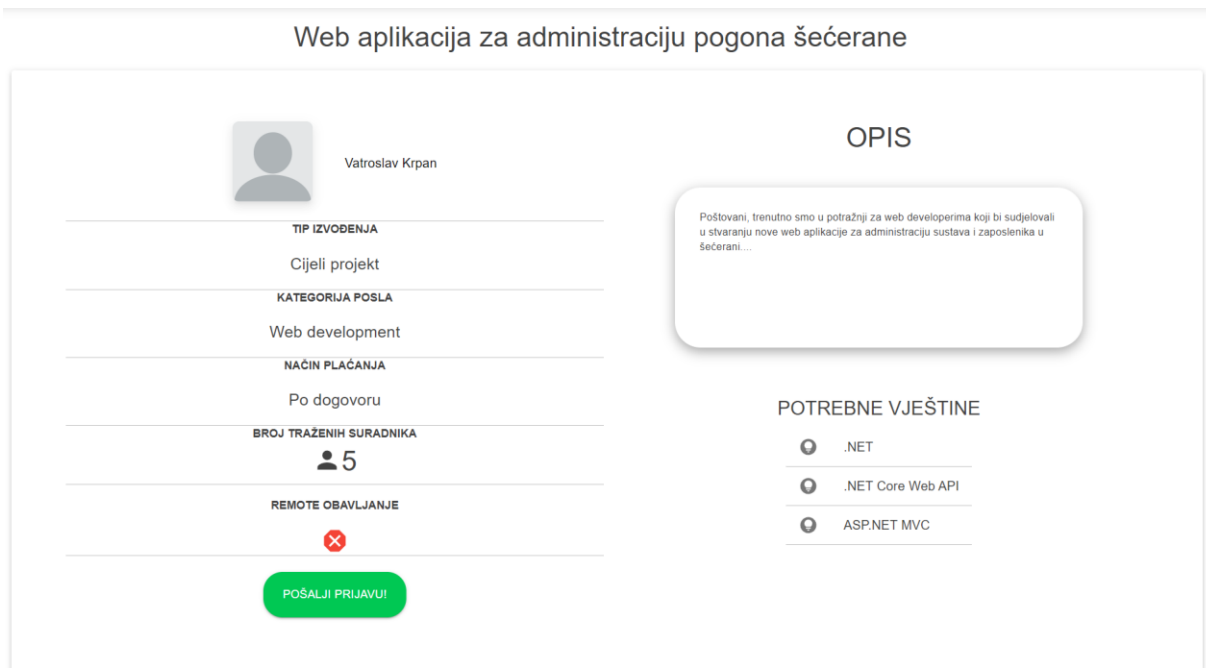
- .NET
- .NET Core Web API
- ASP.NET
- ASP.NET MVC
- VB.NET
- Vue.js
- AngularJS
- Node.js
- Next.js
- React.js

Slika 13. Prikaz forme za unos podataka novog projekta

4.4. Prikaz pojedinog projekta

Nakon stvaranja projekta on postaje vidljiv drugim korisnicima na stranici pregleda svih projekata. Na slici 14. može se vidjeti novi stvoreni projekt iz pogleda drugog prijavljenog korisnika. Na pregledu pojedinog projekta, ukoliko sva mjesta nisu popunjena i nije prošao rok za prijave na projekt, korisniku se klikom na gumb nudi mogućnost slanja zahtjeva za suradnju.

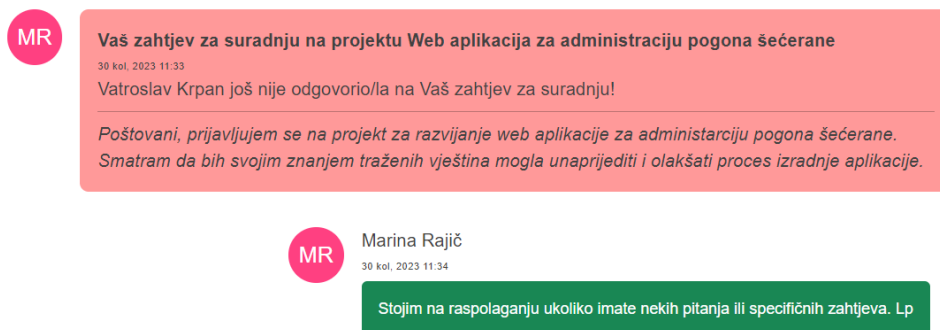
Nakon klika gumba otvara se modal u kojem korisnik mora upisati svoju zamolbu za suradnju kako bi se zahtjev za prijavu na projekt poslao. Nakon slanja poruke korisnika se preusmjerava na sustav za komunikaciju integriran u aplikaciju, tj. *chat*. U *chatu* su vidljive prijašnje poruke između korisnika i novi stvoreni zahtjev za suradnju na projektu zajedno s porukom zamolbe koja je poslana kroz modal za prijavu.



Slika 14. Prikaz podataka pojedinog projekta

4.5. Sustav za komunikaciju

Prilikom slanja zahtjeva za suradnju na projektu korisniku koji je vlasnik tog projekta dolazi prijava kroz sustav za komunikaciju. Zahtjevi za suradnju vizualno su označeni drugačijom bojom i sastavom poruke kako bi se razlikovale od običnih poruka poslanih kroz sustav za komunikaciju što je vidljivo sa slike 15.



Slika 15. Prikaz obične poruke i zahtjeva za suradnju u chatu

Stranica za prikaz sustava za komunikaciju se također može otvoriti i kroz gumb na navigacijskoj traci koji se korisniku prikazuje nakon prijave. Prilikom otvaranja stranice prikazuje se glavni prozor u kojemu se prikazuju poruke i lista kontakata sa desne strane. U kontaktima su prikazane osobe sa kojima je korisnik već imao povijest razgovora.

Sa druge strane, korisnik koji je zaprimio zahtjev za suradnju na projektu, može ga pronaći u chatu te mu se klikom na zahtjev otvara modal za slanje odgovora. Kao što je vidljivo sa slike 16. vlasnik projekta u modalu za odgovor može vidjeti poruku koja je poslana kao zamolba za suradnju, te može napisati odgovor na zamolbu i prihvatiti ili odbiti suradnju korisniku koji šalje zahtjev.

The image shows a modal dialog box with a dark border. At the top, there is a title "Zahtjev za prijavu na projekt" with a pencil icon. Below the title, the text reads: "Korisnik: Marina Rajič Vam je poslao/la zahtjev za prijavu na projekt Web aplikacija za administraciju pogona šećerane". A horizontal line separates this from a message box containing the text: "Poštovani, prijavljujem se na projekt za razvijanje web aplikacije za administraciju pogona šećerane. Smatram da bih svojim znanjem traženih vještina mogla unaprijediti i olakšati proces izradnje aplikacije." Below the message box, the text "Pošaljite odgovor na zahtjev za suradnju!" is displayed. A large, empty text input field is provided for the user's response. At the bottom of the modal, there are three buttons: a grey "ZATVORI" button, a red "ODBIJ ZAHTJEV ZA SURADNJU" button, and a green "PRIHVATI NOVOG SURADNIKA" button.

Slika 16. Modal za slanje odgovora na zahtjev za suradnju

Odbijanjem suradnje korisniku se u chatu izmjenjuje poruka poslana kao zahtjev za suradnju te se korisniku podnositelju šalje odgovor koji je vlasnik projekta napisao kao

obrazloženje odbijenog zahtjeva. Ako se zahtjev za suradnju ipak odobri, obnavlja se stanje broja traženih suradnika na projektu te se na isti način kao i kod odbijanja suradnje korisniku kroz chat daje do znanja da je suradnja prihvaćena. U isječcima programskog koda 4. i 5. prikazan je način na koji funkcionira razmjena poruka preko *hub-a*. U klasi *hub-a* definirane su metode preko kojih se šalju poruke i notifikacije, a u isječku broj 5 prikazan je dio koda koji prima obavijest o tome da je došla nova poruka te obavještava korisnika kroz *Snackbar*.

Isječak programskog koda 4: SignalR hub i metode koje prenose podatke do drugih klijenata

```
1. public class SignalRHub : Hub
2. {
3.     public async Task SendMessageAsync(ChatMessageDTO message, string userName)
4.     {
5.         await Clients.All.SendAsync("ReceiveMessage", message, userName);
6.     }
7.     public async Task ChatNotificationAsync(string message, string receiverUserId,
8.                                           string senderUserId)
9.     {
10.        await Clients.All.SendAsync("ReceiveChatNotification", message, receiverUserId,
11.                                    senderUserId);
12.    }
```

Isječak programskog koda 5: SignalR hub i metode koje prenose podatke do drugih klijenata

```
1. hubConnection = new
   HubConnectionBuilder().WithUrl(_navManager.ToAbsoluteUri("/signalRHub")).Build();
2. await hubConnection.StartAsync();
3. hubConnection.On<string, string, string>("ReceiveChatNotification", (message,
   receiverUserId, senderUserId) =>
4. {
5.     if (loggedInUser == Convert.ToInt32(receiverUserId))
6.     {
7.         Snackbar.Add(message, Severity.Info, config =>
8.         {
9.             config.VisibleStateDuration = 10000;
10.            config.HideTransitionDuration = 500;
11.            config.ShowTransitionDuration = 500;
12.            config.Action = "Chat?";
13.            config.ActionColor = MudBlazor.Color.Info;
14.            config.Onclick = snackbar =>
15.            {
16.                _navManager.NavigateTo($"chat/{senderUserId}");
17.                return Task.CompletedTask;
18.            };
19.        });
20.    } });
21. }
```

Nakon prihvaćenog zahtjeva za suradnju na projektu korisnik se dodaje na listu suradnika na pregledu pojedinog projekta. Vlasnik projekta, nakon što se prijavi željeni broj suradnika, može projekt prebaciti u stanje izrade čime se prijave zaključavaju te se suradnici upisuju na započeti projekt.

Kako bi se upotpunio ciklus, vlasnik projekta nakon završetka posla projekt može prebaciti u stanje završen i time označiti kraj projekta. Nakon završetka projekta svim članovima koji su sudjelovali u izradi projekta otvara se mogućnost ocjenjivanja svojih kolega. Na stranici projekta, ispod informacija o projektu, otvaraju se polja za unos komentara kao osvrta na suradnju i ocjenjivanje suradnika ocjenom od jedan do pet. Polja za ostavljanje osvrta mogu se vidjeti na slici 17.

Ostavite osvrt na suradnike!

Ostavite osvrt na suradnju s korisnikom [Pero Perić](#)

☆☆☆☆☆

SPREMI

Ostavite osvrt na suradnju s korisnikom [Marina Rajič](#)

☆☆☆☆☆

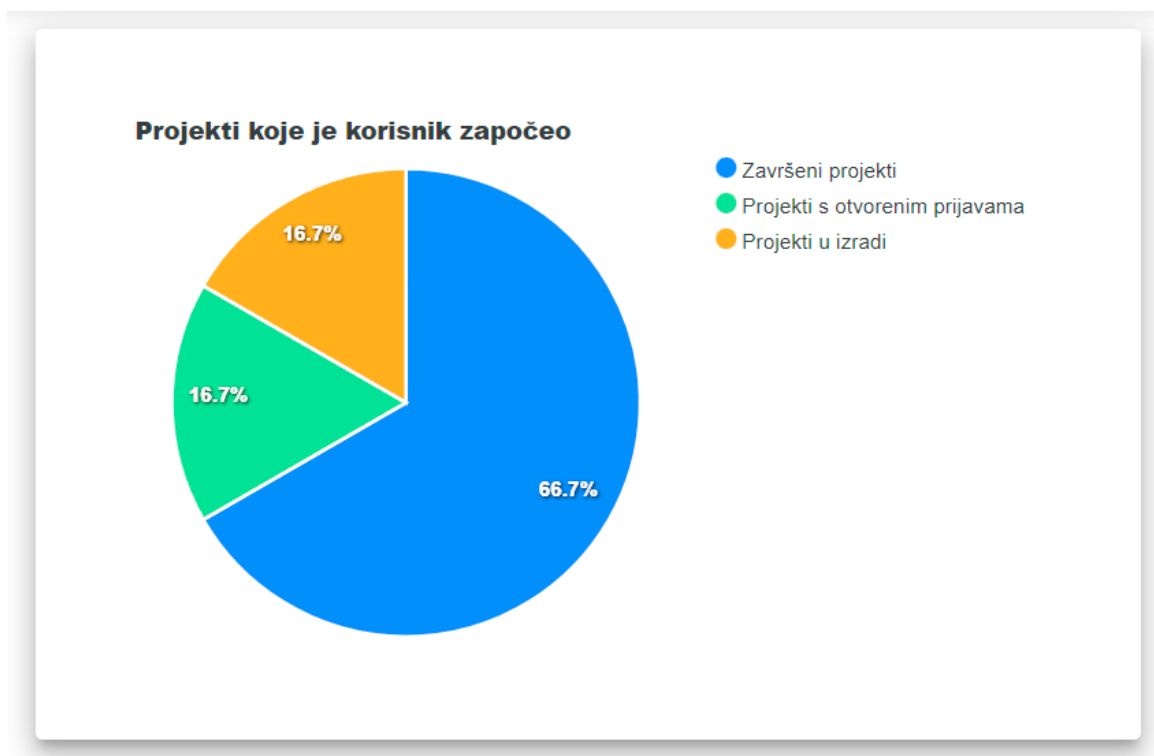
SPREMI

Slika 17. Polja za ostavljanje komentara i ocjene na suradnju

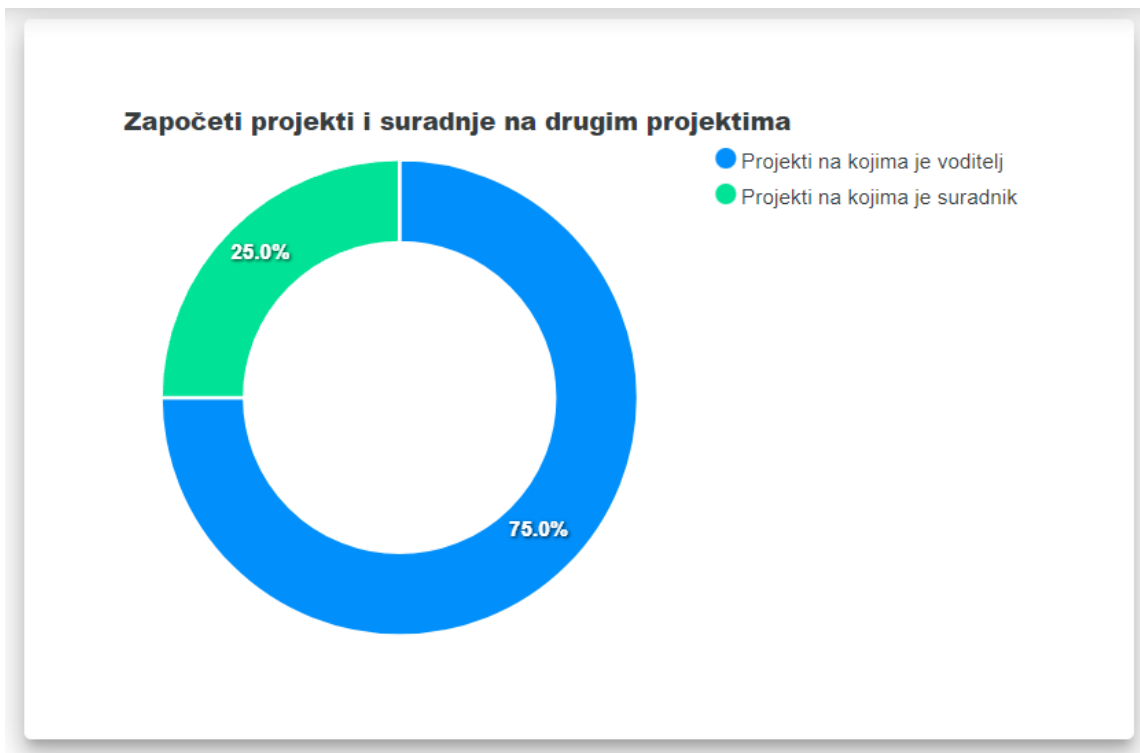
Osvrte koje korisnik primi prikazat će se na njegovom korisničkom računu te služe kao povratna informacija drugim korisnicima koji bi na budućim projektima sklopili suradnju s njim. Ocjene koje korisnik primi spremaju se uz komentar osvrta, a na profilu ispod imena prikazuje prosjek svih dodijeljenih ocjena.

Ispod podataka o korisničkom profilu i osvrta prikazuje se statistika o radu korisnika i njegovoj aktivnosti. U kružnom dijagramu prikazuje postotak projekata koji imaju otvorene

prijave, broj projekata koji su u izradi te projekti koje je on kao vlasnik završio. U prstenastom grafikonu prikazuje se postotak projekata u kojima je sudjelovao kao suradnik te ukupni broj projekata koje je započeo taj korisnik. Statistika o korisniku vidljiva je na slikama 18. i 19.



Slika 18. Kružni dijagram koji prikazuje postotak projekata koje je korisnik pokrenuo



Slika 19. Prstenasti dijagram u kojem je prikazan postotak projekata na kojima je korisnik voditelj i na kojima je suradnik

5. Zaključak

Tematika ovog rada bazira se na ideji da se programerima koji se samostalno odluče uključiti u tržište rada i pokrenuti projekt omogućí oglašavanje i pronalaženje odgovarajućih suradnika. Uz promjene koje dolaze razvojem tehnologije u smjeru povezanosti ili vanjskim utjecajima kao što je COVID virus sve više se ljudi koji posao mogu obavljati od kuće opredjeljuju za tu opciju. Svrha izrade ove aplikacije je približiti programere te im omogućíti pregled poslova koji se nude van organizacije neke kompanije i radnog mjesta koje je ograničeno vremenom i prostorn.

U ovom radu opisane su moderne tehnologije za razvoj web aplikacija. Microsoft je razvojem .NET platforme i Blazor programskog okvira unaprijedio način stvaranja web aplikacija te tako programerima olakšao ukupan proces od stvaranja ideje do implementiranja rješenja. Koristeći Web API i troslojnu arhitekturu na strani poslužitelja dobiva se strukturiran i dobro raspoređen kod čime se povećava preglednost same aplikacije, ali i mogućnosti za nadogradnje i proširivanje funkcionalnosti bez utjecaja na druge slojeve.

Aplikacija DevGuru svojim korisnicimanudi mogućnost objavljivanja svog projekta kojeg mogu pregledati svi korisnici aplikacije. Aplikacija na brz i jednostavan način vlasnicima projekata i potencijalnim suradnicima nudi sustav slanja prijave za suradnju na projektu, odobravanje ili odbijanje suradnje te sustav za komunikaciju kroz koji se suradnici mogu dogovoriti o detaljima suradnje na projektu. Vlasniku projekta omogućava se praćenje i prebacivanje stanja projekta, a nakon završetka izrade projekta otvara se mogućnost ostavljanja osvrta i ocjene na suradnju.

Kroz predmet Projektiranje informacijskih sustava stekao sam potrebna znanja u području REST Web API tehnologija koja su mi pomogla pri realizaciji aplikacije. Tokom razvoja ove aplikacije i pisanja rada usvojio sam znanja o pravilnom razvoju Blazor aplikacija, načinu komunikacije klijenta i poslužitelja preko kanala kao što su WebSocket i SignalR.

6. Popis slika

Slika 1. Dijagram Web API arhitekture [12]	6
Slika 2. Prikaz stvaranja DOM-a iz razor komponenti [13].....	9
Slika 3. Komunikacija Blazora i web preglednika u WebAssembly načinu izvođenja [9]	12
Slika 4. Komunikacija Blazora na poslužitelju i web preglednika [9]	13
Slika 5. Prikaz komunikacije servera sa klijentom putem SignalR-a [14]	14
Slika 6. Shema arhitekture aplikacije DevGuru	17
Slika 7. Prikaz relacija između projekata unutar Visual Studia.....	18
Slika 8. Arhitektura Blazor Server projekta	19
Slika 9. Arhitektura Blazor Server projekta	20
Slika 10. Shema baze podataka	21
Slika 11. Prikaz modala za ocjenjivanje vještine	23
Slika 12. Prikaz filtera za pretraživanje i kartica projekta.....	25
Slika 13. Prikaz forme za unos podataka novog projekta	26
Slika 14. Prikaz podataka pojedinog projekta	27
Slika 15. Prikaz obične poruke i zahtjeva za suradnju u chatu	27
Slika 16. Modal za slanje odgovora na zahtjev za suradnju.....	28
Slika 17. Polja za ostavljanje komentara i ocjene na suradnju.....	30
Slika 18. Kružni dijagram koji prikazuje postotak projekata koje je korisnik pokrenuo.....	31
Slika 19. Prstenasti dijagram u kojem je prikazan postotak projekata na kojima je korisnik voditelj i na kojima je suradnik.....	32

7. Popis literature

- [1] Microsoft, A tour of the C# language, <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>, Pristupljeno: 20. kolovoz 2023.
- [2] Joseph Albahari (2022), C# 10 in a Nutshell, O'Reilly Media, Inc., Dostupno na: <https://dl.ebooksworld.ir/books/CSharp.10.in.a.Nutshell.Joseph.Albahari.OReilly.9781098121952.EBooksWorld.ir.pdf>
- [3] Microsoft, Overview of .NET Framework, <https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview>, Pristupljeno: 20. kolovoz 2023.
- [4] Microsoft, Overview of ASP.NET Core, <https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0>, Pristupljeno: 23. kolovoz 2023.
- [5] Microsoft, Create web APIs with ASP.NET Core, https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-7.0&WT.mc_id=dotnet-35129-website, Pristupljeno: 24. kolovoz 2023.
- [6] Microsoft, Entity Framework Core, <https://learn.microsoft.com/en-us/ef/core/>, Pristupljeno: 24. kolovoz 2023.
- [7] Microsoft, Language Integrated Query (LINQ), <https://learn.microsoft.com/en-us/dotnet/csharp/linq/>, Pristupljeno: 24. kolovoz 2023.
- [8] Ed Charbeneau (2020), Blazor, A Beginners Guide, Dostupno na: <https://raw.githubusercontent.com/lauchacarro/ArchivosBlog/6382a38ed93968546342e8ae07c12d7db0ce41eb/files/blazor-a-beginners-guide4fc18478a1ee45c8a07b057eb79b584f.pdf>
- [9] Microsoft, ASP.NET Core Blazor, https://learn.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-7.0&WT.mc_id=dotnet-35129-website, Pristupljeno: 26. kolovoz 2023.,
- [10] Microsoft, Introduction to SignalR, <https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr>, Pristupljeno: 28. kolovoz 2023.
- [11] Microsoft, What is SQL Server Management Studio, <https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>, Pristupljeno: 28. kolovoz 2023.
- [12] Microsoft, Tutorial: Create a web API with ASP.NET Core, <https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio> , Pristupljeno 6. rujna 2023.

[13] Daniel Roth, Jeff Fritz, Taylor Southwick (2022.) : Blazor for ASP NET Web Forms Developers, <https://dotnet.microsoft.com/en-us/download/e-book/blazor-for-web-forms-devs/pdf> , Pristupljeno: 6. rujna 2023.

[14] <https://vivekcek.wordpress.com/2018/12/15/signalr-communication-in-multiple-projects-multiple-server-hubs-micro-services-architecture/> , Pristupljeno: 6. rujna 2023.



Veleučilište u Virovitici

OBRAZAC 5

IZJAVA O AUTORSTVU

Ja, Vatroslav Kopan

izjavljujem da sam autor/ica završnog/diplomskog rada pod nazivom

Web aplikacija za stvarivanje suradnje
samostalnih programera na projektima

Svojim vlastoručnim potpisom jamčim sljedeće:

- da je predani završni/diplomski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija,
- da su radovi i mišljenja drugih autora/ica, koje sam u svom radu koristio/la, jasno navedeni i označeni u tekstu te u popisu literature,
- da sam u radu poštivao/la pravila znanstvenog i akademskog rada.

Potpis studenta/ice

V. Kopan

OBRAZAC 6

ODOBRENJE ZA OBJAVLJIVANJE ZAVRŠNOG/DIPLOMSKOG RADA U
DIGITALNOM REPOZITORIJUJa Vatroslav Krpan

dajem odobrenje za objavljivanje mog autorskog završnog/diplomskog rada u javno dostupnom digitalnom repozitoriju Veleučilišta u Virovitici sadržanom u Dabar (Digitalni akademski arhivi i repozitoriji) te u javnoj internetskoj bazi završnih radova Nacionalne i sveučilišne knjižnice bez vremenskog ograničenja i novčane nadoknade, a u skladu s odredbama članka 58. stavka 5., odnosno članka 59. stavka 4. Zakona o visokom obrazovanju i znanstvenoj djelatnosti (NN 119/22).

Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog završnog/diplomskog rada. Ovom izjavom, kao autor navedenog rada dajem odobrenje i da se moj rad, bez naknade, trajno javno objavi i besplatno učini dostupnim na sljedeći način:

- a) Rad u otvorenom pristupu
- b) Rad dostupan nakon: _____ (upisati datum nakon kojeg želite da rad bude dostupan)
- c) Pristup svim korisnicima iz sustava znanosti i visokog obrazovanja RH
- d) Pristup korisnicima matične ustanove
- e) Rad nije dostupan (u slučaju potrebe dodatnog ograničavanja pristupa Vašem završnom/diplomskom radu, podnosi se pisani obrazloženi zahtjev).

Potpis studenta/ice

V KrpanU Virovitici, 7.9.2023.