

Izrada web aplikacije u programskim jezicima _____Sql , C#, Javascript , Html_____

Kratofil, **Roman Žarić**

Završni rad

2024

Ustanova izučavanja i vježbanja / stručni stupanj : Sveučilište **Edunova** ustanova za obrazovanje
odraslih , L.Jagera 5, 31000 Osijek

Permanent link / trajna poveznica : <https://www.edunova.hr>

Rights / Prava : <https://rightsstatements.org/page/InC/1.0/?language=en>



Edunova ustanova za obrazovanje odraslih

L.Jagera 5, 31000 Osijek

**Izrada web aplikacije u programskim jezicima
_____Sql , C#, Javascript , Html_____**

Završni rad

ROMAN ŽARIĆ

Osijek, 2024.

SADRŽAJ

1.	UVOD	1
1.1	Zadatak završnog rada	1
2.	Razvoj web aplikacije s naglaskom na korištene alate	2
2.1	Opis postupka razvoja aplikacije	2
Sl.2.1	Swagger	2
Sl.2.2	SQL Server Management Studio (SSMS).....	2
2.3	PROGRAMSKO RJEŠENJE IZGRADNJE TABLICA U SQL-u.....	2
2.3.1	ERA diagram	2
2.3.2	SQL scripta	2
3..0	C# - c sharp.....	3
3.1	Mogućnosti Microsoft okoline.....	3
3.1.1	Microsoft Visual Studio.....	3
Sl.3.1.2	Microsoft Visual Studio sučelje.....	3
3.2	Programski jezik C#.....	3
3.3	Uklanjanje problema s memorijom.....	3
3.4	Neke značajke C#.....	3
3.4.1	Tipovi podataka.....	3
Sl.3.4.1	Tipovi podataka.....	3
4.0	Aplikacija – skladišno poslovanje.....	3
4.1	Prikaz programskih komponenti.....	4
Sl. 4.2	Models.....	4
4.3	DTO	4
Sl. 4.3.1	DTO	4
4.3.2	Moj primjer koda je sljedeći	4
4.4	Extensio	4
Sl. 4.4.1	Extensions – Mapping.....	4
4.5	Mappers	4
4.6	DB context	4
Sl. 4.6.1	DbContext	4
4.7	Controllers	4
Sl. 4.7.1	Controller	4
5.0	Frontend-Visual StudioCode	5
Sl. 5.0.1	Visual Studio Code	5
5.1	Frontend aplikacija skladišno poslovanje.....	5
5.2	Service.js.....	5
5.3	Pages.....	5
Sl. 6.0	Aplikacija skladišno poslovanje osnovni prikaz	6

1. UVOD

U ovom završnom radu zadatak je izrada web aplikacije u programskim jezicima **MySQL, C#, Javascript, HTML**, te korištenje različitim alatima **Microsoft SQL Server Management Studio (SSMS), Visual studio, Visual studio code**.

Završni rad će se baviti konceptualnim i stvarnim rješenjima predviđenog zadatka. Korisniku je potrebno pružiti grafički prikaz tablica za unošenje raznih podataka vezanih za skladišno poslovanje. Aplikacija treba raditi sa bazom podataka, te iz nje dobivati i spremati podatke. To treba biti učinkovito i jednostavno na zadovoljstvo korisnika.

Također za funkcionalnost aplikacije bitna je mogućnost raditi pregleda, unošenja, promjene i brisanja podataka. Cilj ove aplikacije je korisnicima olakšati vođenje skladišnog inventara učinkovito i pregledno. Navedeni cilj postići će se koristeći aplikacijski okvir **ASP.NET MVC, Entity Framework** kao **Object Relational Mapper** te **SQL Server** kao **RDBMS** (Relational Database Management System). Što se prevodi kao "Sustav upravljanja relacijskim bazama podataka". To je vrsta sustava za upravljanje bazama podataka koja se temelji na relacijskom modelu podataka. Primjeri **RDBMS**-a uključuju **MySQL, PostgreSQL, Microsoft SQL Server, Oracle Database** i **SQLite**. Ovi sustavi omogućuju organizaciju podataka u tablicama koje su povezane relacijama, te pružaju mehanizme za upravljanje tim podacima putem **SQL**-a (**Structured Query Language**) i drugih alata.

U završnom radu nakon uvoda slijedi opis postupka razvoja aplikacije sa naglaskom na korištene alate u kojima su dodatno opisani mogućnosti i prednosti Microsoft okoline, programski jezik **C#** te njegove prednosti i nedostaci. Nakon toga slijedi detaljan opis korištenih tehnologija u kojem će se dodatno opisati **ASP.NET MVC** uz njegove tri glavne komponente - modele, preglede i kontrolore, zatim je dodatno opisan **Entity Framework** i **CSS** te njihov rad. Nakon toga slijedi programsko rješenje web aplikacije koje se sastoji od prikaza programskih komponenti i i primjera koda za pojedine dijelove aplikacije. Nastavno na programsko rješenje web aplikacije slijedi upotreba i testiranje aplikacije sadržana od načina upotrebe aplikacije i testiranja aplikacije.

1.1 Zadatak završnog rada

U završnom radu potrebno je savladati i opisati metodologiju izrade web aplikacije u **C#**, kao i ostale potrebne programske tehnologije i pristupe. U prvom redu, u radu će se koristiti aplikacijski okvir **ASP.NET MVC, Entity Framework** kao **Object Relational Mapper** te **SQL Server** kao **RDBMS**. Korištenjem navedenih tehnologija i pristupa, programski je potrebno ostvariti web aplikaciju i bazu podataka za odgovarajuću primjenu, a cjelovito rješenje prikladno ispitati i nalizirati. (mentor: **doc. dr. sc. Tomislav Jakopec**)

2. RAZVOJ WEB APLIKACIJE S NAGLASKOM NA KORIŠTENE ALATE

Kako je već u uvodu spomenuto, u izradi završnog rada u **Microsoft Visual Studiu 2017** korišteni su aplikacijski okviri **ASP.NET MVC**, **Entity Framework** kao **Object Relational Mapper** te **SQL Server** kao **RDBMS**. Uz navedeno korišten je i **Swagger** kako bi se moglo obaviti testiranje aplikacije.

2.1 Opis postupka razvoja aplikacije

Razvoj web aplikacije podrazumijeva sve potrebne postupke i alate za realizaciju traženog programskog rješenja. Navedeno se obično sastoji iz pisanja koda u nekom od razvojnih okruženja, kreiranja potrebne baze podataka, potrebnih modela podataka te testiranja same web aplikacije. U ovome završnom radu, postupak razvoj aplikacije se sastojao iz sljedećeg: ponajprije kreiranje projekta u **SQL Server Management Studio (SSMS)** kao odabranom razvojnom okruženju za kreiranja potrebne baze podataka. Stvaranje potrebnog modela podataka korištenjem **C#** koda, upotrebom **Visual studio code**, uporaba **Entity Frameworka** te kreiranje kontrolera za definiranje načina rada s podacima i pogleda za prikaz istih te konačno testiranje same aplikacije upotrebom **Swaggera** (<https://swagger.io/>).

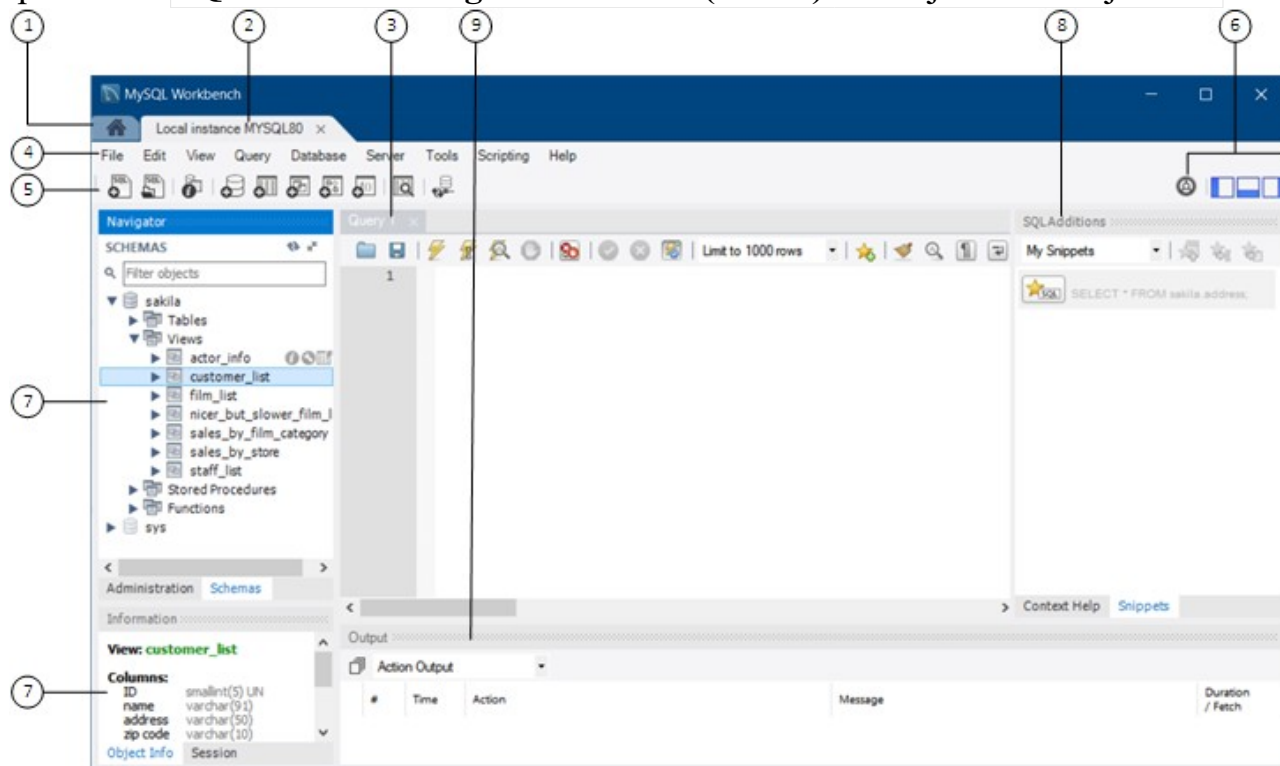
The screenshot displays the Swagger UI for the 'Skladiste API v1'. At the top, there's a header with the Swagger logo and a dropdown menu to 'Select a definition' showing 'SkladisteApi v1'. Below the header, the API title 'Skladiste API v1' is shown with 'v1' and 'OAS3' tags. A URL 'https://romanzanc-001-site1.itempuri.com/swagger/v1/swagger.json' is listed. A brief description 'Ovo je dokumentacija za Skladiste API' and contact information 'Contact Roman Zanic' and 'Skladisno poslovanje' are provided. The main content area lists endpoints for the 'Izdatnica' resource, which is described as 'Namijenjeno za CRUD operacije nad entitetom izdatnice u bazi'. The endpoints are:

- GET** `/api/v1/Izdatnica`: Prikazuje sve izdatnice iz baze
- POST** `/api/v1/Izdatnica`: Dodaje novu izdatnicu u bazu
- GET** `/api/v1/Izdatnica/{sifra}`: Dohvaća sve sifru iz baze
- PUT** `/api/v1/Izdatnica/{sifra}`
- DELETE** `/api/v1/Izdatnica/{sifra}`: Briše izdatnicu iz baze
- GET** `/api/v1/Izdatnica/Proizvodi/{sifraGrupe}`
- POST** `/api/v1/Izdatnica/{sifra}/dodaj/{proizvodSifra}`
- DELETE** `/api/v1/Izdatnica/{sifra}/obrisi/{proizvodSifra}`

At the bottom, the 'Osoba' resource is partially visible, described as 'Namijenjeno za CRUD operacije nad entitetom osoba u bazi'.

Sl.2.1 Swagger

Kreiranje baze podataka podrazumjeva izradu tablica u programskom rješenju upotrebom **SQL Server Management Studio (SSMS)**. Razvojno okruženje nudi



Microsoft

Sl 2.2 **SQL Server Management Studio (SSMS)**

možnosti izrade tablica(entiteta), dodavanje atributa, njihovo povezivanje, te dodavanje vrijednosti atributima kako bi ih mogli testirati kako u samoj bazi podataka tako i u daljnjem radu na **Visual studiu** i **Vizual studio codu**, korištenjem **Swagger**.

Za sliku korišten link **SQL Server Management Studio (SSMS)**
<https://dev.mysql.com/doc/workbench/en/wb-sql-editor.html>

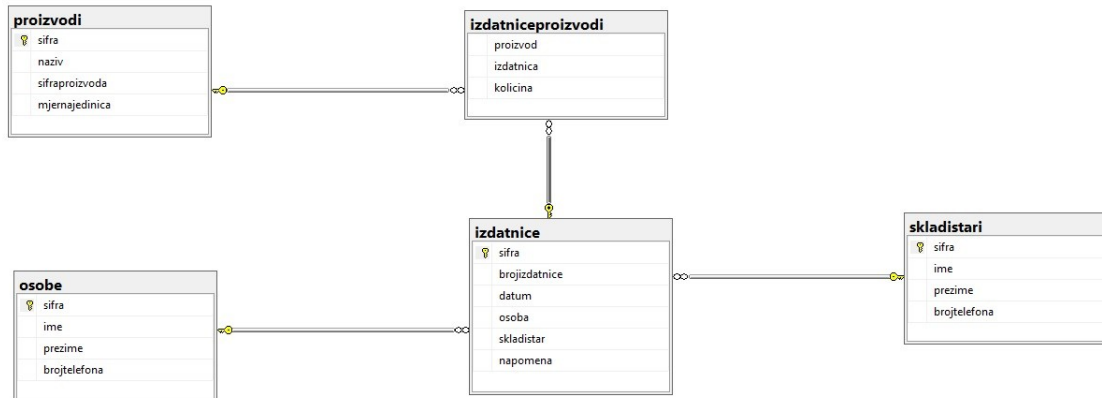
2.2 PROGRAMSKO RJEŠENJE IZGRADNJE TABLICA U SQL-u

Definiranje potrebnih tablica : prvo se identificiraju entiteti koji se žele modelirati u bazi podataka .Što je u mom slučaju 4 entiteta. Svaki entitet odgovara jednoj tablici u bazi podataka.

Definiranje stupaca : Za svaku tablicu su potrebni stupci koji će pohraniti podatke o entitetu. Svaki stupac treba imati naziv i tip podataka.

Definiranje ključeva : Odredio primarni ključ (Sifra) za svaku tablicu. Primarni ključ jedinstveno identificira svaki redak u tablici. Također se mogu definirati i

vanjski ključevi (Foreign Keys) koji uspostavljaju veze između tablica. U mom slučaju postoje 2 vanjska ključa jedan na više, te 1 ključ više na više. kao što je prikazano na slici **2.3 ERA diagram , entitet osobe i entitet skladistari .**



2.3 ERA diagram

Definiranje ograničenja integriteta podataka: Još se mogu postaviti ograničenja integriteta podataka kako bi se osigurali ispravnost i dosljednost podataka. Na primjer, može postaviti ograničenja za provjeru jedinstvenosti vrijednosti u određenom stupcu ili za provjeru referencijalne cjelovitosti između tablica. U mom slučaju to nisam upotrijebio.

zgradnja SQL skripte: Na temelju definicija tablica i njihovih stupaca, napisao sam SQL skriptu koja će stvoriti tablice u bazi podataka. U skripti sam koristio ključne riječi poput CREATE TABLE, ALTER TABLE, ADD CONSTRAINT i slično kako bi definirao strukture tablica. U mom slučaju je kao što se vidi iz priložene skripte. Integrirano 2 vanjska ključa (osobe, skladištari) , koji se spajaju na tablicu „Izdatnice”, što su ključevi jedan na više, te sam napravio i tablicu (Proizvodi)koja ima ključ više na više prema tablici „Izdatnice” .Kako bih mogao uspostaviti tu vezu , napravio sam među tablicu „IzdatniceProizvodi” koja simulira učinak tablica jedan naprama više.

2.4 SQL scripta :

```
create table proizvodi(
```

```
sifra int not null primary key identity(1,1),  
naziv varchar(50) not null,  
sifraproizvoda varchar(50) ,  
mjernajedinica varchar(20) not null  
);
```

```
create table skladistari(
```

```
sifra int not null primary key identity(1,1),  
ime varchar(50) not null,  
prezime varchar(50) not null,  
email varchar(50),  
brojtelefona varchar(20)  
);
```

```
create table osobe(
```

```
sifra int not null primary key identity(1,1),  
ime varchar(50) not null,  
prezime varchar(50) not null,  
email varchar(50),  
brojtelefona varchar(20)  
);
```

```
create table izdatnice(
```

```
sifra int not null primary key identity(1,1),  
brojizdatnice varchar (50) not null,  
datum datetime,  
osoba int not null references osobe(sifra),  
skladistar int not null references skladistari(sifra),
```



```
napomena varchar(250)
);
```

```
create table izdatniceproizvodi (
```

```
    proizvod int not null references proizvodi(sifra),
    izdatnica int not null references izdatnice(sifra),
    kolicina int
);
```

3..0 C# - c sharp

C# je moderni, objektno orijentirani programski jezik razvijen od strane Microsofta. Lanziran je 2000. godine kao ključni jezik u okviru Microsoftove platforme .NET. Od tada, C# je postao jedan od najpopularnijih programskih jezika zbog svoje jednostavnosti, snažnih alata i širokog spektra primjene.

Evo nekoliko ključnih značajki i koncepta u vezi s C#:

1. **Objektno orijentirano programiranje (OOP)**: C# je potpuno objektno orijentiran jezik, što znači da sve u C#-u, uključujući i same tipove podataka, predstavljeno je kao objekt. OOP koncepti kao što su nasljeđivanje, polimorfizam i apstrakcija ključni su dijelovi C#-a.
2. **Tipiziranost na razini stroja**: C# je statički tipiziran jezik, što znači da tipovi varijabli moraju biti određeni pri kompiliranju i ne mogu se mijenjati tijekom izvođenja programa. Ovo pruža veću sigurnost i olakšava otkrivanje grešaka u kodu pri kompiliranju.
3. **.NET platforma**: C# je tijesno povezan s .NET platformom, koja pruža ogroman okvir klasa i biblioteka za razvoj različitih vrsta aplikacija. .NET omogućuje izradu aplikacija za različite platforme, uključujući Windows, Linux i macOS.
4. **Lambda izrazi i LINQ**: C# podržava lambda izraze, što omogućuje definiranje anonimnih funkcija. Također, C# ima ugrađeni jezik upita LINQ (Language Integrated Query), koji omogućuje izvođenje upita nad kolekcijama objekata na način sličan SQL-u.
5. **Async/Await programiranje**: C# podržava asinkrono programiranje putem ključnih riječi `async` i `await`, što omogućuje izvođenje dugotrajnih operacija bez blokiranja glavne niti izvođenja.

6. **Razvoj web aplikacija**: C# se često koristi za razvoj web aplikacija putem tehnologija poput ASP.NET Core-a, koji omogućuje izradu modernih, skalabilnih i sigurnih web aplikacija.
7. **Windows aplikacije**: C# je popularan izbor za razvoj Windows aplikacija putem tehnologija kao što su Windows Forms i WPF (Windows Presentation Foundation).
8. **Razvoj igara**: C# se koristi i za razvoj igara, posebno u kombinaciji s platformama kao što su Unity i Xamarin.

Kreiranje projekta podrazumijeva stvaranje novog programskog rješenja u razvojnom okruženju **Microsoft Visual Studio** kao **ASP.NET MVC** web aplikacije. Dakle, nakon stvorenih klasa (modela) uz pomoć Entity Frameworka. Sljedeći korak je kreiranje kontrolera – pisanjem **C#** koda stvaraju se dijelovi rješenja (datoteke) koje sadrže i opisuju potrebne akcije za upravljanje podacima (dodavanje unosa, izmjena, brisanje, prikaz detalja i sl.) kao i prikaz istih. Razvojno okruženje, pri tome, automatski generira i potrebne prikaze (pogleda) ovisno o odabranom kontroleru. Način rada modela, pogleda i kontrolera, odnosno princip rada **MVC** aplikacije opisan je kasnije u radu.

Nakon kreiranih svih potrebnih datoteka, aplikacija je realizirana te je posljedni korak u postupku razvoja upravo testiranje same aplikacije. Uz unesene vrijednosti u bazu podataka, testiraju se sve moguće funkcionalnosti koje bi aplikacija trebala obavljati, traže se moguće greške i izuzeci koje je potrebno korigirati i ispraviti. Nakon obavljenog testiranja, aplikacija je uspješno stvorena te spremna za korištenje.

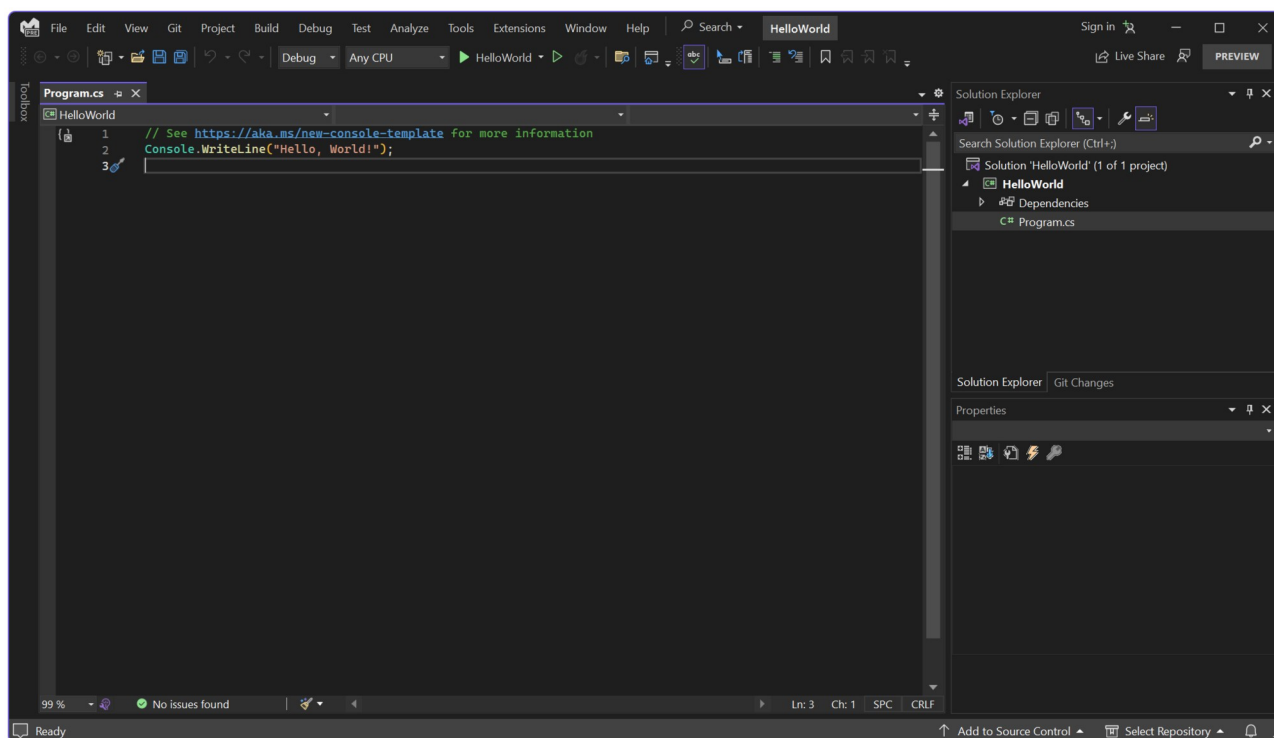
3.1 Mogućnosti Microsoft okoline

Može se reći kako Microsoft nudi mnoge alate i tehnologije koji služe korisnicima u izradama aplikacija. U posljednje vrijeme Microsoft se puno više trudi izdavati vodiče za korištenje njihovih proizvoda i razvijati svoje alate u skladu sa potrebama i sugestijama mnogih programera. Microsoft redovito izdaje vodiče u obliku videa koji detaljno opisuju sve mogućnosti, izmjenu sadržaja te nove mogućnosti u novijim inačicama alata koje izdaju

Microsoft je razvio **Visual Studio** kao proizvod idealan za razvojne programere kako bi im omogućio izrađivanje mnogih vrsta aplikacija. Prema [1], Microsoft **Visual Studio** koristi se za izradu *desktop aplikacija*, *web aplikacija* ili *mobilnih aplikacija*. To je ujedno i jedan od najraširanijih alata za izradu aplikacija na Windows operacijskim sustavima. Prednost **Visual Studia** je što je dosta fleksibilan i omogućuje korisnicima jednostavan razvoj mobilnih aplikacija, ali on zahtjeva puno

više vremena za savladavanje svih njegovih mogućnosti. Postoje tri glavne inačice **Visual Studio** integriranog razvojnog okruženja: **Visual Studio**, **Visual Studio Code** i **Visual Studio Online**. Kako je navedeno u [2], također postoji više inačica programa: **Visual Studio Community** (besplatna inačica dostupna za korištenje za akademske svrhe ili individualni razvoj), **Visual Studio Professional** (plaćena inačica s raznim korisnim alatima za razvoj namijenjena manjim timovima ili pojedinačnim developerima) te **Visual Studio Enterprise** (plaćena inačica za poduzeća i timove bilo koje veličine s naprednim alatima za razvoj i najkompleksnijih rješenja). Cilj svake inačice Visual Studia je pružiti bogato razvojno okruženje Microsoft je razvio **Visual Studio** kao proizvod idealan za razvojne programere kako bi im omogućio izrađivanje mnogih vrsta aplikacija. Prema [1], **Microsoft Visual Studio** koristi se za izradu *desktop* aplikacija, web aplikacija ili mobilnih aplikacija. To je ujedno i jedan od najraširanijih alata za izradu aplikacija na Windows operacijskim sustavima. Prednost **Visual Studia** je što je dosta fleksibilan i omogućuje korisnicima jednostavan razvoj mobilnih aplikacija, ali on zahtjeva puno više vremena za savladavanje svih njegovih mogućnosti.

3.1.1 Microsoft Visual Studio



Microsoft

Sl.3.1.1 *Microsoft Visual Studio sučelje*

Postoje tri glavne inačice **Visual Studio** integriranog razvojnog okruženja: **Visual Studio**, **Visual Studio Code** i **Visual Studio Online**. Kako je navedeno u [2],

također postoji više inačica programa: **Visual Studio Community** (besplatna inačica dostupna za korištenje za akademske svrhe ili individualni razvoj), **Visual Studio Professional** (plaćena inačica s raznim korisnim alatima za razvoj namijenjena manjim timovima ili pojedinačnim developerima) te **Visual Studio Enterprise** (plaćena inačica za poduzeća i timove bilo koje veličine s naprednim alatima za razvoj i najkompleksnijih rješenja). Cilj svake inačice Visual Studia je pružiti bogato razvojno okruženje svim programerima na globalnoj razini na bilo kojoj platformi. **Visual Studio** nudi bogat izbor razvojnih jezika. Trenutno programeri mogu razvijati aplikacije u jezicima **Visual Basic**, **C #**, **PHP**, **Objektni-C**, **JavaScript** i **Visual C + +**. Aplikacijsko sučelje za programiranje na kojem se temelji razvoj tvrtke Microsoft zove se **.NET Framework** i pruža podršku za interoperabilnost jezika.

Microsoft **.NET** okvir je programski okvir koji pruža veliku zbirku kodiranih sklopova koje programeri mogu koristiti za lakše razvijanje programa. On nadograđuje operacijski sustav kako bi olakšao rad programerima koji razvijaju aplikacije u dotičnoj tehnologiji. Najvažniji dio te infrastrukture je **CLR**, to je programski dio u kojem se pisani kod izvršava, on prevodi program u jezik razumljiv kompjuter i na taj način se **.Net** okvir može prenositi i na druge operacijske sustave osim Microsoftovih, odnosno svaki program koji je napisan u jezicima koje podržava Microsoft. Neki primjeri od tih jezika su **C++,C#,VB.Net,Jscript,J#**. Pojednostavljeno, **.NET** je razvojni okvir s kojim imamo mogućnosti novog sučelja za programiranje aplikacija te objedinjuje klasična sučelja operacijskog sustava Windows s mnogim tehnologijama koje je izdao

3.2 Programski jezik C#

Andres Hejlsberg - zajedno sa svojim timom 1999. godine napravio je programski jezik **C#**. Početni naziv za **C#** bio je „Cool“, što je značilo **C-like Object Oriented Language**. Nakon godinu dana naziv se promjenio u **C#**. **C#** sintaksa je zapravo dosta slična sintaksi programskog jezika **C++**. Napomenit ćemo da je **C#** modernizirana inačica **C++**. Izvorno je napravljen programski jezik **C**, koji je imao široku primjenu u programiranju. Nakon njega **C++** je postao jezik za izgradnju cijelovitih aplikacija za Windows operacijski sustav. **C++** se koristio za pisanje infrastrukture i aplikacija nižih razina, dok su programeri u programskom jeziku **Visual Basic** razvijali poslovne aplikacije. **C#** koristi prednosti **.NET** okvira, što znači da imate pristup brojnim aplikacijskim okvirima, baš kao i **Visual Basic** razvojni programeri godinama. Dodane su nove vrste podataka, kao što je decimalna vrsta podataka za obavljanje financijskih kalkulacija što dodatno olakšava dio posla. **C#** je siguran jezik, što znači nekoliko stvari. Na primjer, ne može se upotrebljavati neinicijalizirane varijable. U **C++** lako je proglasiti varijablu, a zatim provjeriti njegovu vrijednost; Onda bi se prikazalo ono što je bilo u memorijskoj adresi koja se daje tim varijablama, što bi moglo onemogućiti cijelu aplikaciju za njezino ispravno

korištenje. C# - ov compiler će vas obavijestiti ako pokušate upotrijebiti varijablu prije nego što ga inicijalizirate na neku ispravnu vrijednost. Važno je objektna orijentiranost kod C# jezika, za razliku od C++ jezika, daleko više istaknuta.

3.3 Uklanjanje problema s memorijom

C# uklanja probleme za upravljanje memorijom od strane programera pomoću sheme za sakupljanje smeća .NET-a. Stavke koje se više ne spominju označene su za sakupljanje smeća, a Okvir može po potrebi vratiti ovu memoriju. C# također podržava uvođenje XML komentara. To nije samo još jedan način za dodavanje komentara u kod, XML komentari se mogu zapravo pretvoriti u vlastitu dokumentaciju. Komentari se postavljaju u XML format i mogu se upotrijebiti prema potrebi za dokumentiranje koda. Ova dokumentacija može uključivati primjer koda, parametre i reference na druge teme. Napokon ima smisla da programer dokumentira svoj kod, jer ti komentari mogu postati dokumentacija neovisno o izvornom kodu. C# više nije namjenjen samo za korisnike Microsofta. Microsoft je izdao C# ECMA-u (European Computer Manufacturers Association). Također se koristi kao skraćunica za European Computer Manufacturers Association Script (ECMAScript), što je standardizacija skriptnog jezika koji se koristi za web razvoj, a poznat je kao JavaScript. i objavio ga je u osnovnoj inačici. Osim toga, daljnji Mono projekti nastoje napraviti otvorenu i prilagodljivu inačicu .NET okvira za druge operacijske sustave, čak i za Linux. Sponzoriran od strane Microsofta, Mono je otvorena i prilagodljiva implementacija Microsoftovog .NET okvira na temelju ECMA standarda za C# i Common Language Runtime.

3.4 Neke značajke C#

3.4.1 Tipovi podataka

Tipovi podataka koje koristi C# identični su tipovima korištenim u C i C++ jezicima. U idućoj tablici (Tab. 2.1.) možemo vidjeti neke od najčešće korištenih tipova podataka, njihovu vrstu te veličinu memoriju koju podatak zauzima

char	Jedan karakter	Bilo koji karakter, npr. a,*, \x0058 (hex), ili\u0058 (Unicode)
string	Više karaktera	Niz unikod karaktera.
int	Cijeli broj	−9.223.372.036.854.775.808 do 9.223.372.036.854.775.807
float	Cijeli broj jednostruke	−3.4·10 na−38 do

	preciznosti	$3.4 \cdot 10$ na 38
double	Cijeli broj dvostruke preciznosti	$-1.7 \cdot 10$ na -308 do $1.7 \cdot 10$ na 308
decimal	Decimalni broj	(+ ili -) $1.0 \times 10e-28$ do $7.9 \times 10e28$
bool	Logički podatak	True,false

Sl.3.4.1 Tipovi podataka

C# posuđuje koncepte Jave i C++, usvajajući samo dobre dijelove tih jezika i uklanjajući prekomjerno zbunjujuće i pogrešne značajke koje su glavni izvori grešaka u kodu.

NET Framework omogućava je upotrebu više jezika unutar istog rješenja, kao i korištenje dodataka pisanih u drugim alatima sa podrškom za .NET.

C# implementira suvremeni kencept objektno orijentiranog programiranja koji omogućuje razvojnom programeru da izrađuje sigurne aplikacije koje pružaju korisniku sljedeću razinu iskustva.

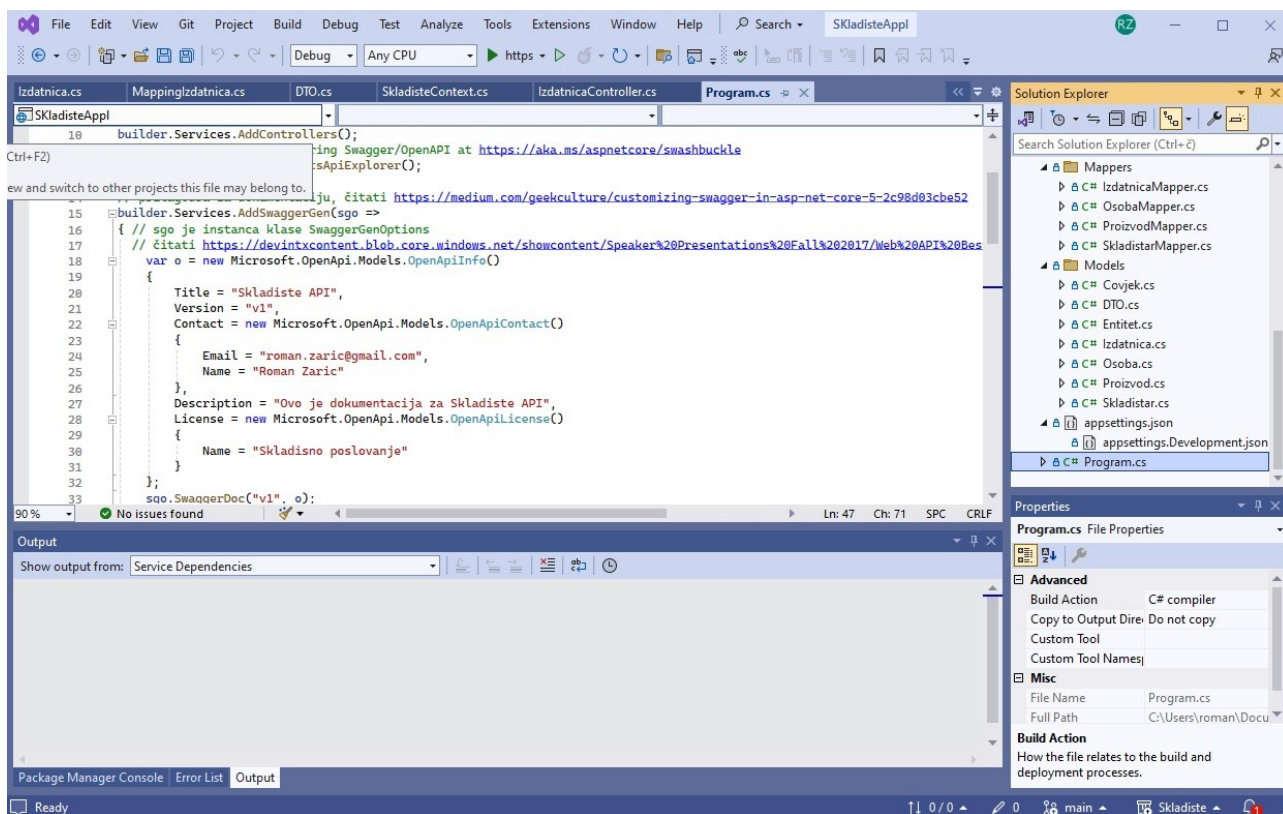
C# se može koristiti za pisanje širokog raspona aplikacija, od jednostavnih desktop widgeta do razvoja raznih web usluga, programiranje sigurnosnih sustava, pa čak i robota.

C# se koristi za web programiranje i sa svakodnevnim novim informacijama donosi raspodijeljenu razmjenu informacija, tako da sve što korisnik treba je računalo i preglednik.

4.0 Aplikacija – skladišno poslovanje

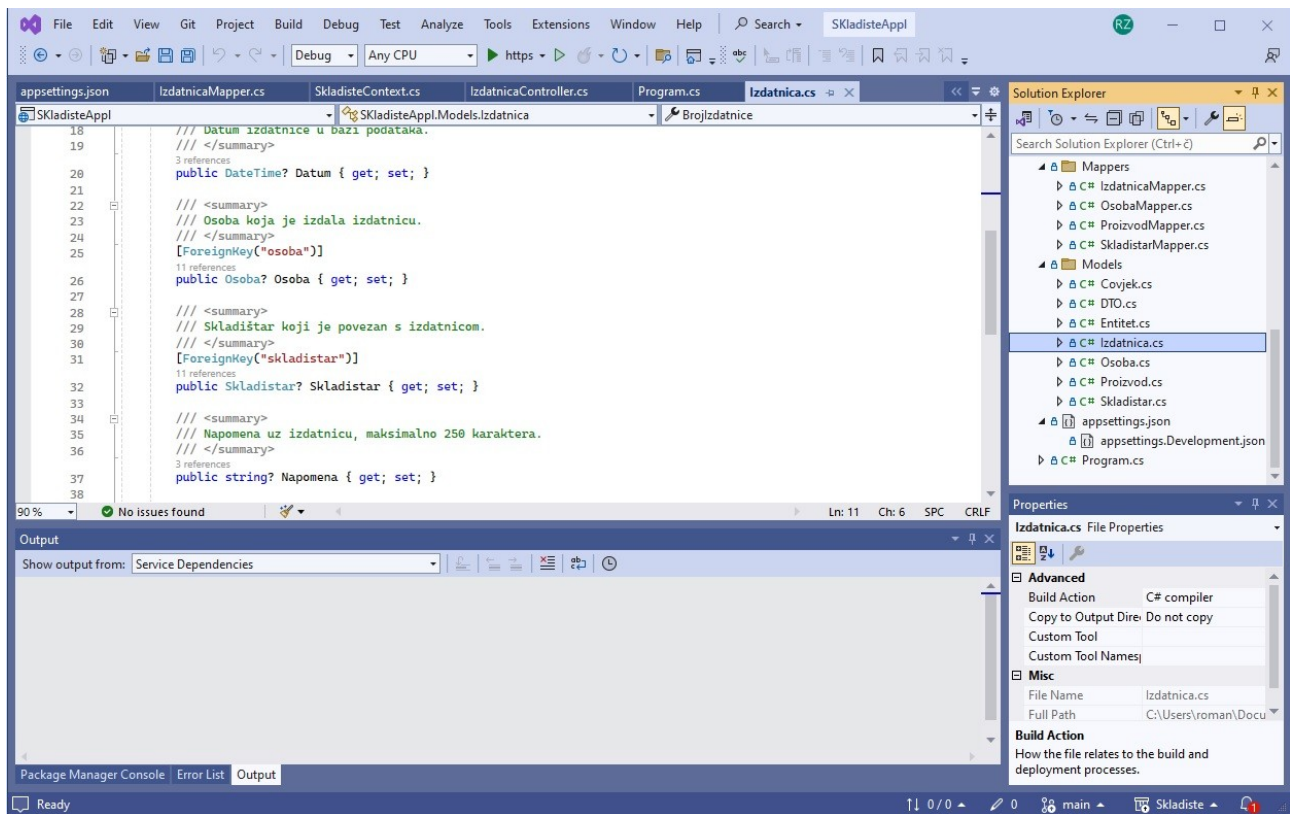
4.1 Prikaz programskih komponenti

Aplikacija (skldišno poslovanje) sastoji se od više datoteka Sl. 41.1 . Izdvojio bi Dependencies (ovisnosti o NET frameworks), wwwroot (frontend),Controllers,Data(data context),Mapping,Mapper,Models u kojem je ubačen DTO (data transfer object),json ...



Sl. 4.1 radna okolina

Models : Najprije sam ispisao modele s potrebnim podacima .Dakle Nazivi modela kao entiteta , te sam upisao attribute te properties za svaki atribut . Ujedno sam opisao koji su atributi obavezni , te sam upisao vanjske kljuceve (ForeignKey) . Napravio sam i model Entitet kako bih upravljao s jedinstvenim kljucem svakog entiteta (sifra). Jednako tako upogonio u modelu „Izdatnice” kljuc više na više u odnosu na entitet „Proizvodi” , public List<Proizvod>? Proizvodi { get; set; }.Takodjer sam ubacio i DTO .



Sl. 4.2 Models

4.3 DTO :

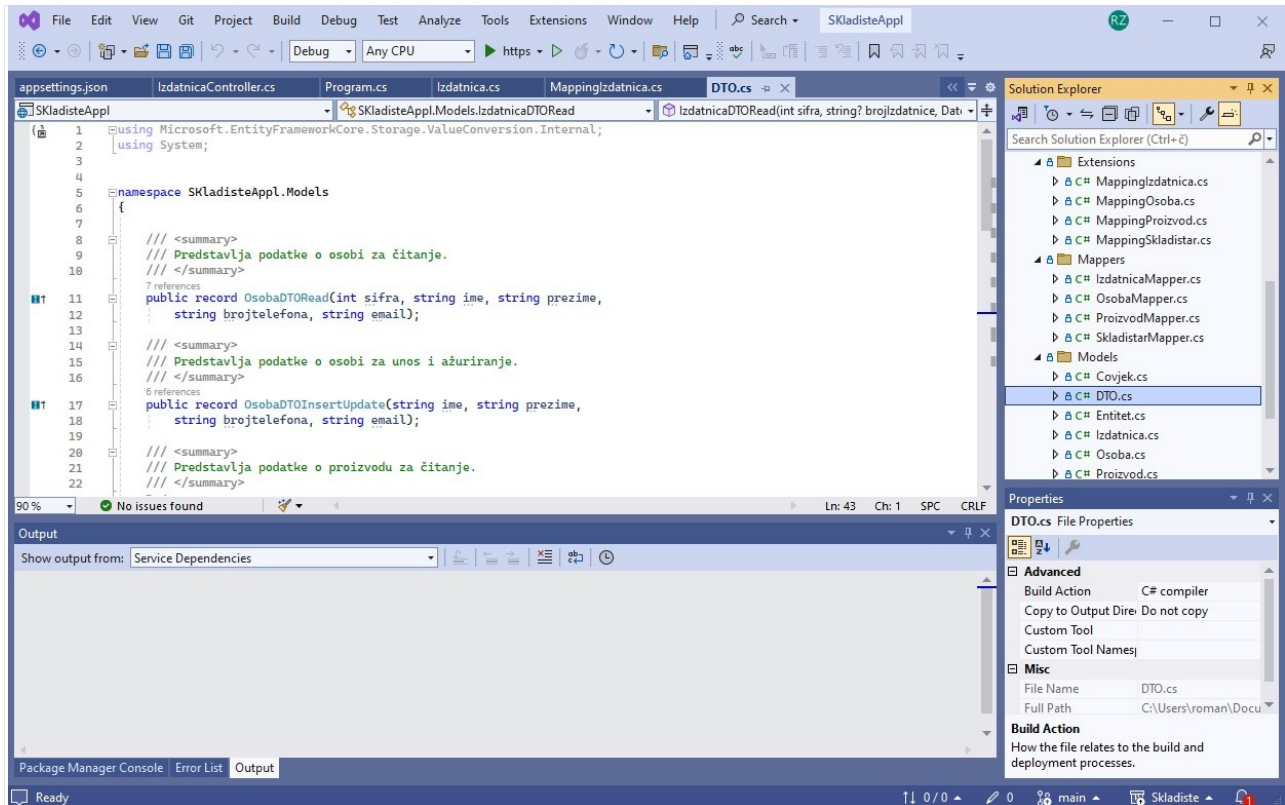
DTO (Data Transfer Object) je obrazac dizajna softvera koji se koristi za prijenos podataka između slojeva aplikacije ili komponenti. Osnovna svrha DTO-a je olakšati prijenos podataka tako što omogućava grupiranje više atributa u jedan objekt ili strukturu podataka.

Evo nekoliko ključnih karakteristika DTO-a:

1. **Jednostavnost**: DTO-ovi su jednostavni objekti koji sadrže samo podatke, bez logike ili ponašanja. Obično nemaju metode osim gettera i settera za pristup i postavljanje vrijednosti atributa.
2. **Prijenos podataka**: DTO-ovi se koriste za prijenos podataka između različitih dijelova aplikacije, kao što su kontroleri, servisi, repozitoriji ili klijentski programi. Na primjer, mogu se koristiti za prenošenje podataka iz baze podataka do korisničkog sučelja ili obrnuto.
3. **Neovisnost o slojevima**: DTO-ovi omogućuju neovisnost između slojeva aplikacije tako što omogućuju komunikaciju između slojeva pomoću jednostavnih objekata podataka. To znači da se promjene u jednom sloju ne bi trebale utjecati na druge slojeve, sve dok se struktura DTO-a ne mijenja.
4. **Optimizacija prijenosa podataka**: DTO-ovi se često koriste za optimizaciju prijenosa podataka između aplikacije i baze podataka ili između mikroservisa u arhitekturi temeljenoj

na mikroservisima. Umjesto slanja velikog broja pojedinačnih atributa, možete poslati samo jedan objekt DTO-a s svim potrebnim podacima.

5. **Prilagodljivost**: DTO-ovi se mogu prilagoditi specifičnim zahtjevima aplikacije ili komunikacijskim protokolima. Na primjer, možete stvoriti različite vrste DTO-ova za različite operacije ili verzije API-ja.



Sl. 4.3.1 DTO

Moj primjer koda je sljedeći :

```
using Microsoft.EntityFrameworkCore.Storage.ValueConversion.Internal;
```

```
using System;
```

```
namespace SKladisteAppl.Models
```

```
{
```

```
    /// <summary>
```

```
    /// Predstavlja podatke o osobi za čitanje.
```

```
/// </summary>
```

```
public record OsobaDTORead(int sifra, string ime, string prezime,  
    string brojtelefona, string email);
```

```
/// <summary>
```

```
/// Predstavlja podatke o osobi za unos i ažuriranje.
```

```
/// </summary>
```

```
public record OsobaDTOInsertUpdate(string ime, string prezime,  
    string brojtelefona, string email);
```

```
/// <summary>
```

```
/// Predstavlja podatke o proizvodu za čitanje.
```

```
/// </summary>
```

```
public record ProizvodDTORead(int sifra, string naziv, string sifraproizvoda,  
    string mjernajedinica);
```

```
/// <summary>
```

```
/// Predstavlja podatke o proizvodu za unos i ažuriranje.
```

```
/// </summary>
```

```
public record ProizvodDTOInsertUpdate(string naziv, string sifraproizvoda,  
    string mjernajedinica);
```

```
/// <summary>
```

```
/// Predstavlja podatke o skladištaru za čitanje.
```

```
/// </summary>
```

```
public record SkladistarDTORead(int? sifra, string ime, string prezime,  
    string brojtelefona, string email);
```

```
/// <summary>
```

```
/// Predstavlja podatke o skladištaru za unos i ažuriranje.
```

```

/// </summary>
public record SkladistarDTOInsertUpdate(string ime, string prezime,
    string brojtelefona, string email);

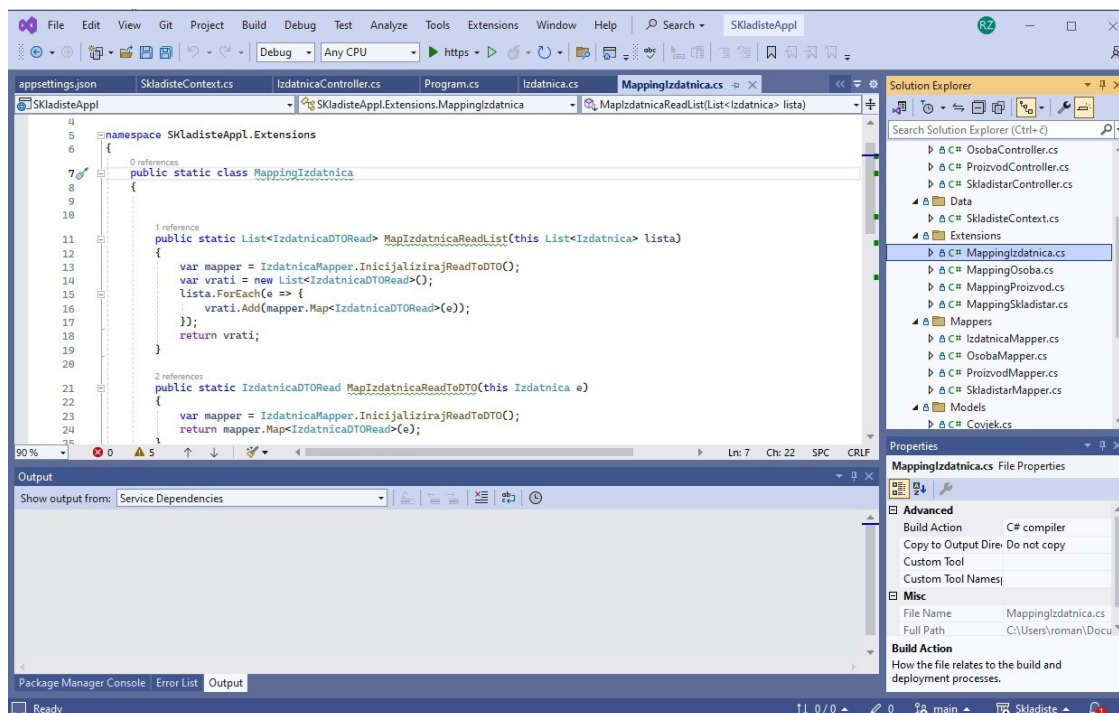
/// <summary>
/// Predstavlja podatke o izdatnici za čitanje.
/// </summary>
public record IzdatnicaDTORead(int sifra, string? brojIzdatnice,
    DateTime? datum, string? osobaImePrezime, string? skladistarImePrezime,
    string? proizvodNaziv, string napomena);

/// <summary>
/// Predstavlja podatke o izdatnici za unos i ažuriranje.
/// </summary>
public record IzdatnicaDTOInsertUpdate(string? brojizdatnice,
    DateTime? datum, int? osobasifra, int? skladistarSifra, string napomena);
}

```

4.4 Extensions :

U ekstenzije sam postavio mapping koji će upravljat s DTO podacima . Dakle sa read , insert , update to , update from . Upravlja s postavljenim metodama koje daju potrebne podatke o pozivanju , unošenju , povezivanju entiteta s atributima .



Sl. 4.4.1 Extensions – Mapping

4.5 Mappers :

Mapirao sam sve entitete i atribute kako bih mogao to sve upogoniti putem controllera. Mapper je komponenta softverskog sustava koja se koristi za mapiranje ili transformaciju podataka iz jednog oblika u drugi. U kontekstu softverskog razvoja, mapper se obično koristi za preslikavanje podataka iz jednog objekta ili strukture podataka u drugi objekt ili strukturu podataka, često u različitim slojevima ili formatima aplikacije.

Ključne značajke mappera uključuju:

1. **Preslikavanje atributa**: Mapper omogućuje preslikavanje pojedinačnih atributa ili polja iz jednog objekta u drugi objekt. Ovo je korisno kada imate objekte slične strukture, ali različite tipove ili potpuno različite strukture.
2. **Transformacija podataka**: Mapper omogućuje transformaciju ili manipulaciju podataka tijekom procesa preslikavanja. Na primjer, možete konvertirati tipove podataka, filtrirati podatke ili kombinirati više atributa u jedan.
3. **Neovisnost slojeva**: Mapper omogućuje neovisnost između slojeva aplikacije tako što omogućuje komunikaciju između slojeva pomoću jednostavnih objekata podataka. To znači da promjene u jednom sloju ne bi trebale utjecati na druge slojeve, sve dok se struktura podataka održava.

4. **Prilagodljivost**: Mapper se može prilagoditi specifičnim zahtjevima aplikacije ili transformacijama podataka. Možete definirati pravila preslikavanja koja odgovaraju vašim potrebama i zahtjevima aplikacije.
5. **Optimizacija performansi**: Učinkoviti mapper može poboljšati performanse aplikacije optimizirajući proces preslikavanja podataka. To može uključivati caching preslikanih podataka ili korištenje optimiziranih algoritama za transformaciju podataka.

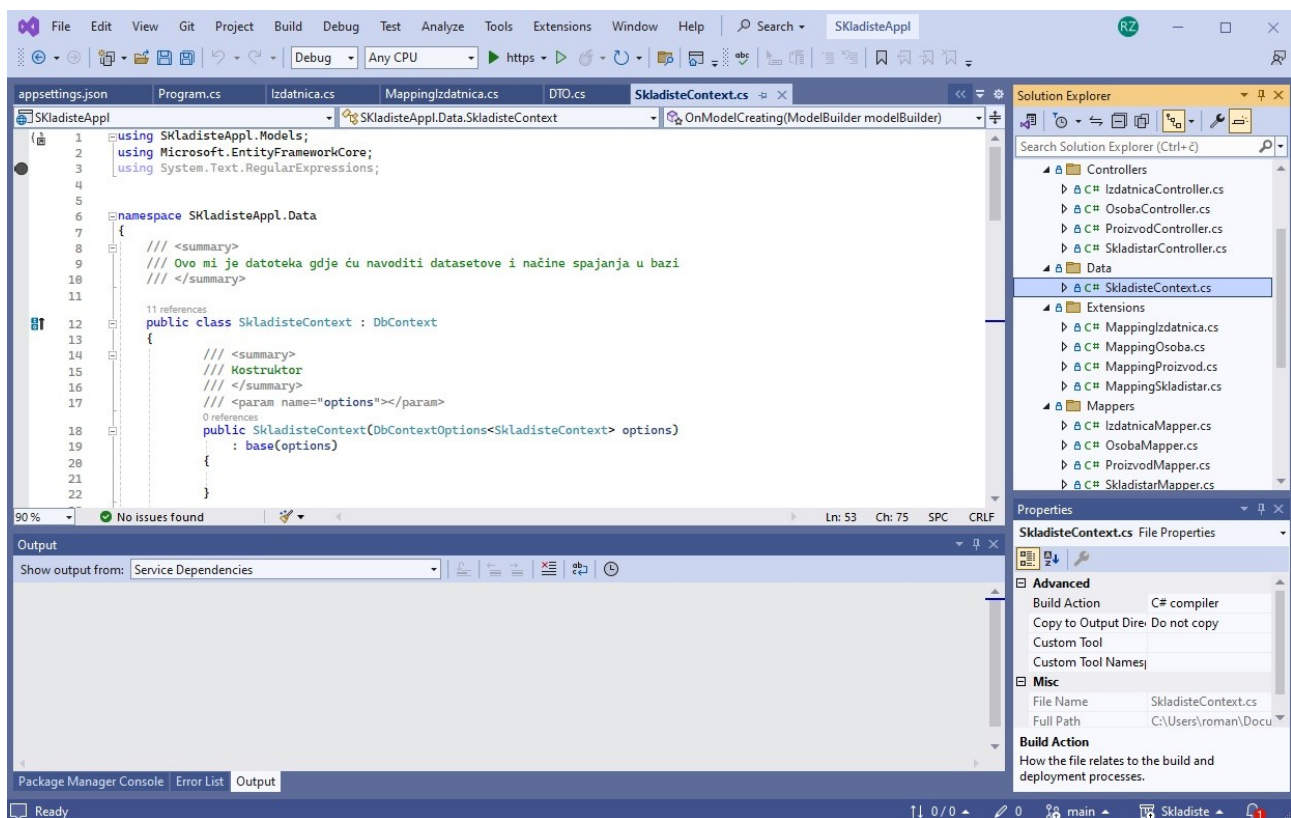
Mapperi se često koriste u arhitekturi softvera kao što su MVC (Model-View-Controller), RESTful web usluge, mikroservisi i ostali arhitektonski obrasci gdje je potrebno preslikavanje podataka između različitih komponenti ili slojeva aplikacije.

Kao što je i u mom slučaju. Ja sam koristio AutoMapper.

U C#-u, popularni alati za mapiranje uključuju AutoMapper, Mapster, emitmapper i rukom pisane mapirane metode. Ovi alati omogućuju definiranje pravila preslikavanja između objekata i automatiziranje procesa preslikavanja.

4.6 DB context :

DbContext: U kontekstu Entity Framework-a ili sličnih ORM (Object-Relational Mapping) alata, DbContext predstavlja sesiju komunikacije s bazom podataka. On sadrži logiku za upravljanje entitetima i povezivanje s bazom podataka. Koristio sam ga kako bi izvršio komunikaciju s bazom podataka. Korištenje "contexta" u softverskom razvoju omogućuje pristup relevantnim informacijama i resursima koji su potrebni za izvršavanje određene funkcionalnosti ili operacije. Ovisno o kontekstu primjene, "context" može biti od vitalnog značaja za ispravno funkcioniranje aplikacije.



Sl.4.6.1 DbContext

4.7 Controllers

U kontekstu web razvoja moje aplikacije, controller (kontroler) služi za obradu HTTP zahtjeva i upravljanje različitim tipovima zahtjeva koje korisnik ili klijentska aplikacija mogu poslati. Ključne metode koje se obično implementiraju u kontroleru su GET, POST, PUT i DELETE, odgovarajući standardnim HTTP metodama za čitanje, stvaranje, ažuriranje i brisanje resursa.

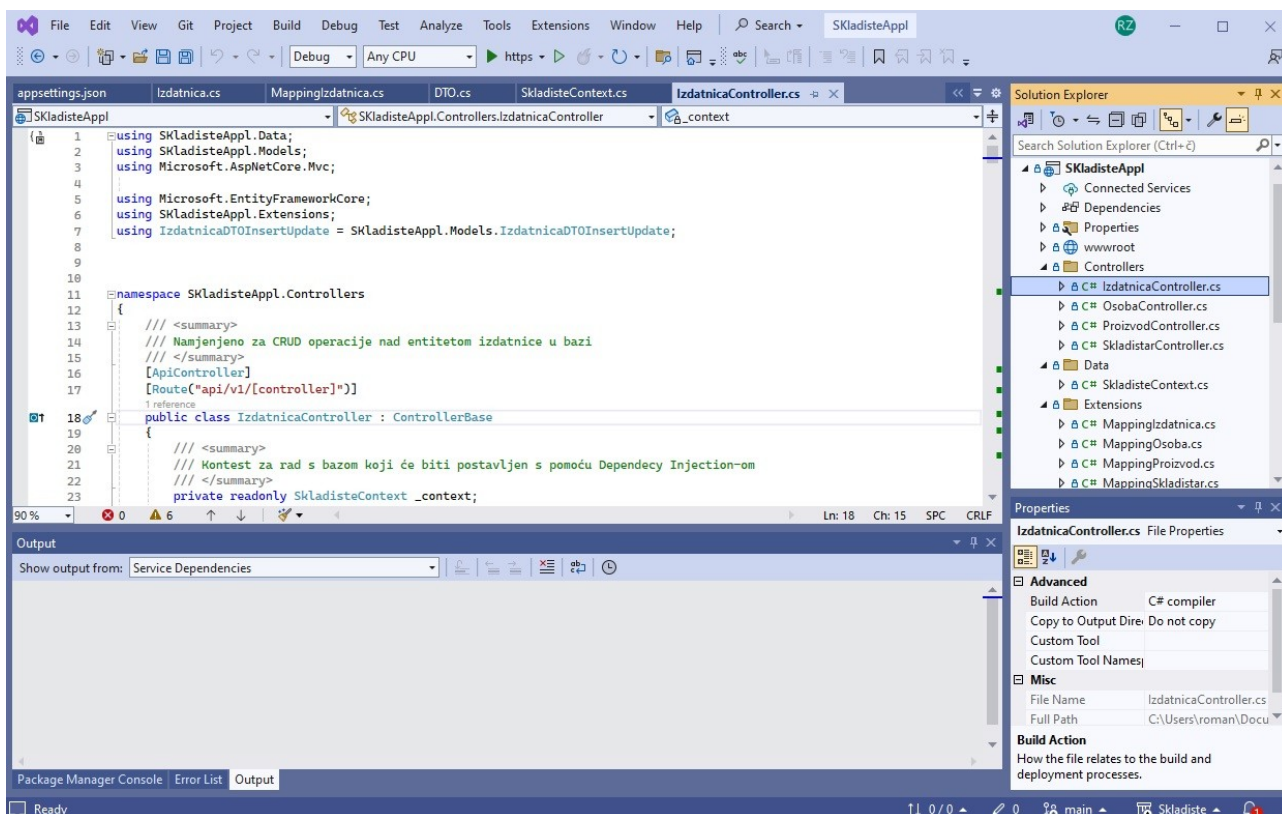
Evo kratkog pregleda kako se svaka od ovih metoda obično koristi u kontroleru:

1. **GET**: Koristi se za dohvaćanje podataka. Obično se koristi za prikazivanje informacija korisniku ili klijentskoj aplikaciji. Na primjer, GET metoda može se koristiti za prikazivanje popisa korisnika, detalja pojedinog korisnika ili drugih resursa.
2. **POST**: Koristi se za stvaranje novih resursa ili izvršavanje akcija koje mijenjaju stanje servera. Obično se koristi za slanje podataka na server, poput formi ili JSON objekata, kako bi se stvorio novi resurs. Na primjer, POST metoda se može koristiti za stvaranje novog korisnika ili dodavanje novog elementa u bazu podataka.
3. **PUT**: Koristi se za ažuriranje postojećih resursa. Obično se koristi za slanje podataka na server koji zamjenjuju postojeće podatke za određeni resurs. Na primjer, PUT metoda se

može koristiti za ažuriranje podataka o postojećem korisniku ili promjenu svojstava nekog elementa.

4. **DELETE**: Koristi se za brisanje resursa. Obično se koristi za slanje zahtjeva na server koji označava određeni resurs za brisanje. Na primjer, DELETE metoda se može koristiti za brisanje korisnika iz baze podataka ili uklanjanje određenog elementa iz sustava.

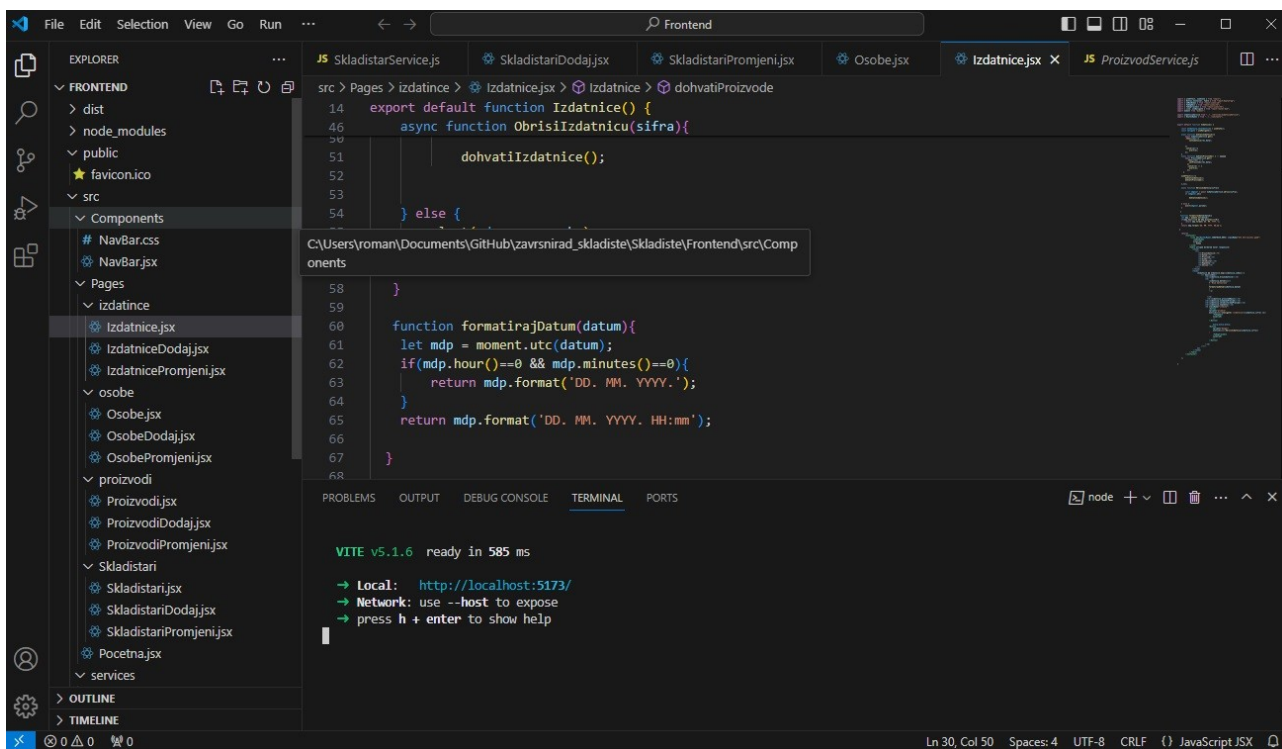
Kontroleri u ASP.NET Core, kao i u drugim okruženjima za web razvoj, obično imaju metode koje odgovaraju ovim HTTP metodama, te se kroz njih implementira logika za obradu zahtjeva i interakciju s modelom i servisima aplikacije.



Sl. 4.7.1 Controller

Sve ovo sam na kraju upogoni sa swaggerom **Sl. 2.1 Swagger**, kako bih testirao backend aplikacije. Za samu konstrukciju završnog rada sam se koristio programom **Visual Studio Code** kojeg sam spominjao u **3.1 Mogućnosti Microsoft okoline**. Navedeni program sam koristio za Frontend aplikacije.

5.0 Frontend-Visual Studio Code



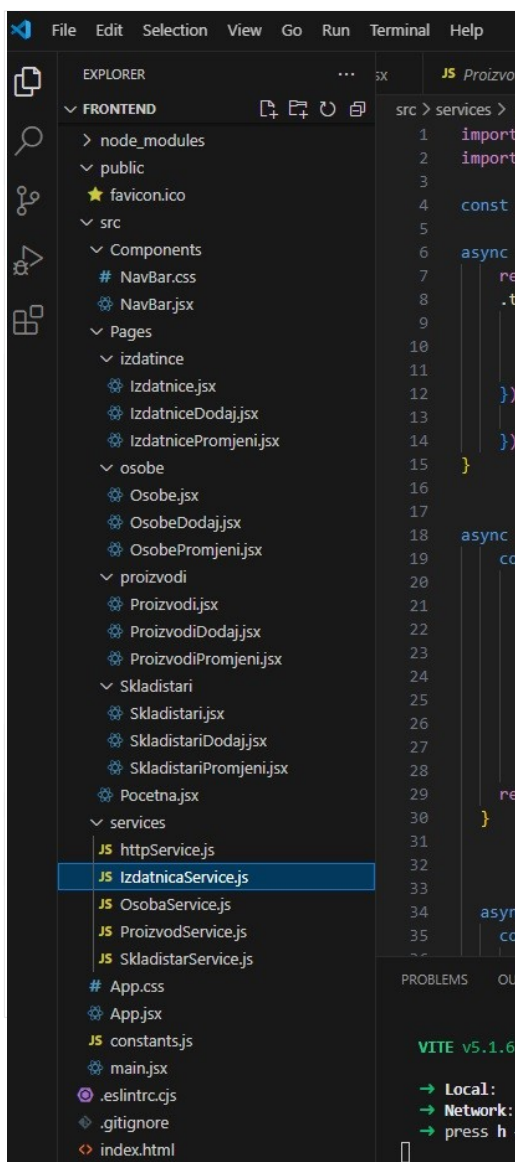
Sl. 5.0.1 Visual Studio Code

Visual Studio Code (VS Code) je besplatan i open-source tekstualni editor razvijen od strane Microsofta. Iako se zove "Visual Studio", VS Code je znatno lakši od punokrvnog razvojnog okruženja (IDE) kao što je Visual Studio.

Evo nekoliko ključnih značajki i informacija o Visual Studio Code:

- Cross-platform:** VS Code je dostupan za Windows, macOS i Linux operativne sustave, što ga čini izvrsnim izborom za razvoj na različitim platformama.
- Lagan i brz:** Za razliku od nekih drugih IDE-ova, VS Code je lagan i brz, što znači da se brzo pokreće i efikasno koristi resurse sustava.
- Prilagodljivost:** VS Code je visoko prilagodljiv zahvaljujući bogatom ekosustavu proširenja (extensions). Korisnici mogu instalirati proširenja za podršku različitim jezicima, alatima, temama i drugim značajkama koje poboljšavaju produktivnost i prilagođavaju iskustvo razvoja.
- Integracija s Git-om:** VS Code ima integriranu podršku za Git, što olakšava upravljanje verzijama i suradnju na projektima koji koriste Git za upravljanje izvornim kodom.

5. **Integracija s raznim jezicima i tehnologijama**: VS Code podržava širok raspon programskih jezika, uključujući JavaScript, TypeScript, Python, C#, Java, PHP i mnoge druge. Također pruža integraciju s popularnim alatima i okruženjima poput Node.js, .NET Core, Docker-a i mnogih drugih.
6. **Debugger**: VS Code dolazi s ugrađenim alatima za debugiranje, što omogućuje korisnicima lako pronalaženje i ispravljanje grešaka u kodu.
7. **Command Palette**: Omogućuje korisnicima brz pristup svim naredbama i postavkama unutar VS Code-a putem jednostavnog pretraživanja.
8. **Live Share**: Omogućuje suradnju u realnom vremenu na kodu s drugim developerima koristeći Live Share ekstenziju.
9. **Rad s raznim tipovima datoteka**: VS Code nije ograničen samo na razvoj softvera. Može se koristiti za uređivanje različitih tipova datoteka, uključujući tekstualne datoteke, konfiguracijske datoteke, Markdown datoteke i mnoge druge.



Koristio sam razne datoteke za spajanje na data bazu (SQL) , HttpService za određivanje adrese baze. Prvo je usljedila instalacija

NodeJS <https://nodejs.org/en/download>

potom kreiramo Vite + ReactJS aplikaciju s nazivom Frontend.

Potom se otvori u Visual studio code folder s navedenim nazivom.

U samom sam programu morao sam dodavati razne zavisnosti : react-router-dom, react-bootstrap, bootstrap, axios, react-number-format, react-icons, moment. Za više informacija se poslužite sljedećim linkovima:

<https://react-bootstrap.netlify.app/>

<https://getbootstrap.com/>

<https://axios-http.com/docs/intro>

Sl. 5.0.2 Datoteke

5.1 Frontend aplikacija skladišno poslovanje

U samom početku sam postavio Favicon , ikonicu koja se nalazi na adresnoj liniji .Icane se mogu koristiti postojeće ili napraviti svoje na generatoru <https://favicon.io/> . Krećemo sa main datotekom u kojoj moramo postaviti rutu same aplikacije . Ta se datoteka prva poziva kod pokretanja programa. U App.jsx pravimo route za svaku stranicu (src) te putanju . Datoteka App.css nam služi da bi organizirali razne zavisnosti u korištenju butona koje naravno koristim iz react-boostrapa . Na datoteci Pocetna.jsx sam napravio pozdravnu poruku koja se prva prikazuje prilikom aktivacije aplikacije.Izradio sam Service.js za sve entitete i HTTP zahtjev.

5.2 Service.js

Service.js je naziv fajla koji se često koristi u razvoju softvera kako bi se organizirala logika vezana za komunikaciju sa serverom ili za obavljanje određenih poslova u pozadini. Ovaj fajl obično sadrži funkcije ili klase koje omogućuju slanje HTTP zahtjeva, manipulaciju podacima i obradu odgovora koji se dobiju od servera.

Evo nekoliko tipova funkcionalnosti koje se mogu naći u `service.js` fajlu:

1. **HTTP zahtjevi:** Service fajl može sadržavati funkcije ili klase koje se koriste za slanje HTTP zahtjeva serveru. To može uključivati GET, POST, PUT, DELETE zahtjeve i druge HTTP metode koje su potrebne za interakciju sa serverom.
2. **Autentikacija i autorizacija:** Ako je aplikacija zahtjeva autentikaciju korisnika, service fajl može sadržavati funkcionalnost za slanje autentikacijskih zahtjeva serveru, kao i manipulaciju sa tokenima za pristup.
3. **Pristup bazi podataka:** Service fajl može sadržavati funkcije koje pristupaju bazi podataka, ili koriste ORM (Object-Relational Mapping) biblioteke kako bi olakšali čitanje, pisanje, ažuriranje i brisanje podataka.
4. **Manipulacija podacima:** Service fajl može sadržavati funkcije za manipulaciju podacima koji su primljeni od servera prije nego što se prikažu

na korisničkom interfejsu. To može uključivati filtriranje, sortiranje, transformaciju i druge operacije.

5. **Obrada grešaka:** Service fajl može sadržavati logiku za obradu grešaka koje se mogu pojaviti prilikom komunikacije sa serverom ili prilikom obrade podataka.
6. **Pomoćne funkcije:** Osim osnovnih funkcionalnosti vezanih za komunikaciju sa serverom, service fajl može sadržavati i pomoćne funkcije koje olakšavaju rad sa određenim tipovima podataka ili obavljaju druge korisne zadatke.

Ukratko, **service.js** fajl obično služi kao interfejs između korisničkog interfejsa (npr. web aplikacije) i servera, omogućavajući aplikaciji da komunicira sa serverom, pristupa podacima i obavlja druge zadatke neophodne za njeno pravilno funkcionisanje. U slučaju moje aplikacije se koristim kako bih upogonio komunikaciju kako sa serverom tako i sa stranicama (Pages).

5.3 Pages

Datoteke „**Pages**” imaju funkciju dodavanja izgleda atributima na konačnoj web aplikaciji. Mora se poštovati kako velika, mala slova prilikom imenovanja atributa, tako i potrebna ograničenja koja su navedena u samj bazi podataka (SQL). Na datotekama koje imaju datum, količine i sl.. vrijednosti se sam izračun i akcija izvode u Frontendu. Prvo se održuje sama konekcija s beekendom same stranice, te se upogonjuje gumb za dodavanje vrijednosti, te se dodaje mogućnost promjene vrijednosti. Sve to upogonjuje service-naziv stranice `Service.js`. Pogledaj sliku **Sl. 5.0.2**

Datoteke. Te se koristi **NavBar.jsx**. U mojoj aplikaciji se koristim **React.js**, **NavBar.jsx**. Ova komponenta napisana u JavaScriptu ili TypeScriptu koja se koristi za definiranje izgleda i funkcionalnosti navigacijske trake aplikacije. Ova komponenta se može koristiti na svakoj stranici aplikacije kako bi se osigurala dosljednost i lakša navigacija korisnika kroz aplikaciju. Obično se koristi u razvoju web aplikacija kako bi se definirao navigacijski traka (ili traka sa izbornicima) koja

















se prikazuje na vrhu ili na nekom drugom dijelu korisničkog interfejsa. Ova traka obično sadrži izborne stavke koje omogućuju korisnicima da navigiraju kroz aplikaciju ili da pristupe različitim funkcionalnostima.

romanzaric-001-site1.itemurl.com/proizvodi

Google Imported From Fire... Imported (1) YouTube Ljepilo za tekstil Pen... Moj Telekom Portal... Gmail YouTube Nova mapa Karte Naslovnica Oglasi igra http://radio.garden/ Google Translate

Skladište APP Izbornik API dokumentacija

+ Dodaj

Naziv	SifraProizvoda	MjernaJedinica	Akcija
lopata	100	kom	 
metla	101	kom	 
sapun	102	kom	 
šampon	103	lit	 
gedore	104	gar	 
deterdent	105	kg	 
pijesak	106	m3	 
popopo	gfjspjirges	fesfefespfe	 

Sl. 6.0 Aplikacija skladišno poslovanje osnovni prikaz

