Sustav za praćenje skladišta trgovine u C#

Franjić, Matej

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matej Franjić

Sustav za praćenje skladišta trgovine u C#

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matej Franjić

Matični broj: 0016154467

Studij: Informacijski i poslovni sustavi – Razvoj programskih sustava

Sustav za praćenje skladišta trgovine u C#

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, lipanj 2024.

Matej Franjić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ovaj završni rad se bavi proučavanjem, primjenom i integracijom programskog jezika C# i komponenti LINQ i WPF razvojnog okvira .NET dizajniranog od strane Microsofta. Navedeno će biti implementirano u praktičnom programskom rješenju aplikacije za upravljanje skladištenjem i transakcijama trgovine.

Ideja rada je teoretski i praktično proučiti programski jezik C# u kontekstu razvoja stolnih aplikacija te integraciju spomenutih komponenti .NET okvira u rada i razvoja sa C# jezikom.

Od spomenutih komponenti, LINQ je zadužen i bit će korišten za dohvaćanje i rad s bazama podataka, a Microsoftova WPF komponenta, nastala kao zamjena za Windows Forms, služit će za rad s UI dijelom aplikacije.

Ključne riječi: C#, WPF, LINQ, .NET, SQL server, skladišni sustav,

Sadržaj

1. Uvod 1
2. Baza podataka 2
2.1. Microsoft SQL Server Management Studio 2
2.2. Baza u aplikaciji praktičnog dijela 3
3. C# 7
3.1. Što je c# 7
3.2. Različite primjene C# jezika 9
4. LINQ 10
4.1. Što je LINQ 10
4.2. Prednosti i nedostatci LINQ-a naspram SQL upita 11
5. WPF 12
5.1. Što je WPF 12
6. Implementacija aplikacije 13
6.1. Način rada aplikacije – front-end 13
6.2. Način rada aplikacije – back-end 24
6.2.1. Data access layer (DAL) 24
6.2.2. Bussiness Logic Layer (BLL) 28
6.2.3. Presentation Layer (PL) 30
7. Zaključak 33
Popis literature 34
Popis slika 35

1. Uvod

Ovaj završni rad nastao je s ciljem teoretskog i praktičnog proučavanja i istraživanja različitih tehnologija potrebnih za razvoj i održavanje stolnih aplikacija različitih složenosti kroz njihovu implementaciju u obliku stolne aplikacije za upravljanjem i praćenjem stanja skladišta trgovine. Proučavat će se rad s programskim jezikom C# unutar razvojnog okvira .NET koristeći različite komponente koje nam sam okvir nudi.

Neke od ključnih komponenti koje će se koristiti su LINQ za spajanje na bazu podataka, koja će biti implementirana koristeći Microsoftov SQL Server, i dohvaćanje podataka sa samog servera. LINQ komponenta je specifična po tome što ne zahtjeva poznavanje SQL koda za rad pri spajanju i dohvaćanju podataka iz baze.

Za razvoj i izradu korisničkog sučelja (eng. User Interface, UI) koristit će se komponenta po nazivu Windows Presentation Foundation (skraćeno WPF) koju je razvio Microsoft kao zamjenu za Windows Forms koja se također koristi u .NET razvojnom okviru za izradu stolnih aplikacija. Windows Presentation Foundation koristi razvojni jezik XAML razvijen na temelju XML programskog jezika za kreiranje grafičkog sučelja.

Aplikacija će služiti za prikaz tehnologija u njihovoj praktičnoj primjeni. Kroz izradu aplikacije provjeravat će te različite mogućnosti rada spomenutih tehnologija i komponenti. Provjerit će se korištenost tehnologija u današnjem razvoju, te njihova korisnost u izradi aplikacije. Također će se i dodatno spomenuti razvoj koristeći danas sve prisutniji AI, te će se provjeriti koliko je on koristan u razvoju s našim tehnologijama.

2. Baza podataka

2.1. Microsoft SQL Server Management Studio

Podaci korišteni u radu aplikacije bit će spremljeni u bazu pohranjenu na Microsoftov SQL Server Management Studio (skraćeno SSMS). Najnovija verzija SSMS za dan 11. kolovoza 2024. je 20.2.30.0 izdana 09. srpnja 2024., te se može preuzeti na službenoj Microsoftovoj stranici[1] gdje se također nalazi i sva potrebna dokumentacija o radu s aplikacijom.

SQL Server Management Studio koristi se za upravljanje SQL serverom i SQL bazama podataka. Koristi se primarno za Windows platforme, ali također podupire povezivanje s drugim platformama poput Linux-a i MacOS-a. Danas je jedan od korištenijih sustava za rad s bazama podataka što potvrđuju i broje statističke analize provede po pitanju toga.

Prema 6sense.com[2] statistici provedenog za godinu 2024 SSMS zauzima 28.29% tržišta čime zauzima mjesto broj jedan u svojoj kategoriji. Ova statistika pokazuje nam postotak krajnjih korisnika koji koriste aplikaciju koja za svoj rad koristi SSMS, a podatci su iz njihove vlastite baze podataka.

Slika 1. Udio tržišta SSMS (6sense.com)

Analiza stranice worldmetrics.org[3] objavljena 23. srpnja 2024. govori da SSMS zauzima 17.1% tržišta i ostvaruje prihode od 1.5 milijardi USD, a ima ukupno 2 milijuna aktivnih baza podataka. Ovdje je postotak manji jer se provodi na širem tržištu i većem broju korisnika.

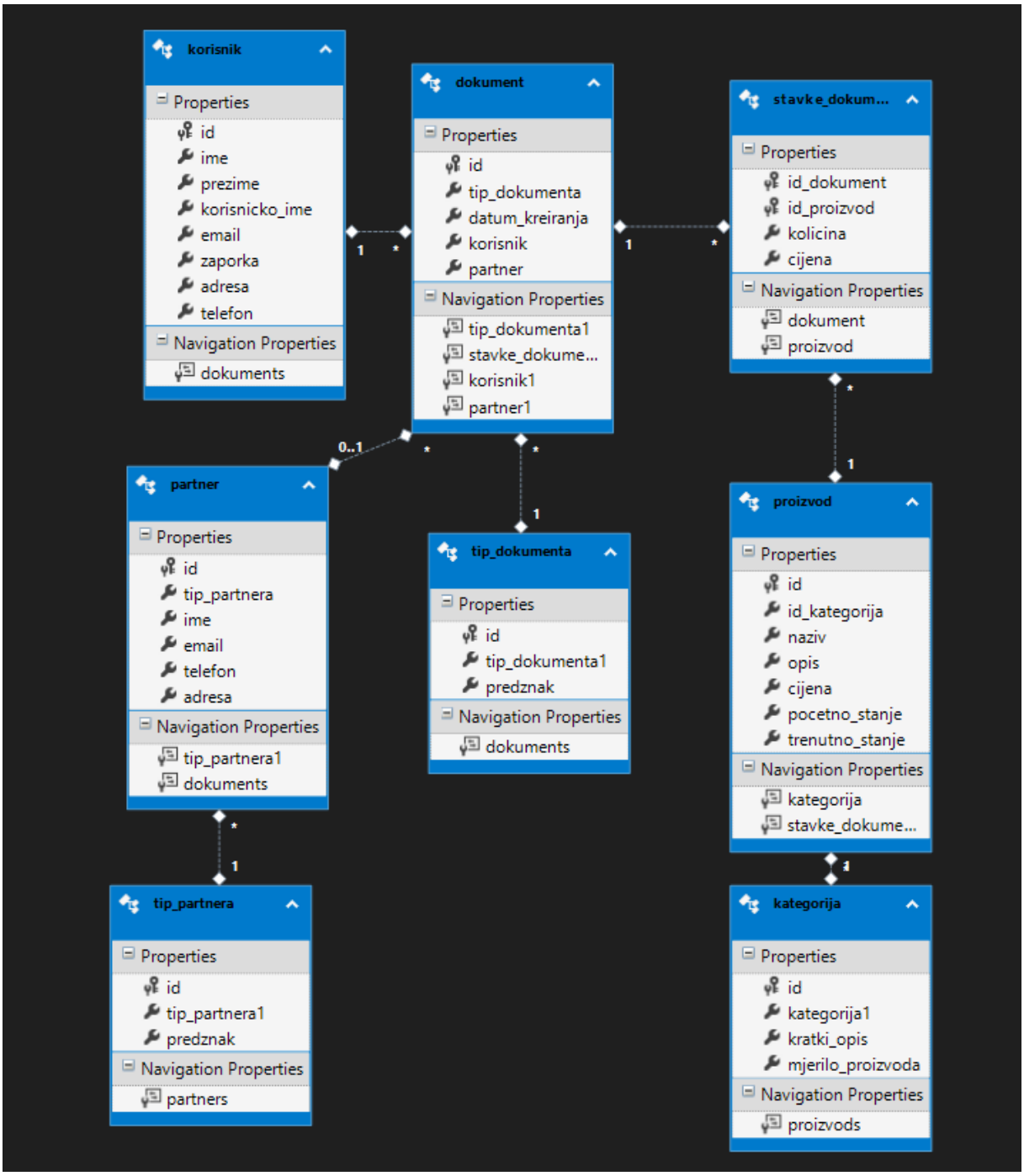
[1] https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16

[2] https://6sense.com/tech/database/microsoft-sql-server-market-share

[3] https://worldmetrics.org/sql-server-statistics/

2.2. Baza u aplikaciji praktičnog dijela

Kao što je već spomenuto baza podataka je izrađena u aplikaciji SQL Server Management Studio. Baza se sastoji od 8 različitih tablica. Za spajanje s aplikacijom korištena je Microsoftova ekstenzija za Visual Studio zvana EntityFramework koja omogućuje brzu i jednostavnu implementaciju modela baze i tablica u programsko rješenje.



Slika 2. Model baze podataka (samostalna izrada)

Glavni dio baze je tablica *dokument* koja sprema sve relevantne dokumente potrebne za praćenje stanja skladišta. Ona u sebi sadrži elemente *tip_dokumenta* koji je vanjski ključ na istoimenu tablicu, *datum_kreiranja* gdje je pohranjen datum i vrijeme spremanja objekta u bazu te *partner* i *korisnik* koji su vanjski ključevi na istoimene tablice.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	tip_dokumenta	int	☐
	datum_kreiranja	datetime	☐
	korisnik	int	☐
	partner	int	☑

Slika 3. tablica "dokument" (samostalna izrada)

Tablica *korisnik* pohranjuje korisnike aplikacije. Korisnici navedeni u ovoj tablici imaju mogućnost prijave u samu aplikaciju. Ova tablica sadrži *ime* i *prezime* korisnika, *korisnicko_ime* i *zaporku* koji su potrebni za prijavu u aplikaciju te *email, telefon* i *adresa* koja bilježi kontakte korisnika.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	ime	varchar(50)	☐
	prezime	varchar(50)	☐
	korisnicko_ime	varchar(50)	☐
	email	varchar(250)	☐
	zaporka	varchar(250)	☐
	adresa	varchar(250)	☑
	telefon	varchar(20)	☑

Slika 4. tablica "korisnik" (samostalna izrada)

Tablica *tip_dokumenta* koja sadrži različite tipove dokumenta od kojih svaki ima svoj *predznak*. Predznak tipa obilježava skidaju li se proizvodi sa skladišta ili se dodaju na njega.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	tip_dokumenta	varchar(50)	☐
	predznak	varchar(10)	☐

Slika 5. tablica "tip_dokumenta" (samostalna izrada)

Tablica *partner* sprema i bilježi podatke o našim partnerima. Ovdje su spremljeni oni partneri s kojima korisnici skladišta često trguju bili oni dostavljači robe, kupci ili oboje. U tablicu spremamo *tip_partnera, ime, email, telefon* i njegovu *adresu.* Također partner nije nužan vanjski ključ u tablici *dokument* što znači da mogu postojati dokumenti koji ne sadrže partnera.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	tip_partnera	int	☐
	ime	varchar(250)	☐
	email	varchar(250)	☑
	telefon	varchar(20)	☑
	adresa	varchar(250)	☑

Slika 6. tablica "partner" (samostalna izrada)

Tablica *tip_partnera* koja se veže na prethodno spomenutu tablicu sadrži podatke o tipu korisniku i njegov predznak. Predznak slično kao o kod tipa dokumenta označava radi li se o kupcu, dostavljaču ili trgovinskom partneru kod kojeg trgovina ide obostrano. Ova oznaka postoji kako bi se pri kreiranju dokumenta lakše pronašli i odabrali partneri relevantni za određeni tip dokumenta.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	tip_partnera	varchar(50)	☐
	predznak	varchar(10)	☑

Slika 7. tablica "tip_partnera" (samostalna izrada)

Tablica *stavke_dokumenta* je poveznica između tablica *dokument* i *proizvod*. Ona sadrži podatke o tome koji dokument sadrži koje proizvode. Osim dokumenta i proizvoda također se bilježe količina proizvoda i ukupna cijena proizvoda za navedenu količinu.

	Column Name	Data Type	Allow Nulls
🔑	id_dokument	int	☐
🔑	id_proizvod	int	☐
	kolicina	varchar(50)	☐
	cijena	varchar(50)	☐

Slika 8. tablica "stavke_dokumenta" (samostalna izrada)

Tablica *proizvod* sadrži sve potrebne informacije o proizvodima na skladištu. Osim samog *naziva* proizvoda tablica sadrži *opis* i *cijenu* proizvoda po komadu. Tablica također ključno prati i stanje proizvoda na zalihama te bilježi i početno stanje kako bi se bolje pratilo stanje.

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	id_kategorija	int	☐
	naziv	varchar(50)	☐
	opis	varchar(250)	☑
	cijena	varchar(20)	☐
	pocetno_stanje	float	☐
	trenutno_stanje	float	☐

Slika 9. tablica "proizvod" (samostalna izrada)

Tablica *kategorija* sprema naziv kategorije proizvoda, kratak opis te kategorije i mjerilo kojim se mjere predmeti u toj kategoriji (L, kg, kom).

	Column Name	Data Type	Allow Nulls
🔑	id	int	☐
	kategorija	varchar(50)	☐
	kratki_opis	varchar(250)	☑
	mjerilo_proizvoda	varchar(50)	☑

Slika 10. tablica "kategorija" (samostalna izrada)

3. C#

3.1. Što je c#

C# (C SHARP) je programski jezik nastao sredinom 2000. godine kao još jedan jezik u obitelji C jezika. Razvijen je na greškama i iskustvu iz prethodnih jezika iz spomenute obitelji odnosno ima dobre performanse kao i jezik C, ali je i objektno orijentiran kao i C++ (Liberty, 2005). Stekao je svoju popularnost zbog asocijacije s .NET razvojnim okvirom. Povećanjem popularnosti .NET razvojnog okvira u izradi i razvoju stolnih aplikacija također se povećavala i sama popularnost C# jezika.

Dolaskom C# verzije 3.0 i standardiziranih queryja, te lambda funkcija .NET i C# postaju još zastupljeniji u programerskom svijetu (Hejlsberg, 2008).

Prema stranici orientsoftware.com[4] tržišni udio C# zauzima peto mjesto sa 6.73%. Statistika je provedena za 2024. godinu:

Rank	Programming Language	Market Share	Trend
1	Python	28.11%	+0.6%
2	Java	15.52%	-0.1%
3	JavaScript	8.57%	-0.1%
4	C/C++	6.92%	+0.1%
5	C#	6.73%	-0.1%
6	R	4.75%	+0.7 %
7	PHP	4.57%	-0.6%
8	TypeScript	2.78%	-0.0%

Slika 11. tržišni udio C# jezika (orientsoftware.com)

[4] https://www.orientsoftware.com/blog/most-popular-programming-languages/

Osim što zauzima veliki tržišni dio poznavanje C# jezika može olakšati i pronalaženje posla jer prema Linkedin-u[5] u Hrvatskoj u mjesecu srpnju 2024. godine postoji 487 oglasa za posao koji zahtijevaju poznavanje C# programskoj jezika.

Slika 12. Popis C# poslova (Linkedin.com)

[5] https://hr.linkedin.com/jobs/c%23-developer-jobs

3.2. Različite primjene C# jezika

Kao što je već spomenuto C# je vrlo čest programski jezik kada se radi o stolnim aplikacijama. No prema nekim izvorima C# nije jedino popularan u tom području. Kao što je navedeno na stranici WordPressa[6] C# je cross-platform programski jezik. Cross-platform označava da se programski jezik može koristiti kroz više platformi poput WEB developmenta, te svih operacijski sustava stolnog računala (iOS, Linux, Windows).

Na službenoj Microsoft web stranici .NET razvojnog okvira se mogu pronaći nekolicina dodatnih alata i ekstenzija za pomoć pri radu u web developmentu[7] od kojih je jedan ASP.NET Core, koji je open-source i cross-platform razvojni okvir namijenjen za razvoj web aplikacija i API-ja. ASP.NET Core omogućava izradu skalabilnih, visokih performansi web aplikacija koje mogu raditi na različitim platformama, uključujući Windows, macOS i Linux.

C# se koristi i za razvoj mobilnih aplikacija pomoću Xamarin platformi, koja omogućava razvoj mobilnih aplikacija za iOS i Androin. Xamarin omogućava razvoj aplikacija koje su optimizirane za različite platforme, koristeći zajednički C# kod, čime se štedi vrijeme i resursi.

Jedna od najpoznatijih primjena C# jezika izvan razvoja poslovnih aplikacija je u industriji igara. Unity, jedan od najpopularnijih game engine-ova u svijetu, koristi C# kao svoj glavni skriptni jezik. Razvoj igara u Unityu omogućava stvaranje 2D i 3D igara za više platformi, uključujući Windows, macOS, iOS, Android, pa čak i konzole poput PlayStationa i Xboxa.

[6] https://basicsofwebdevelopment.wordpress.com/2015/12/08/using-c-for-cross-platform-development/

[7] https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps

4. LINQ

4.1. Što je LINQ

LINQ (Language Integrated Query) je alat unutar .NET razvojnog okvira koji omogućava pisanje queryja (upita) pomoću C# programskog jezika. Njegova ključna karakteristika je to što omogućava jednostavno rukovanje različitim vrstama podataka (Microsoft, 2023).

LINQ se temelji na tome da upiti prema bazi podataka izgledaju kao i drugi cio C# programa te sintaktički dosta podsjeća na C# kod. To je bitno kako bi se olakšalo pisanje upita te se smanjila razina eventualnih grešaka.

LINQ je uveden s verzijom 3.5 .NET frameworka. Njegova uloga dolazi do značaja pri spajanju s bazom podataka gdje korisnik ne mora poznavati SQL razvojni jezik i njegovu sintaksu umjesto toga pomoću LINQ-a s lakoćom obavlja ono što s SQL ne odradi.

Razlika sintakse se može vidjeti u sljedeće navedenim primjerima iz kasnije spomenutog praktičnog dijela;

```
public virtual IQueryable<T> GetAll()
{
    var query = from e in Entities select e;
    return query;
}
```

Slika 13. primjer LINQ jednostavne funkcije (samostalna izrada)

```
private void GetTotalPrice()
{
    foreach(var item in documentItems)
    {
        totalPrice += double.Parse(item.cijena);
    }
}
```

Slika 14. primjer C# jednostavne funkcije (samostalna izrada)

4.2. Prednosti i nedostatci LINQ-a naspram SQL upita

Jedna od najvećih prednosti je to što sintaktički dosta podsjeća na C# te je vrlo jednostavan za naučiti ukoliko se poznaje C# razvojni jezik i koristiti ga s ostalim .NET tehnologijama i alatima. Zbog toga je vrlo jednostavna za integraciju u takvo okruženje.

LINQ omogućava također da se stupci tablice koriste kao objekti što programeru olakšava obradu podataka i rad s njima. Samim time smanjuje i mogućnost grešaka koje mogu uzrokovati razlike u tipovima podataka. Osim toga LINQ je type-safe što znači da će već prilikom kompiliranja aplikacije izbaciti se greška o pogrešnom upitu.

Također se može koristiti i za rad s različitim izvorima podataka, uključujući baze podataka, XML datoteke, JSON datoteke, liste, polja i rječnike. Ova fleksibilnost ga čini dobrom opcijom jer programer nema potrebu za učenjem i korištenjem više jezika upita za različite izvore podataka.

Međutim, prilikom svakog izvršavanja LINQ upita cijeli upit se obrađuje ponovo, čak i kada je samo mali dio podataka promijenjen od posljednjeg izvršavanja. Također LINQ izravno ne podržava značajke poput enkripcije i dekripcije podataka. Iako je moguće koristiti LINQ za šifriranje i dešifriranje podataka, to se obično radi uz pomoć drugih alata i tehnika, što može povećati složenost.

5. WPF

5.1. Što je WPF

WPF (Windows Presentation Foundation) je dio .NET razvojnog okvira. WPF je u svome razvoju dosta sličan drugim .NET alatima poput ASP.NET-a ili Windows Forms-a. U WPF-u se također instanciraju klase, postavljaju svojstva, pozivaju metode i obrađuju događaje. WPF za usporedbu dodaje naprednije konstrukte poput ovisničkih svojstava (dependency properties) i proslijeđenih događaja (routed events).

Jedan od ključnih svojstva WPF-a je mogućnost razvoja aplikacije koristeći markup jezik (XAML) u kombinaciji s code-behind pristupom. XAML je XML-bazirani jezik za razvoj front-end dijela aplikacije, dok se funkcionalnosti aplikacije implementiraju u programskom jeziku. Ovaj razdvojeni pristup omogućuje dizajnerima i programerima da istovremeno rade na različitim aspektima aplikacije, što poboljšava efikasnost razvoja. Također, olakšava održavanje jer izgled aplikacije nije čvrsto povezan s funkcionalnim kodom.

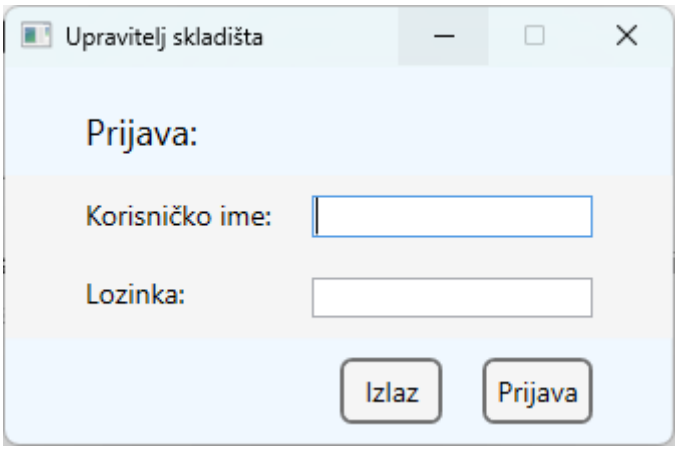
XAML se koristi za deklarativno definiranje prozora, dijaloških okvira, stranica i korisničkih kontrola, te za popunjavanje tih elemenata različitim kontrolama, oblicima i grafikom. Kod koji reagira na korisničke interakcije, poput klika na gumb ili izbornik, implementira se u code-behind datotekama, što omogućuje jasnu organizaciju i odvajanje izgleda od poslovne logike.

Prednosti WPF-a u usporedbi s Windows Forms su to što WPF nudi sofisticiranije i vizualno privlačnije korisničko sučelje, WPF omogućava veću fleksibilnost i prilagodljivost izgleda samog sučelja i skalabilnost aplikacija u WPF-u omogućava bolji prikaz aplikacije u različitim rezolucijama nego što to Forms čini.

6. Implementacija aplikacije

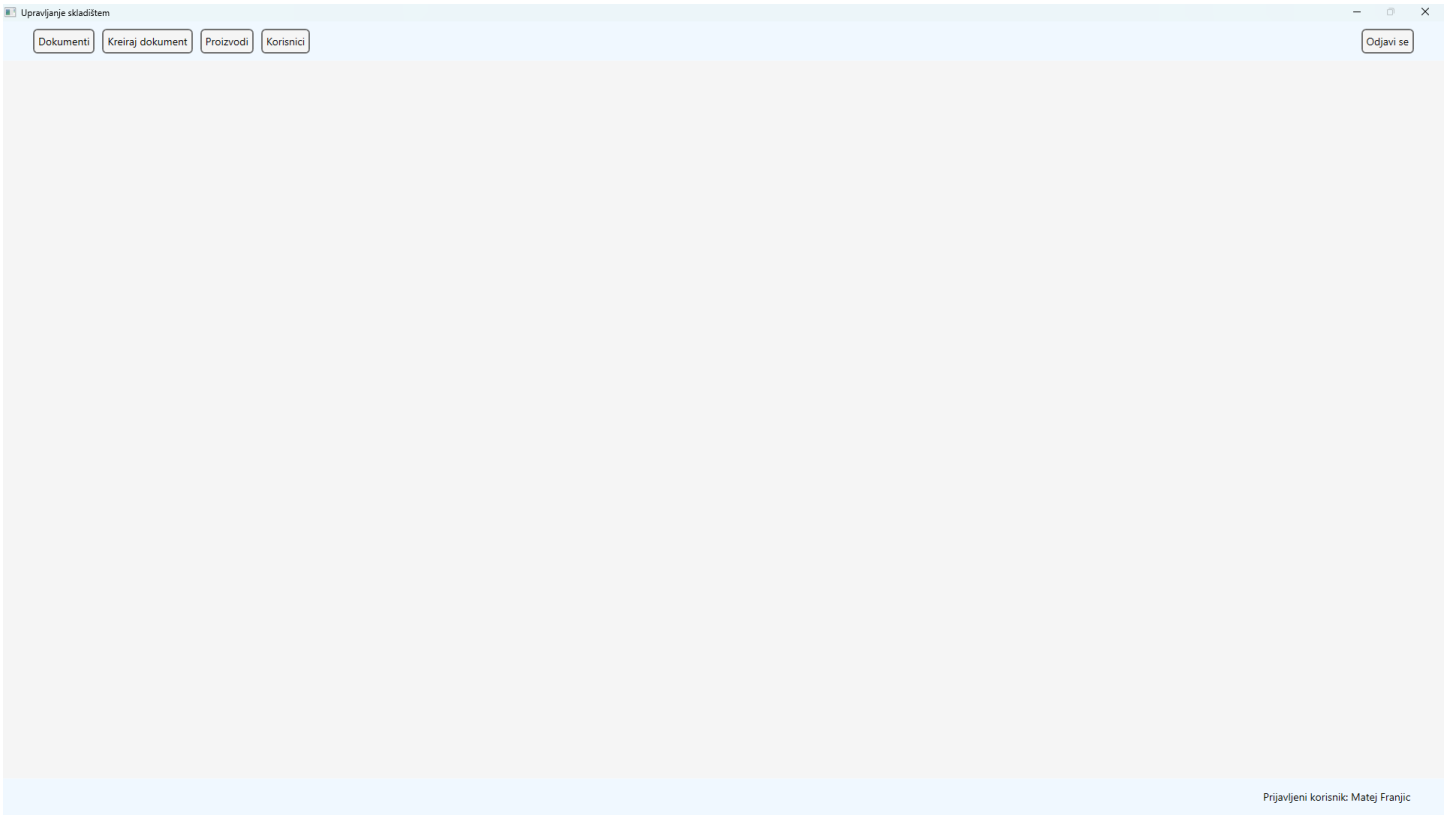
6.1. Način rada aplikacije – front-end

Aplikacija pruža nekoliko opcija kao što su kreiranje dokumenata i njihov pregled. Pri pokretanju aplikacije korisnik prvo prolazi kroz „Login“ zaslon:



Slika 15. prikaz Login zaslona (samostalna izrada)

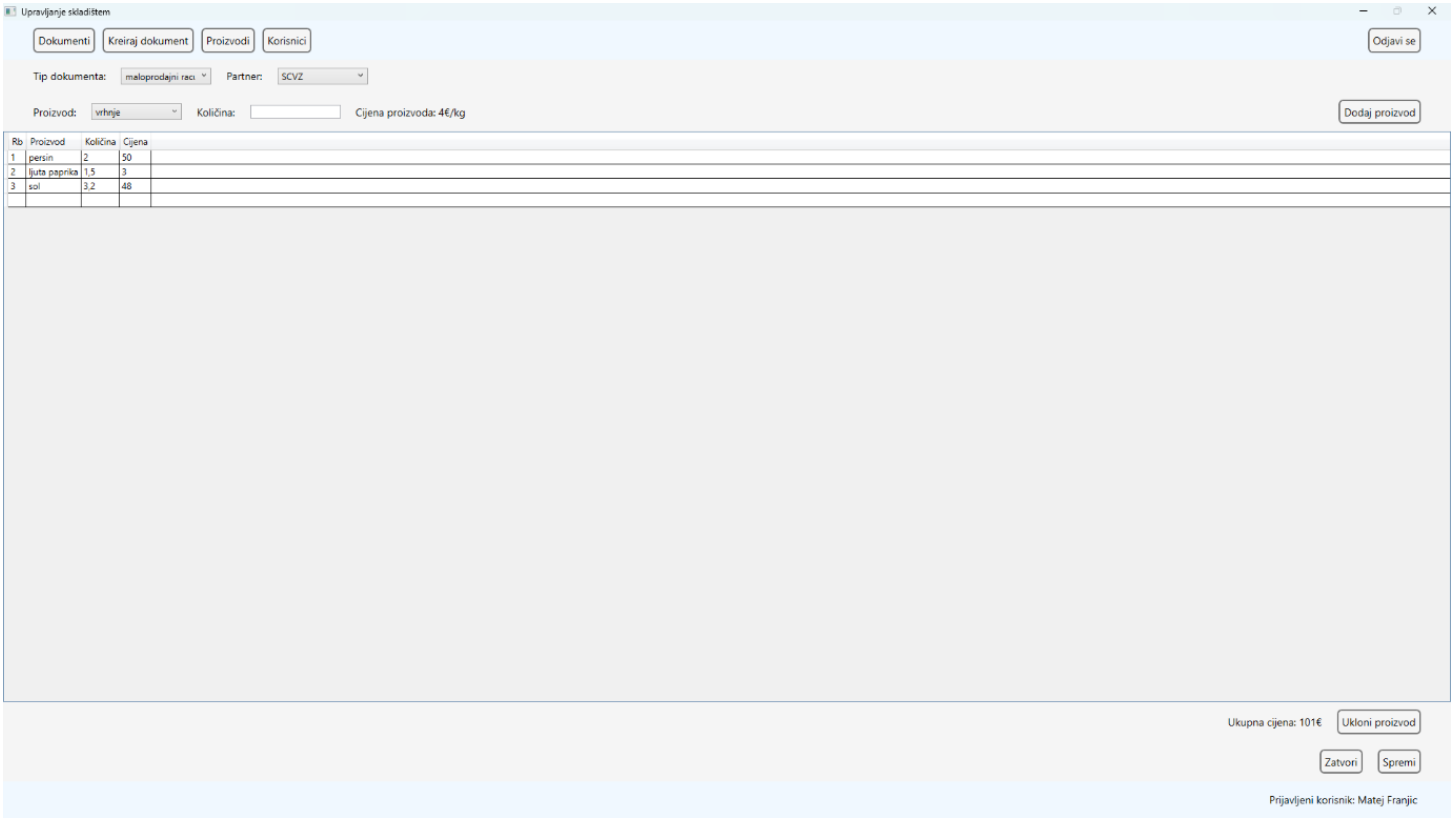
Korisnik unosi svoje podatke te pomoću njih se prijavljuje u aplikaciju. Podatci se uspoređuju s korisnicima iz baze podataka te u slučaju da prijava uspije korisnik dolazi na početni zaslon.



Slika 16. prikaz početnog zaslona (samostalna izrada)

Na početnom zaslonu korisnik ima mogućnost otvoriti 4 glavne funkcionalnosti koje program nudi, a to su *Dokumenti*, *Kreiraj dokument*, *Proizvodi* i *korisnici*. Osim toga u aplikaciji se prikazuje prijavljeni korisnik te postoji gumb odjave koji korisnika vraća na *Login* zaslon.

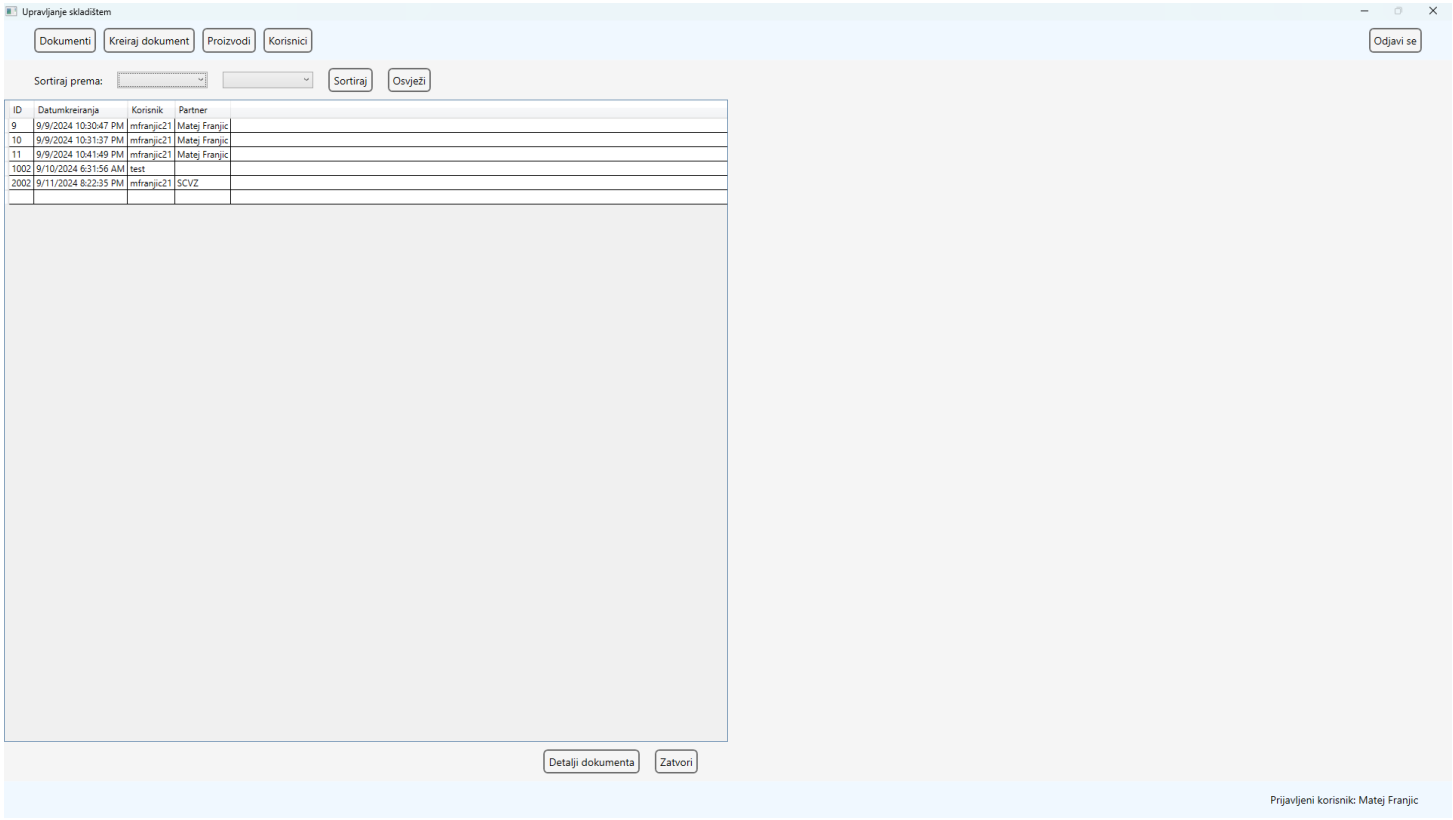
Ovaj zaslon služi kao temelj na kojem se prikazuju drugi prikazi s funkcionalnostima. Pritiskom na bilo koji gumb se otvara zaseban prikaz (UserControl u WPF sintaksi).



Slika 17. prikaz kreiranja dokumenta (samostalna izrada)

Gumbom *Kreiraj dokument* otvara nam se forma u kojoj odabire podatke potrebne za dodavanjem novog dokumenta. Korisnik odabire tip dokumenta i partnera iz padajućih izbornika (ComboBox u WPF-u).

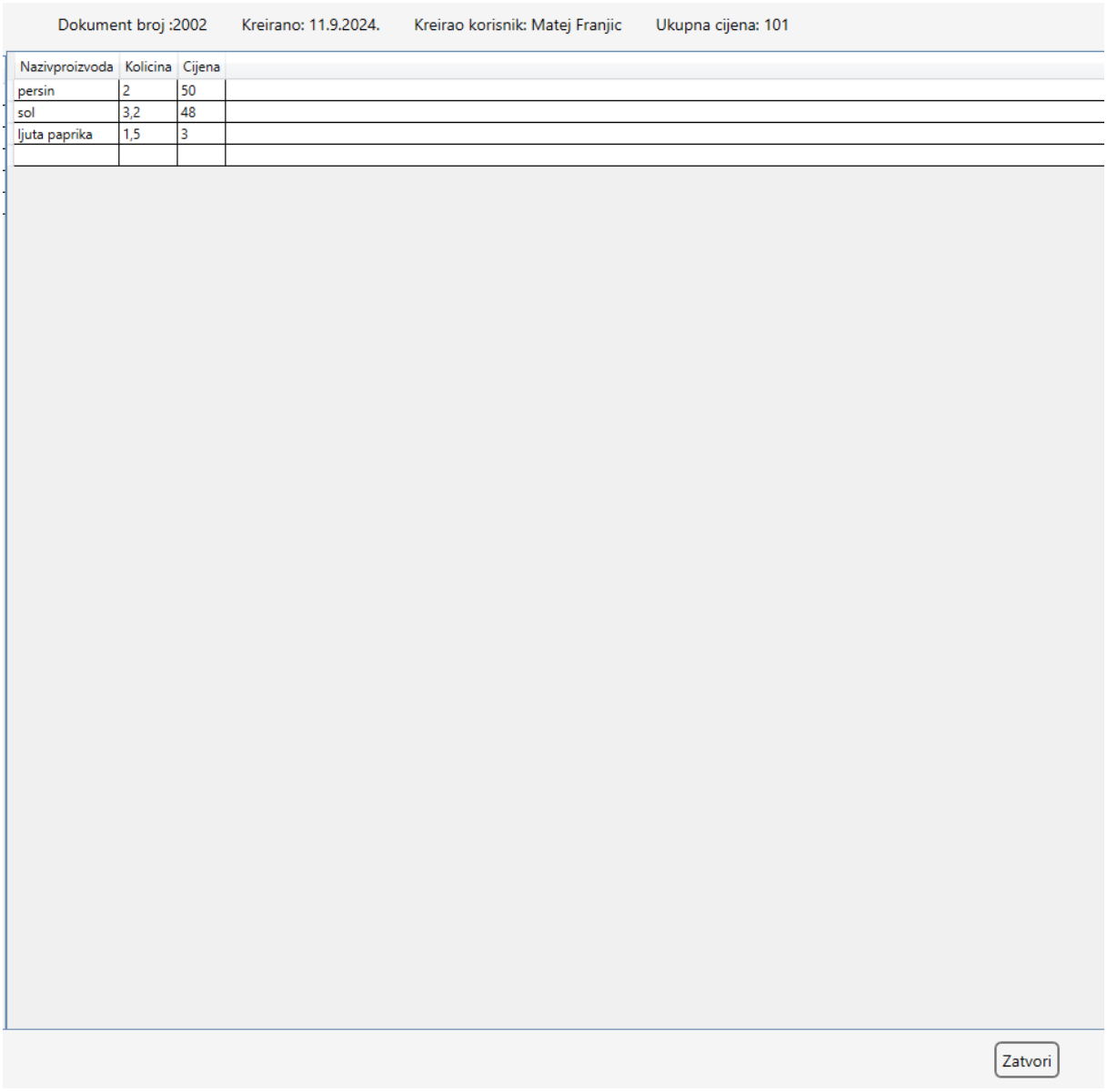
Za svaki odabrani produkt se unosi količina željenog proizvoda. Klikom na gumb *Dodaj proizvod* on se dodaje u tablični prikaz (DataGrid u WPF-u). Stavke u tablici se kasnije dodaju kao redovi *stavke_dokumenta* tablice iz baze podataka i vezani su uz kreirani dokument.



Slika 18. prikaz dokumenata (samostalna izrada)

Pritiskom gumba *Dokumenti* otvara se prikaz svih kreiranih dokumenata. U padajućem izborniku također možemo odabrati prema čemu želimo sortirati dokumente (tip dokumenta, korisnik, partner), a potom odabiremo određeni stavku iz te tablice (primjerice tip dokumenta – maloprodajni račun, korisnik – test, partner – SCVZ). Nakon toga se ispisuje filtrirani sadržaj.

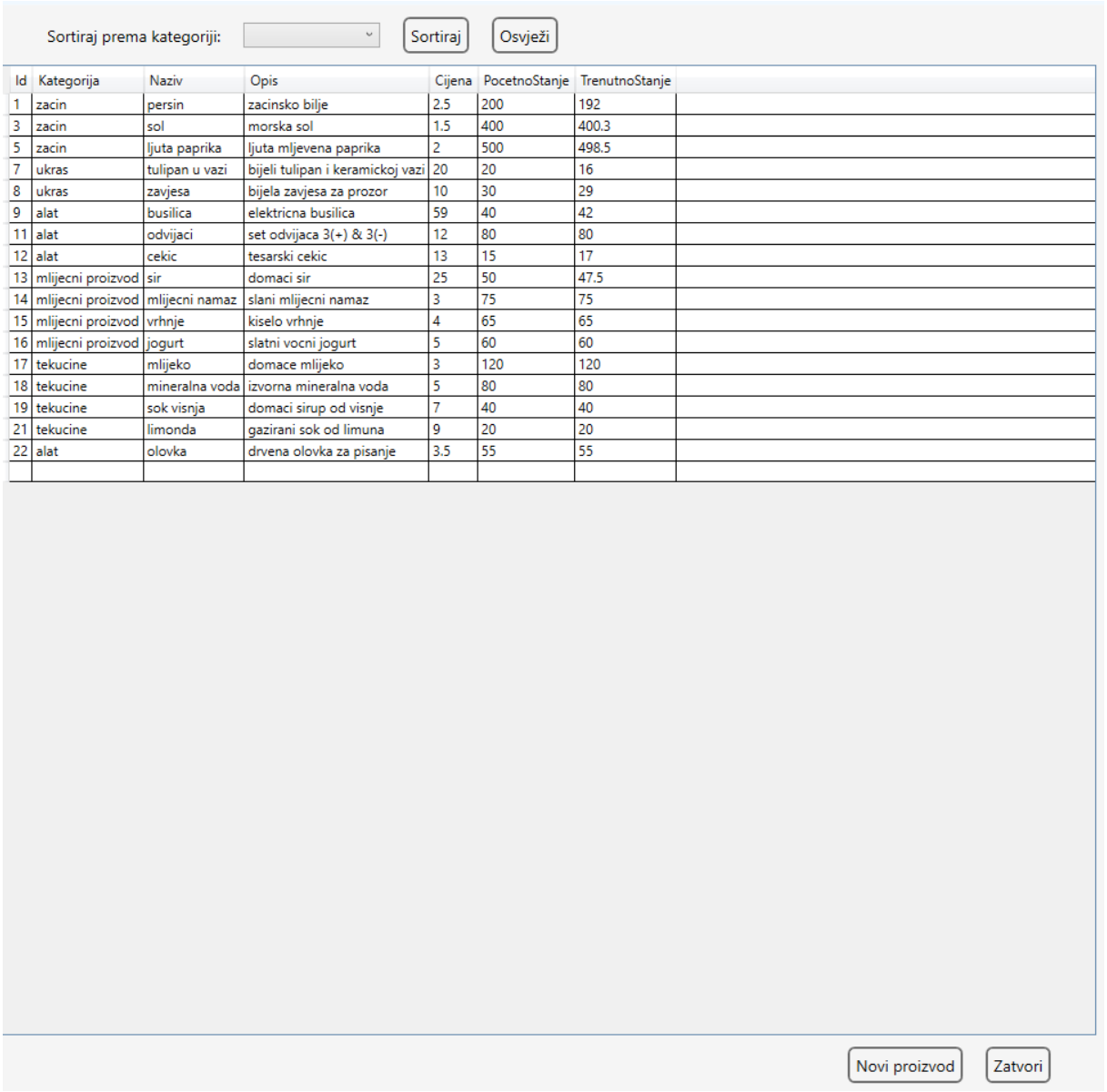
Također se može primijetiti kako je prikaz dokumenata otvoren samo na jednoj (lijevoj) strani početnog zaslona. To je zato što se u desnom zaslonu može otvoriti prikaz detalja odabranog dokumenta. To se čini tako što odabiremo dokument i pritisnemo gumb *Detalji dokumenta*.



Slika 19. prikaz detalja dokumenta (samostalna izrada)

Na ovom prikazu koji je otvoren na desnoj strani početnog prikaza pored prikaza svih dokumenata su prikazani detalji odabranog dokumenta. Može se vidjeti kako se prikazuju svi detalji vezani uz taj dokument kao što su broj dokumenta (id dokumenta u bazi), datum njegovog kreiranja, korisnik koji ga je kreirao i ukupna cijena.

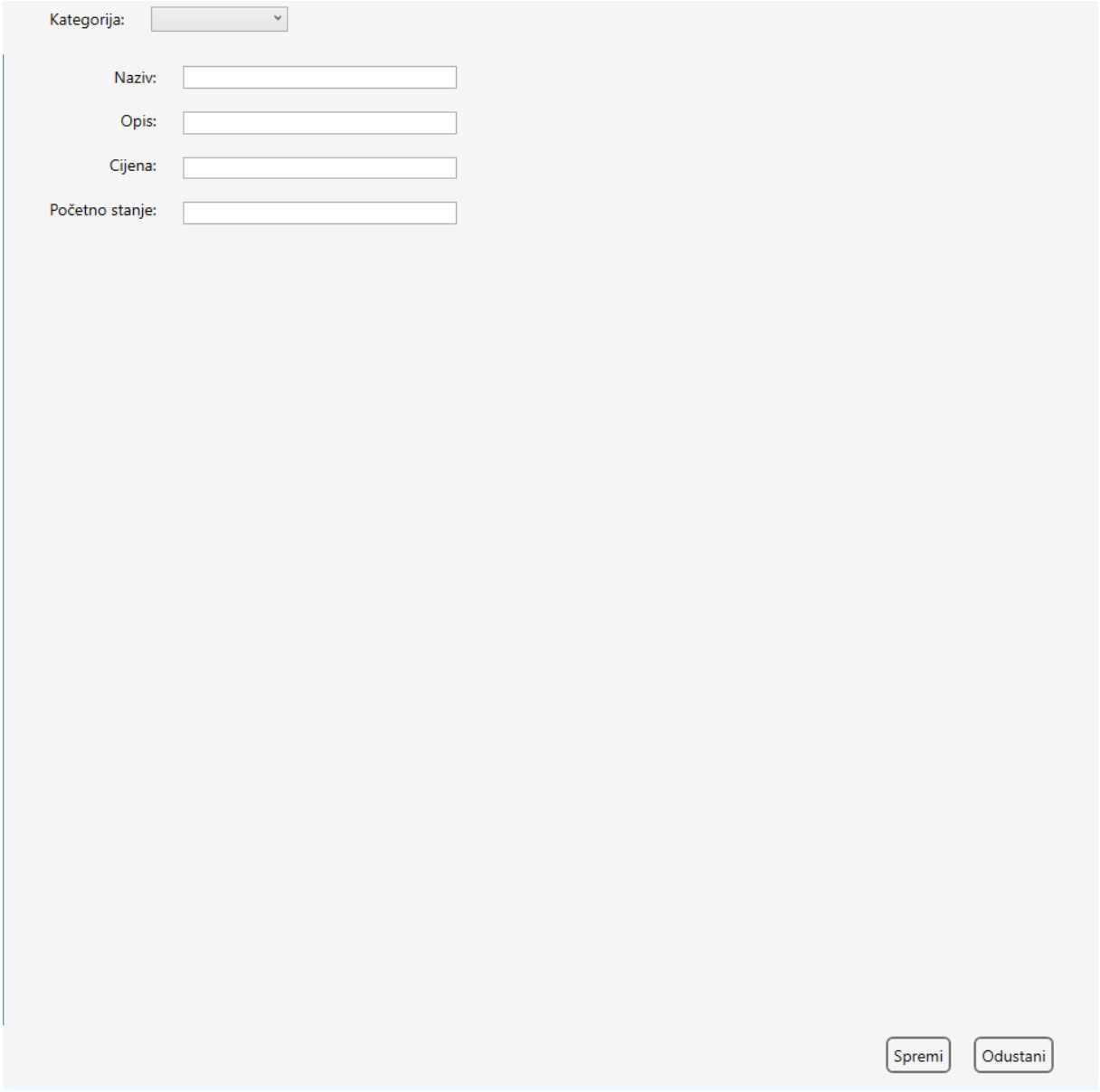
U tablici se također prikazuju proizvodi koji se nalaze na dokumentu. Uz svaki proizvod je i njegova ukupna cijena za odabranu količinu te i sama količina.



Id	Kategorija	Naziv	Opis	Cijena	PocetnoStanje	TrenutnoStanje
1	zacin	persin	zacinsko bilje	2.5	200	192
3	zacin	sol	morska sol	1.5	400	400.3
5	zacin	ljuta paprika	ljuta mljevena paprika	2	500	498.5
7	ukras	tulipan u vazi	bijeli tulipan i keramickoj vazi	20	20	16
8	ukras	zavjesa	bijela zavjesa za prozor	10	30	29
9	alat	busilica	elektricna busilica	59	40	42
11	alat	odvijaci	set odvijaca 3(+) & 3(-)	12	80	80
12	alat	cekic	tesarski cekic	13	15	17
13	mlijecni proizvod	sir	domaci sir	25	50	47.5
14	mlijecni proizvod	mlijecni namaz	slani mlijecni namaz	3	75	75
15	mlijecni proizvod	vrhnje	kiselo vrhnje	4	65	65
16	mlijecni proizvod	jogurt	slatni vocni jogurt	5	60	60
17	tekucine	mlijeko	domace mlijeko	3	120	120
18	tekucine	mineralna voda	izvorna mineralna voda	5	80	80
19	tekucine	sok visnja	domaci sirup od visnje	7	40	40
21	tekucine	limonda	gazirani sok od limuna	9	20	20
22	alat	olovka	drvena olovka za pisanje	3.5	55	55

Slika 20. prikaz proizvoda (samostalna izrada)

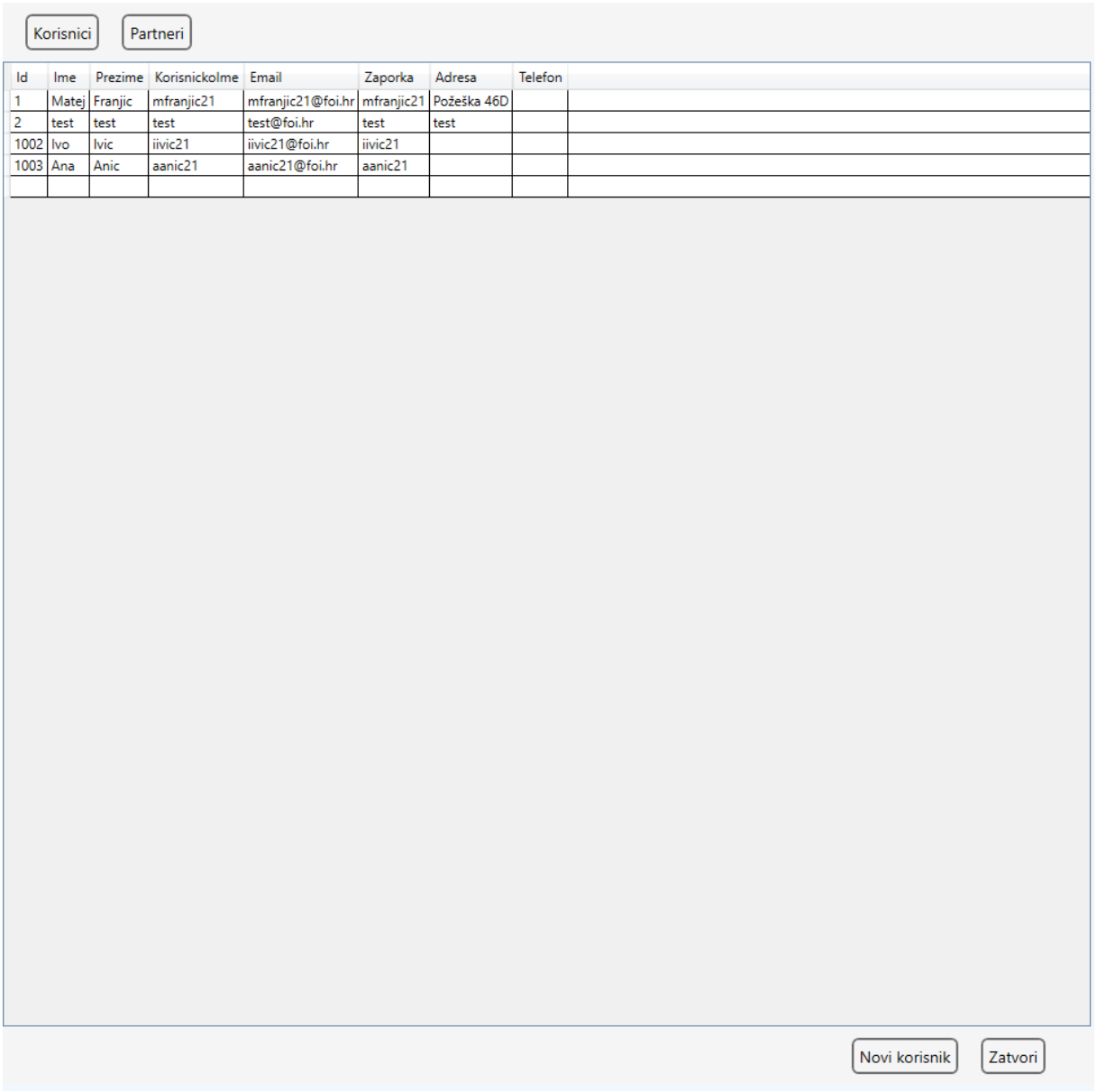
Pritiskom na gumb *Proizvodi* na početnom zaslonu s lijeve strane se otvara popis svih proizvoda koji su na skladištu. Sve proizvode možemo sortirati prema kategoriji kako bi se olakšao prikaz. U tablici se uz svaki proizvod prikazuje njegov opis, cijena, početno i trenutno stanje.



Slika 21. kreiranje novog proizvoda (samostalna izrada)

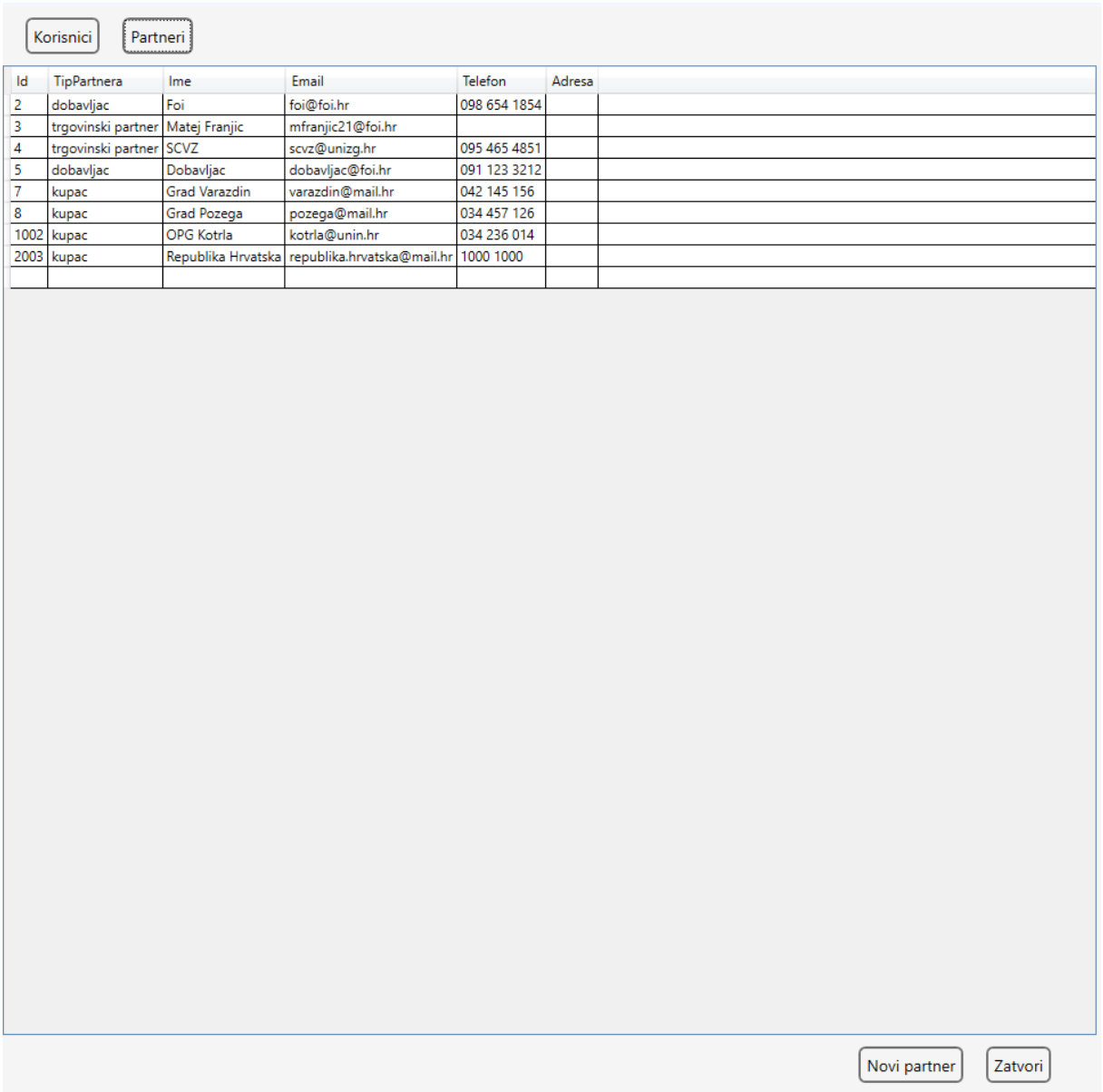
Klikom na gumb *Novi proizvod* koji se nalazi na prikazu sa prethodne slike otvara se prikaz za kreiranje novog proizvoda. Ovdje korisnik unosi sve podatke potrebne za kreiranje proizvoda od odabira kategorije iz padajućeg izbornika do početnog stanja koji se tekstualni unos (TextBox u WPF-u).

Pritiskom na tipku *Spremi* proizvod se pohranjuje u bazu podataka te je u buduće dostupan za pregled na prikazu dokumenata.



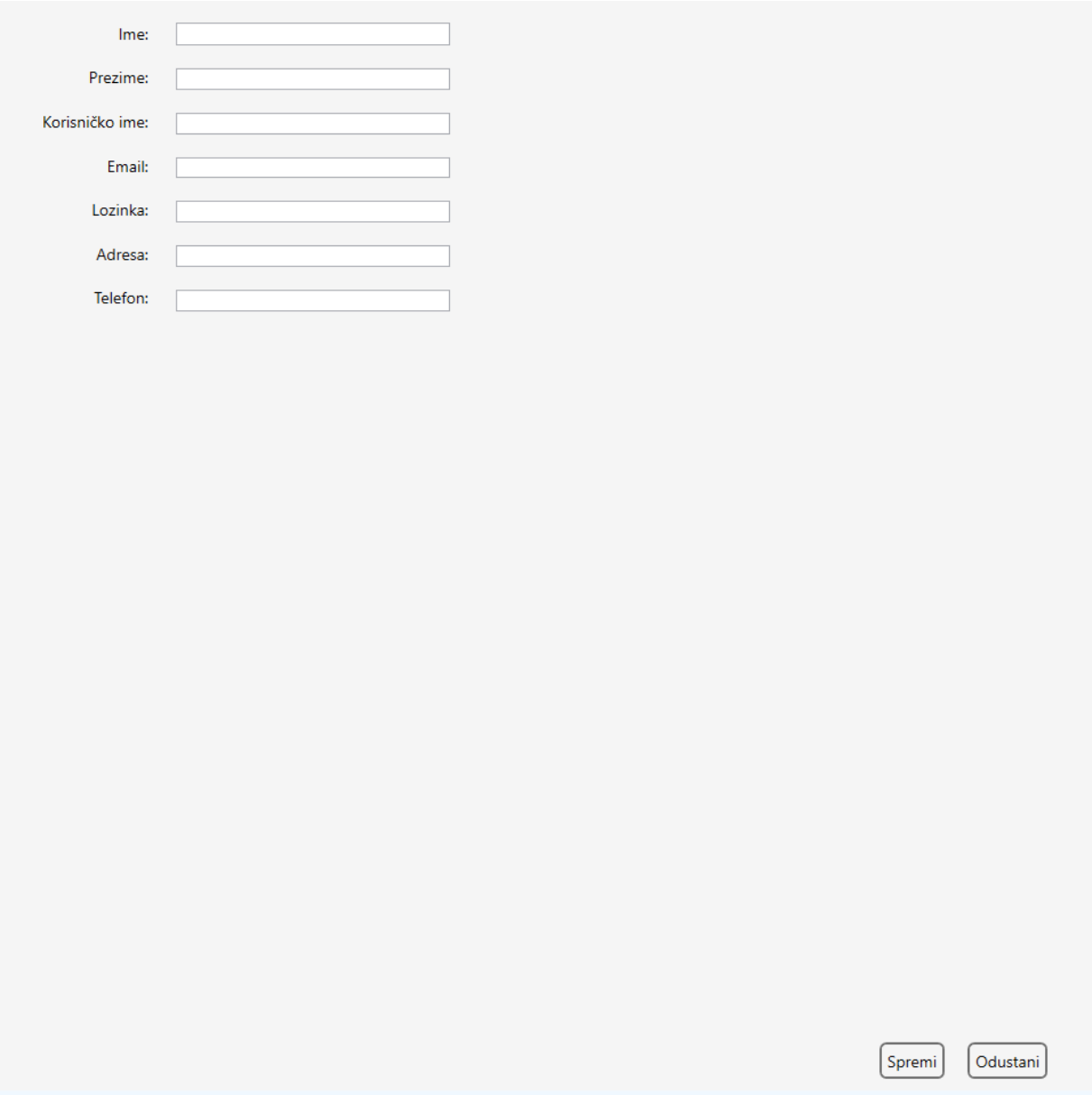
Slika 22. prikaz korisnika

Pritiskom gumba *Korisnici* otvara se prikaz svih registriranih korisnika. Ovdje se mogu vidjeti svi njihovi podaci u tabličnom prikazu. Također osim korisnika na ovom prikazu je dostupan i prikaz partnera koji se radi tako što se odabire gumb *Partneri* i zatim se prikaz prepravlja u sljedeći prikaz:



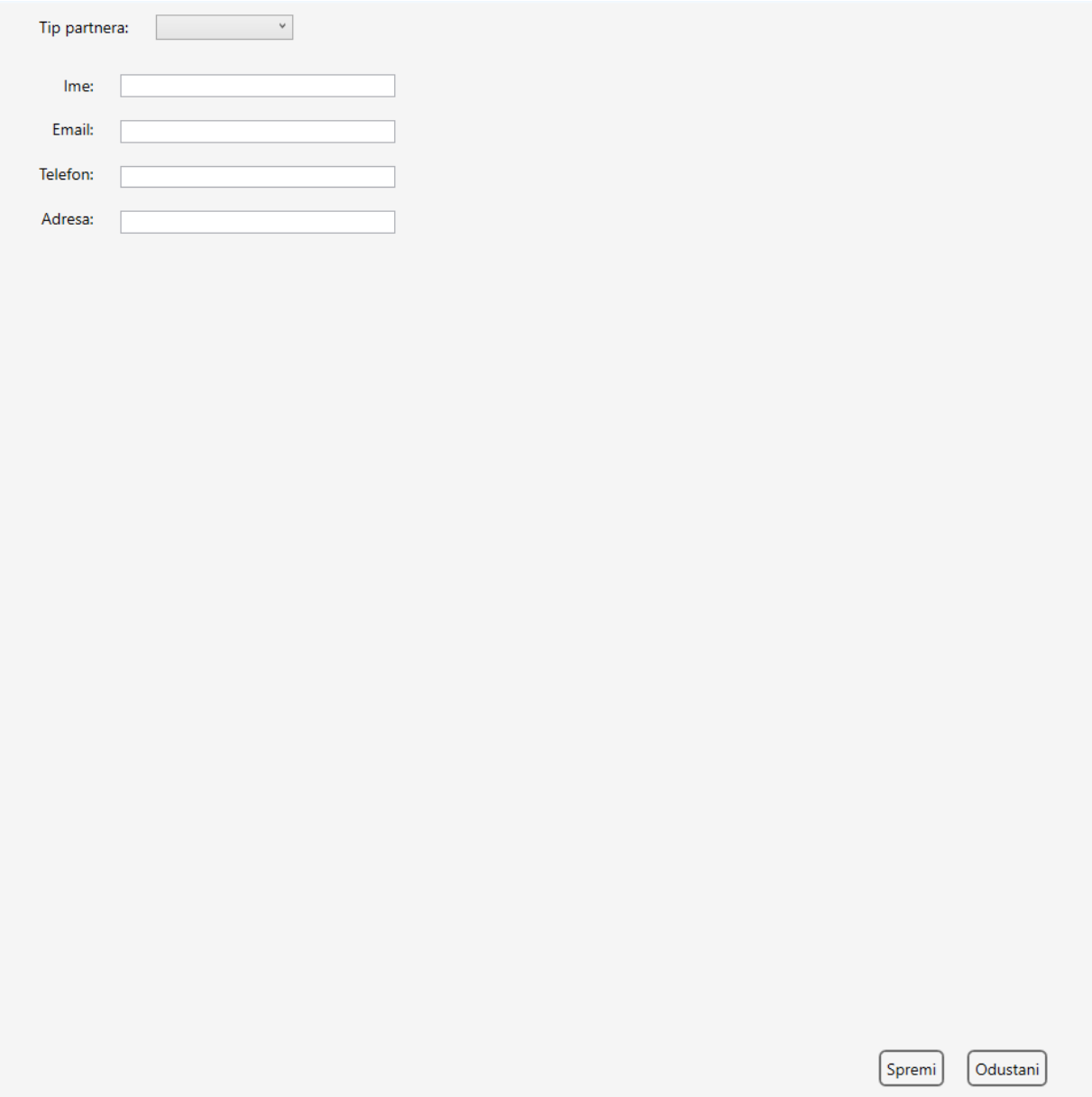
Slika 23. prikaz partnera (samostalna izrada)

Nešto što se također da primijetiti je gumb *Novi partner,* odnosno *Novi korisnik* koji se nalazi na tim prikazima. Pritiskom na njega otvara se novi prikaz za dodavanje novog partnera, odnosno korisnika ovisno o tome koji tablični prikaz korisnik gleda.



Slika 24. dodavanje korisnika (samostalna izrada)

Na ovom prikazu registriramo novog korisnika u aplikaciji koji će se kasnije moći prijaviti i koristiti aplikaciju za sve navedene svrhe. Dok na sljedećem prikazu imamo sličan prikaz samo što je on za registraciju novih partnera.



Slika 25. dodavanje partnera (samostalna izrada)

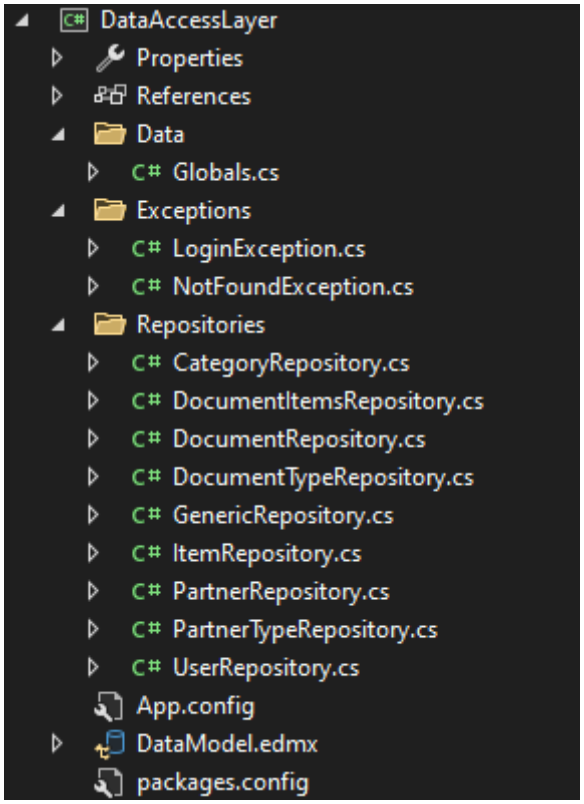
6.2. Način rada aplikacije – back-end

Sljedeće će biti prikazan back-end način funkcioniranja aplikacije. Aplikacije je izgrađena u višeslojnoj arhitekturi što znači da su prikaz i priprema prikaza (Presentation Layer), obrada podataka (Bussiness Logic Layer) i spoj s bazom i dohvat podataka (Data Access Layer) odvojeni.

6.2.1. Data access layer (DAL)

Najbitniji dio ovog dijela su repozitoriji koji sadrže sam LINQ kod za rad sa bazom podataka. Primjer jednog LINQ repozitorija je UserRepository koji služi za rad s korisnicima.

Osim repozitorija postoji i klasa *Globals* koja je definirana na razini cijelog projekta i služi za pospremanje varijabli koje se mogu dohvatiti iz bilo kojeg dijela programa



Slika 26. prikaz strukture datoteke DAL (samostalna izrada)

U ovom sloju se također nalazi i popis Exceptionsa (iznimaka). Iznimke služe za slučaje kada pokušamo izvršiti funkciju unutar „try – catch“ mehanizma za obradu iznimki. „Try – catch“ funkcionira na način da u „try“ dijelu pokušava se izvršiti funkcija te ukoliko ona nije uspješna funkcionalnost „baca iznimku“ koju „catch“ dio „hvata“ i obrađuje iznimku.

Primjer je funkcija *Login* koja pokušava se prijaviti u sustav, a u slučaju pogrešnih podataka za prijavu funkcija *GetUserByUserName* baca grešku o netočnosti informacija:

```
private void BtnLogin_Click(object sender, RoutedEventArgs e)
{
    try
    {
        var user = userRepository.GetUserByUserName(TxtUserName.Text);
        if(user.zaporka == TxtPassword.Password)
        {
            MessageBox.Show("Prijava uspješna", "Prijava");
            Globals.User = user;
            FrmMain main = new FrmMain();
            main.Show();
            this.Close();
        } else
        {
            MessageBox.Show("Kriva lozinka", "Prijava");
        }
    } catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Greška");
    }
}
```

Slika 28. try-catch funkcija Login (samostalna izrada)

```
2 references
public korisnik GetUserByUserName(string phrase)
{
    var query = from c in Entities
                where c.korisnicko_ime == phrase
                select c;

    if (!query.Any())
    {
        throw new LoginException("Korisnik nije pronađen");
    }

    return query.FirstOrDefault();
}
```

Slika 27. try-catch funkcija GetUserByUserName (samostalna izrada)

```
public class UserRepository : GenericRepository<korisnik>
{
    9 references
    public UserRepository() : base(new DataEntities())
    {
    }

    2 references
    public override int Update(korisnik entity, bool saveChanges = true)
    {
        var user = Entities.SingleOrDefault(c => c.id == entity.id);

        if (user == null)
        {
            return 0;
        }

        user.ime = entity.ime;
        user.prezime = entity.prezime;
        user.korisnicko_ime = entity.korisnicko_ime;
        user.email = entity.email;
        user.zaporka = entity.zaporka;
        user.adresa = entity.adresa;
        user.telefon = entity.telefon;

        if (saveChanges)
        {
            return SaveChanges();
        } else
        {
            return 0;
        }
    }
```

Slika 29. korisnički repozitorij i update funkcija (samostalna izrada)

```
    public korisnik GetUserByUserName(string phrase)
    {
        var query = from c in Entities
                    where c.korisnicko_ime == phrase
                    select c;

        if (!query.Any())
        {
            throw new LoginException("Korisnik nije pronađen");
        }

        return query.FirstOrDefault();
    }

    1 reference
    public korisnik GetUserByEmail(string phrase)
    {
        var query = from c in Entities
                    where c.email == phrase
                    select c;

        return query.FirstOrDefault();
    }
    1 reference
    public korisnik GetUserById(int id)
    {
        var query = from c in Entities
                    where c.id == id
                    select c;

        return query.FirstOrDefault();
    }

    1 reference
    public IQueryable<String> GetAllNames()
    {
        var query = from c in Entities
                    select (c.ime + " " + c.prezime);

        return query;
    }
}
```

Slika 30. funkcije UserRepozitorija (samostalna izrada)

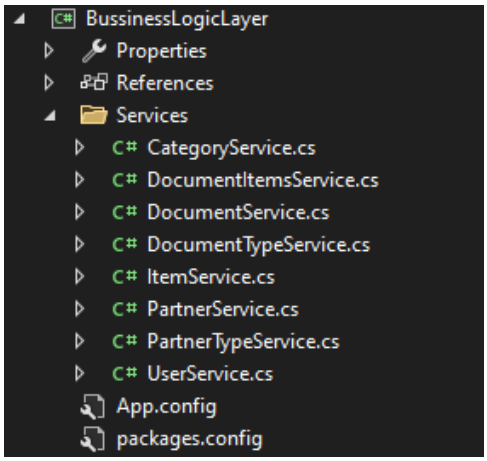
Na ovom repozitoriju se može primijetiti kao on nasljeđuje GenericRepository koji sadrži funkcije vezane uz generičke stvari poput dodavanja, dohvaćanja i brisanja objekta iz baze, ali i ažuriranje koje nije implementirano. Zbog toga se u svakom repozitoriju koji ga nasljeđuje on mora ponovo definirati.

Osim funkcije ažuriranja ovdje su definirane i funkcije dohvaćanja po korisničkom imenu, dohvaćanja po emailu, dohvaćanja po id-u i dohvaćanje svih imena.

Svaka tablica baze ima svoj repozitorij namješten kako bi radio za tu tablicu. Ovdje također možemo jasnije vidjeti sintaksu LINQ koda te se s lakoćom i minimalnim poznavanjem C# i LINQ i engleskog jezika da zaključiti što rade i kako rade. U funkciji dohvaćanja po

korisničkom imenu također vidimo i primjer maloprije objašnjene „try – catch“ metode u kojoj funkcija „baca“ grešku koja se kasnije „hvata“.

6.2.2. Bussiness Logic Layer (BLL)



Slika 31. prikaz strukture datoteke BLL (samostalna izrada)

Bussiness Logic Layer samom strukturom je vrlo jednostavniji od DAL-a, a servisi spremljeni u njemu imaju funkciju obrade podataka dohvaćenih iz DAL-a, te prosljeđivanje u Prezentacijski sloj (Presentation Layer).

```
public class UserService
{
    3 references
    public List<korisnik> GetUsers()
    {
        using (var repo = new UserRepository())
        {
            return repo.GetAll().ToList();
        }
    }

    0 references
    public korisnik GetUsersUserName(string username)
    {
        using (var repo = new UserRepository())
        {
            return repo.GetUserByUserName(username);
        }
    }
    0 references
    public korisnik GetUsersEmail(string email)
    {
        using (var repo = new UserRepository())
        {
            return repo.GetUserByEmail(email);
        }
    }
    2 references
    public korisnik GetUsersById(int id)
    {
        using (var repo = new UserRepository())
        {
            return repo.GetUserById(id);
```

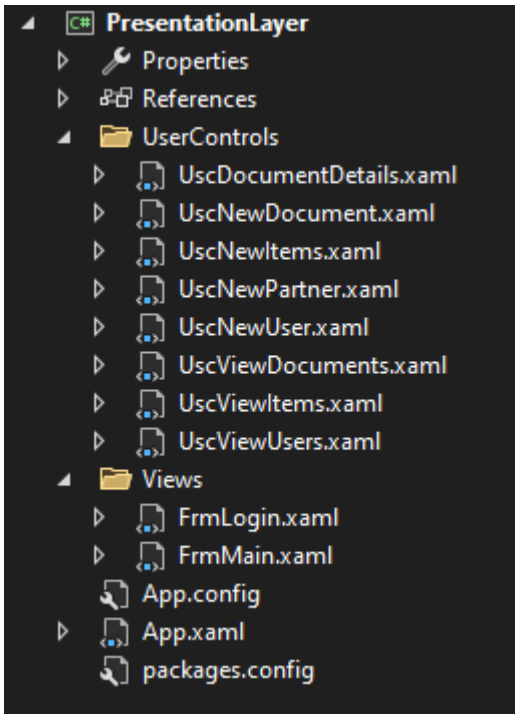
Slika 32. korisnički servis (samostalna izrada)

```
    public List<String> GetAllNames()
    {
        using (var repo = new UserRepository())
        {
            return repo.GetAllNames().ToList();
        }
    }

    1 reference
    public bool AddUser(korisnik user)
    {
        bool isSuccessful = false;
        using (var repo = new UserRepository())
        {
            int affectedRows = repo.Add(user);
            isSuccessful = affectedRows > 0;
        }
        return isSuccessful;

    }

    0 references
    public bool RemoveUser(korisnik user)
    {
        bool isSuccessful = false;
        using (var repo = new UserRepository())
        {
            int affectedRows = repo.Remove(user);
            isSuccessful = affectedRows > 0;
        }
        return isSuccessful;
    }

    0 references
    public bool UpdateUser(korisnik user)
    {
        bool isSuccessful = false;
        using (var repo = new UserRepository())
        {
            int affectedRows = repo.Update(user);
            isSuccessful = affectedRows > 0;
        }
        return isSuccessful;
    }
}
```

Slika 33. korisnički servis drugi dio (samostalna izrada)

Može se vidjeti kako servis koristi funkcije korisničkog repozitorija te nakon dohvaćanja rezultata iz njega obrađuje ih kako bi bili spremni za korištenje u ostatku aplikacije.

6.2.3. Presentation Layer (PL)



Slika 34. struktura datoteke PL-a (samostalna izrada)

Prezentacijski sloj sadrži dvije glavne vrste datoteka, a to su WPFovi Windowsi (Fmr prefiks) i User kontrole (USC prefiks). Windowsi služe za prikaz u zasebnom „prozoru“ koji su prethodno prikazani kao *Login* i početni zaslon *Main*. U njima se kasnije prikvače korisničke kontrole koje se same po sebi ne prikazuju već služe da se prikažu u prozoru.

Izglede samih prozora i korisničkih kontrola u front-endu su bile prikazane, a izgled back-enda u XAML-u i C#-u izgledaju ovako:

```
public partial class FrmLogin : Window
{
    readonly UserRepository userRepository;

    1 reference
    public FrmLogin()
    {
        InitializeComponent();

        userRepository = new UserRepository();
        TxtUserName.Focus();
        TxtUserName.SelectAll();
    }

    1 reference
    private void BtnLogin_Click(object sender, RoutedEventArgs e)
    {
        try
        {
            var user = userRepository.GetUserByUserName(TxtUserName.Text);
            if(user.zaporka == TxtPassword.Password)
            {
                MessageBox.Show("Prijava uspješna", "Prijava");
                Globals.User = user;
                FrmMain main = new FrmMain();
                main.Show();
                this.Close();
            } else
            {
                MessageBox.Show("Kriva lozinka", "Prijava");
            }
        } catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Greška");
        }
    }

    1 reference
    private void BtnClose_Click(object sender, RoutedEventArgs e)
    {
        this.Close();
    }
}
```

Slika 35. prikaz C# dijela Login prozora (samostalna izrada)

U C# dijelu back-enda prozora za login može se vidjeti obrada događaja kada se pritisnu gumbi prikazani na prozoru. Pritiskom gumbom *Login* se pokušava metodom „try – catch“ prijaviti u sustav. Kao što je već prikazana funkcija dohvaćanja po korisničkom imenu koja „baca“ iznimku, ona se ovdje u slučaju neuspješne prijave „hvata“ i ispisuje se greška i korisnika se traži ponovo unošenje podataka.

```xaml
<Window.Resources>
    <Style x:Key="RowHeight" TargetType="RowDefinition">
        <Setter Property="Height" Value="auto"/>
    </Style>
    <Style x:Key="ColumnWidth" TargetType="ColumnDefinition">
        <Setter Property="Width" Value="auto"/>
    </Style>
    <Style x:Key="LoginButton" TargetType="Button">
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="VerticalAlignment" Value="Center"/>
        <Setter Property="Background" Value="WhiteSmoke"/>
        <Setter Property="BorderThickness" Value="2"/>
        <Setter Property="Padding" Value="5"/>
        <Style.Resources>
            <Style TargetType="Border">
                <Setter Property="CornerRadius" Value="5"/>
            </Style>
        </Style.Resources>
    </Style>
</Window.Resources>
```

Slika 37. definiranje resursa u Login prozoru (samostalna izrada)

```xaml
<Grid Background="AliceBlue">
    <Grid.RowDefinitions>
        <RowDefinition Style="{StaticResource RowHeight}"/>
        <RowDefinition Style="{StaticResource RowHeight}"/>
        <RowDefinition Style="{StaticResource RowHeight}"/>
    </Grid.RowDefinitions>
    <Grid Grid.Row="1" Background="WhiteSmoke">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Style="{StaticResource ColumnWidth}"/>
            <ColumnDefinition Style="{StaticResource ColumnWidth}"/>
        </Grid.ColumnDefinitions>
        <StackPanel Grid.Column="1" HorizontalAlignment="Left" VerticalAlignment="Top">
            <TextBox x:Name="TxtUserName" Margin="20, 10, 40, 10" TextWrapping="Wrap" Width="140" F
            <PasswordBox x:Name="TxtPassword" Margin="20, 10, 40, 10" Width="140" InputScope="Passw
        </StackPanel>
        <StackPanel Grid.Column="0" HorizontalAlignment="Left" VerticalAlignment="Top">
            <TextBlock  Margin="40, 10, 0, 10" Text="Korisničko ime:" Width="auto" FontSize="14"/>
            <TextBlock Text="Lozinka:" Margin="40, 10, 0, 10" FontSize="14"/>
        </StackPanel>
    </Grid>
    <TextBlock Grid.Row="0" HorizontalAlignment="Left" Margin="40, 20, 40, 10" TextWrapping="Wrap"

    <Grid Grid.Row="2" HorizontalAlignment="Right">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Style="{StaticResource ColumnWidth}"/>
            <ColumnDefinition Style="{StaticResource ColumnWidth}"/>
        </Grid.ColumnDefinitions>
        <Button x:Name="BtnClose" Grid.Column="0" Content="Izlaz" HorizontalAlignment="Right" Verti
        <Button x:Name="BtnLogin" Grid.Column="1" Content="Prijava" HorizontalAlignment="Right" Mar
    </Grid>
</Grid>
```

Slika 36. XAML kod Login prozora (samostalna izrada)

7. Zaključak

Ovaj završni rad obuhvatio je proučavanje i praktičnu primjenu ključnih komponenti iz .NET razvojnog okvira, uključujući C#, LINQ i WPF. Kroz implementaciju aplikacije za upravljanje skladištem, prikazane su sposobnosti C# jezika u izradi stolnih aplikacija te je demonstrirana implementacija spomenutih tehnologija u razvoju aplikacije.

Praktična primjena LINQ-a pokazala je učinkovitost rada s bazama podataka bez potrebe za pisanjem SQL koda, što je pojednostavilo manipulaciju podacima. S druge strane, WPF je omogućio napredno oblikovanje korisničkog sučelja, osiguravajući odvajanje logike od prikaza putem XAML jezika, čime je postignuta veća modularnost i održivost aplikacije.

Rezultati ovog rada pokazuju da su .NET tehnologije, unatoč pojavi novih razvojnih platformi, i dalje vrlo relevantne za razvoj složenih stolnih aplikacija. Razvoj aplikacije također je omogućio procjenu funkcionalnosti i fleksibilnosti ovih alata u stvarnim scenarijima, te se može zaključiti da C#, u kombinaciji s LINQ-om i WPF-om, pruža robustan i efikasan temelj za izradu poslovnih aplikacija.

Popis literature

Chauhan, S. (29. 5 2024). *Comparing LINQ with Stored Procedure*. Dohvaćeno iz ScholarHat: https://www.scholarhat.com/tutorial/linq/comparing-linq-with-stored-procedure

Hejlsberg Anders, T. M. (2008). The C# Programming Language. U T. M. Hejlsberg Anders, *The C# Programming Language* (str. 1-2).

Liberty, J. (2005). Programming C#: Building .NET Applications with C#. U J. Liberty, *Programming C#: Building .NET Applications with C#* (str. 11).

Market Share of Microsoft SQL Server. (2024). Dohvaćeno iz 6Sense: https://6sense.com/tech/database/microsoft-sql-server-market-share

Microsoft. (6. 2 2023). *Desktop Guide (WPF .NET)*. Dohvaćeno iz Microsoft Learn: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-8.0

Microsoft. (15. 12 2023). *Microsoft Learn*. Dohvaćeno iz Language Integrated Query (LINQ): https://learn.microsoft.com/en-us/dotnet/csharp/linq/

Microsoft. (n.d.). *ASP.NET documentation*. Dohvaćeno iz Microsoft Learn: https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0

Microsoft. (n.d.). *Build beautiful web apps with Blazor*. Dohvaćeno iz Microsoft .NET: https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor

Microsoft. (n.d.). *Xamarin documentation*. Dohvaćeno iz Microsoft Learn: https://learn.microsoft.com/en-us/previous-versions/xamarin/

Tran, T. (22. 2 2024). *Top 20 Most Popular Programming Languages in 2024 & Beyond*. Dohvaćeno iz Orient: https://www.orientsoftware.com/blog/most-popular-programming-languages

Unity. (n.d.). *Unity Manual*. Dohvaćeno iz Unity Documentation: https://docs.unity3d.com/Manual/index.html

Using C# for cross-platform development. (n.d.). Dohvaćeno iz WordPress Basics of Web Development: https://basicsofwebdevelopment.wordpress.com/2015/12/08/using-c-for-cross-platform-development/

Popis slika

Slika 1. Udio tržišta SSMS (6sense.com) 2
Slika 2. Model baze podataka (samostalna izrada) 3
Slika 3. tablica "dokument" (samostalna izrada) 4
Slika 4. tablica "korisnik" (samostalna izrada) 4
Slika 5. tablica "tip_dokumenta" (samostalna izrada) 4
Slika 6. tablica "partner" (samostalna izrada) 5
Slika 7. tablica "tip_partnera" (samostalna izrada) 5
Slika 8. tablica "stavke_dokumenta" (samostalna izrada) 5
Slika 9. tablica "proizvod" (samostalna izrada) 6
Slika 10. tablica "kategorija" (samostalna izrada) 6
Slika 11. tržišni udio C# jezika (orientsoftware.com) 7
Slika 12. Popis C# poslova (Linkedin.com) 8
Slika 13. primjer LINQ jednostavne funkcije (samostalna izrada) 10
Slika 14. primjer C# jednostavne funkcije (samostalna izrada) 10
Slika 15. prikaz Login zaslona (samostalna izrada) 13
Slika 16. prikaz početnog zaslona (samostalna izrada) 14
Slika 17. prikaz kreiranja dokumenta (samostalna izrada) 15
Slika 18. prikaz dokumenata (samostalna izrada) 16
Slika 19. prikaz detalja dokumenta (samostalna izrada) 17
Slika 20. prikaz proizvoda (samostalna izrada) 18
Slika 21. kreiranje novog proizvoda (samostalna izrada) 19
Slika 22. prikaz korisnika 20
Slika 23. prikaz partnera (samostalna izrada) 21
Slika 24. dodavanje korisnika (samostalna izrada) 22
Slika 25. dodavanje partnera (samostalna izrada) 23
Slika 26. prikaz strukture datoteke DAL (samostalna izrada) 24
Slika 28. try-catch funkcija Login (samostalna izrada) 25
Slika 27. try-catch funkcija GetUserByUserName (samostalna izrada) 25
Slika 29. korisnički repozitorij i update funkcija (samostalna izrada) 26
Slika 30. funkcije UserRepozitorija (samostalna izrada) 27
Slika 31. prikaz strukture datoteke BLL (samostalna izrada) 28
Slika 32. korisnički servis (samostalna izrada) 28
Slika 33. korisnički servis drugi dio (samostalna izrada) 29
Slika 34. struktura datoteke PL-a (samostalna izrada) 30

Slika 35. prikaz C# dijela Login prozora (samostalna izrada).....31
Slika 36. XAML kod Login prozora (samostalna izrada).....32
Slika 37. definiranje resursa u Login prozoru (samostalna izrada).....32