Sustav upravljanja skladištem u C#

Žebčević, Danijel

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Danijel Žebčević

Sustav upravljanja skladištem u C#

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Danijel Žebčević

JMBAG: 0016149470

Studij: Informacijski sustavi

Sustav upravljanja skladištem u C#

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, rujan 2023.

Danijel Žebčević

Izjava o izvornosti

Izjavljujem da je moj završni 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

Završni rad detaljno opisuje proces izrade C# WPF aplikacije naziva InventoryPro. Aplikacija radi sa NoSQL bazom podataka naziva MongoDB. Za pristup aplikaciji je potrebno prijaviti se u aplikaciju sa već postojećim računom koji je moguće kreirati putem registracije. Aplikacija sadrži repozitorij klasu koja služi za komuniciranje sa bazom, dok je prezentacijski dio odvojen u zasebnu mapu gdje se nalaze svi prozori. Unutar istog projekta se nalaze i pomoćne entitetne klase koje predstavljaju objekte koji se spremaju u bazu. Aplikacija može pomoći studentima razviti njihove vještine programiranja u C# te pružiti mogućnost rada sa novim tipom baze podataka koji se sve više koristi u današnjem svijetu.

Ključne riječi: C#, MongoDB, NoSQL, WPF

Sadržaj

1. Uvod

C# je jedan od najpopularnijih programskih jezika danas. Nudi korisnicima razne kreativne mogućnosti kao što su izrada web aplikacija ili izrada desktop aplikacija. U ovom slučaju, radi se o izradi desktop aplikacije koja služi za upravljanje skladištem koja za bazu koristi MongoDB. Aplikacija može imati razne primjene u svakodnevnom životu. Razni poduzetnici koji imaju svoje obrte i imaju različite narudžbe i dostave mogu koristiti ovu aplikaciju kako bi olakšali upravljanje svojim poduzećem.

Rađena je u Visual Studio razvojnom okruženju, a tip aplikacije je WPF. Korišten je WPF jer je to novija tehnologija u odnosu na starije Windows Forms tehnologije.

Bit će opisano što je uopće MongoDB, koje su razlike NoSQL i SQL baza te i sam postupak izrade opisane aplikacije.

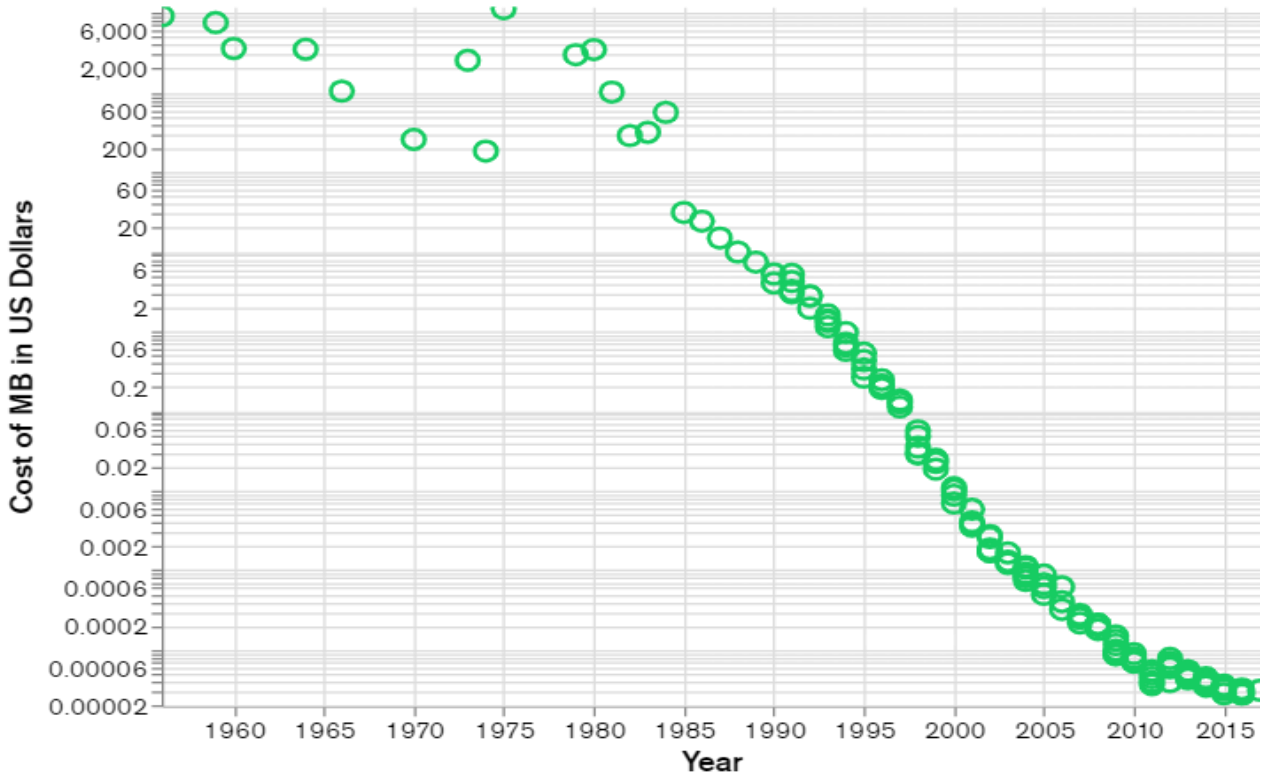
2. MongoDB baza podataka

2.1 Definicija MongoDB-a

MongoDB je NoSQL program otvorenog koda koji služi za upravljanje bazom podataka. U ovom slučaju se NoSQL koristi kao alternativa klasičnim relacijskim bazama podataka. Stoga se MongoDB koristi kao alat koji može upravljati informacijama koje su spremljene u obliku dokumenata.

2.2 Što je NoSQL

Termin „NoSQL baza podataka“ se često koristi kako bi se referenciralo na bilo koju ne relacijsku bazu podataka. Neki smatraju da „NoSQL“ znači „non SQL“ dok drugi smatraju da ono znači „not only SQL“. U svakom slučaju, postoji konsenzus da termin označava bazu podataka koja podatke sprema u obliku drukčijem od tablica.



Slika 1. Trošak spremanja MB tijekom vremena [1]

Povijest NoSQL baza podataka kreće u 2000-tim godinama kad su se troškovi spremanja podataka drastično smanjili. Kako su se troškovi smanjili, količina podataka za spremanje se povećavala. Podaci koji su se trebali spremati su bili u različitim oblicima: strukturirani, polustrukturirani i polimorfni. Tako različiti oblici podataka se teško spremaju u klasične baze podataka te su se stoga pojavile NoSQL baze koje olakšavaju upravljanje takvim podacima.

Ovo su glavne značajke svih NoSQL baza:

- Fleksibilne sheme
- Horizontalno skaliranje
- Brzi upiti
- Jednostavno korištenje

Osim toga, postoje 4 vrste NoSQL baza podataka:

- Baze podataka dokumenata: pohranjuju podatke u dokumente slične JSON objektima. Svaki dokument sadrži parove polja i vrijednosti. Ovdje pripada MongoDB
- Baze podataka ključ-vrijednost: jednostavnija vrsta baze podataka gdje svaka stavka sadrži ključeve i vrijednosti
- Pohrane širokih stupaca: pohranjuju podatke u tablice retke i dinamične stupce
- Baze podataka grafova: pohranjuju podatke u čvorovima i rubovima

2.3 Razlike između NoSQL i SQL

Glavna razlika između ove dvije vrste bazi podataka je način na koji se podaci modeliraju u bazi. Možemo to objasniti kroz primjer: želimo pohraniti informacije o korisniku kao što su ime prezime itd. te informacije o njegovim hobijima. U relacijskoj bazi bismo stvorili dvije tablice na sljedeći način:

Users

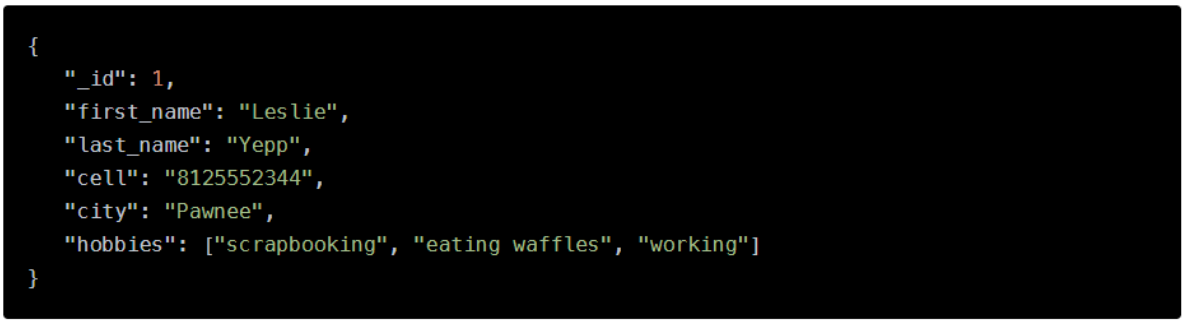
ID	first_name	last_name	cell	city
1	Leslie	Yepp	8125552344	Pawnee

Hobbies

ID	user_id	hobby
10	1	scrapbooking
11	1	eating waffles
12	1	working

Slika 2. Primjer relacijske baze [1]

S druge strane, spremanje tih podataka u NoSQL bi izgledalo ovako:



Slika 3. Primjer NoSQL baze [1]

Kako bi se dohvatili svi podaci o korisniku i hobijima, može se dohvatiti samo jedan dokument, nisu potrebna spajanja pa su upiti brži.

2.4 Zašto NoSQL?

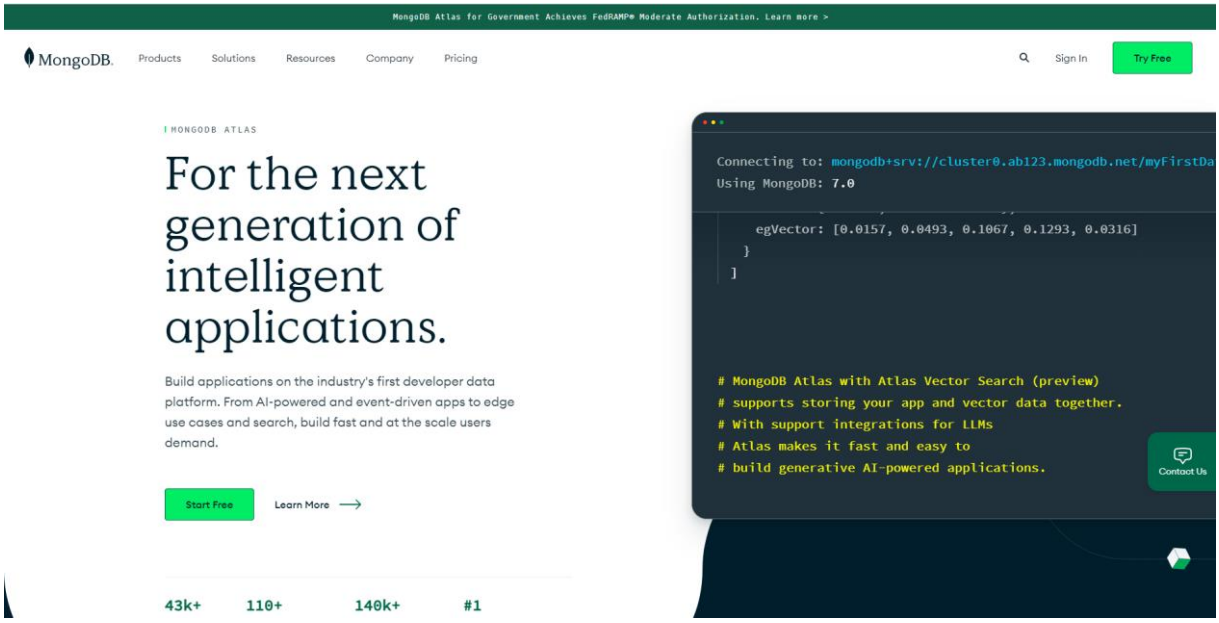
Prilikom odabira baze podataka, sljedeći čimbenici utječu na odabir NoSQL baze:

- Agilni razvoj brzim tempom
- Pohranjivanje strukturiranih i polustrukturiranih podataka
- Velike količine podataka
- Zahtjevi za scale-out arhitekturom
- Moderne aplikativne paradigme poput mikroservisa

Kao što je i već rečeno, NoSQL mogu spremiti ogromne količine podataka koje ne moraju biti strukturirane te sam dohvati podataka je jednostavan.

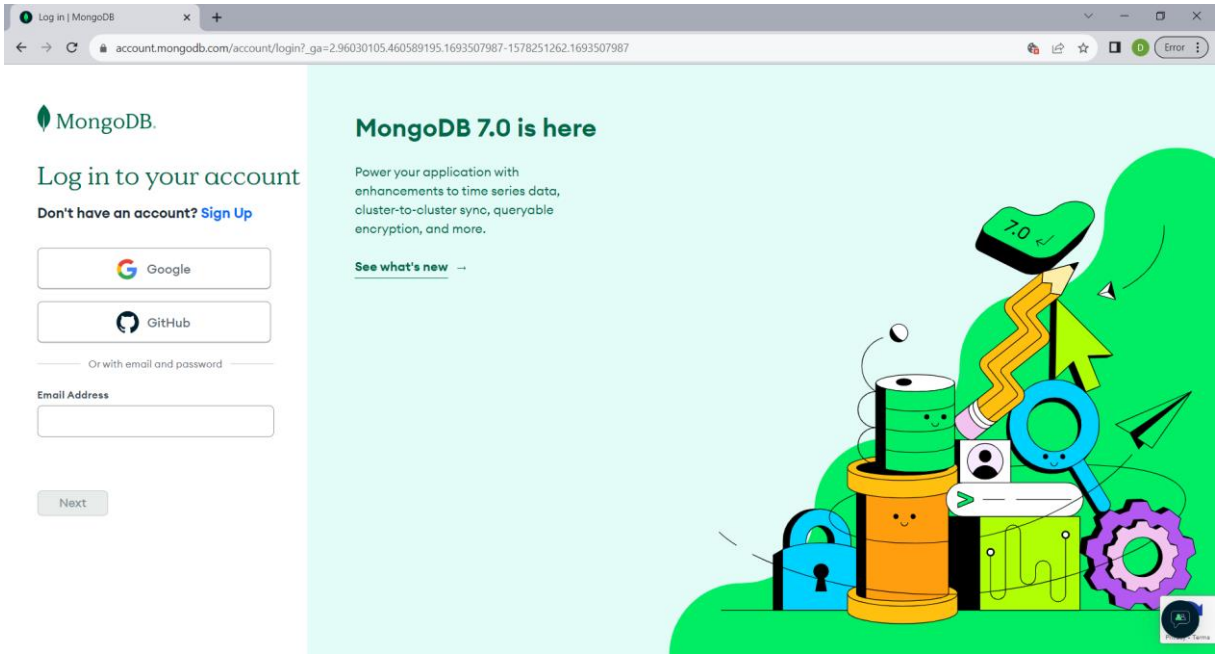
2.5 Registracija u MongoDB

Proces prijave u MongoDB je vrlo jednostavan. Moguće je kreirati korisnički račun tako da se korisnik registrira sa već postojećim Google računom.



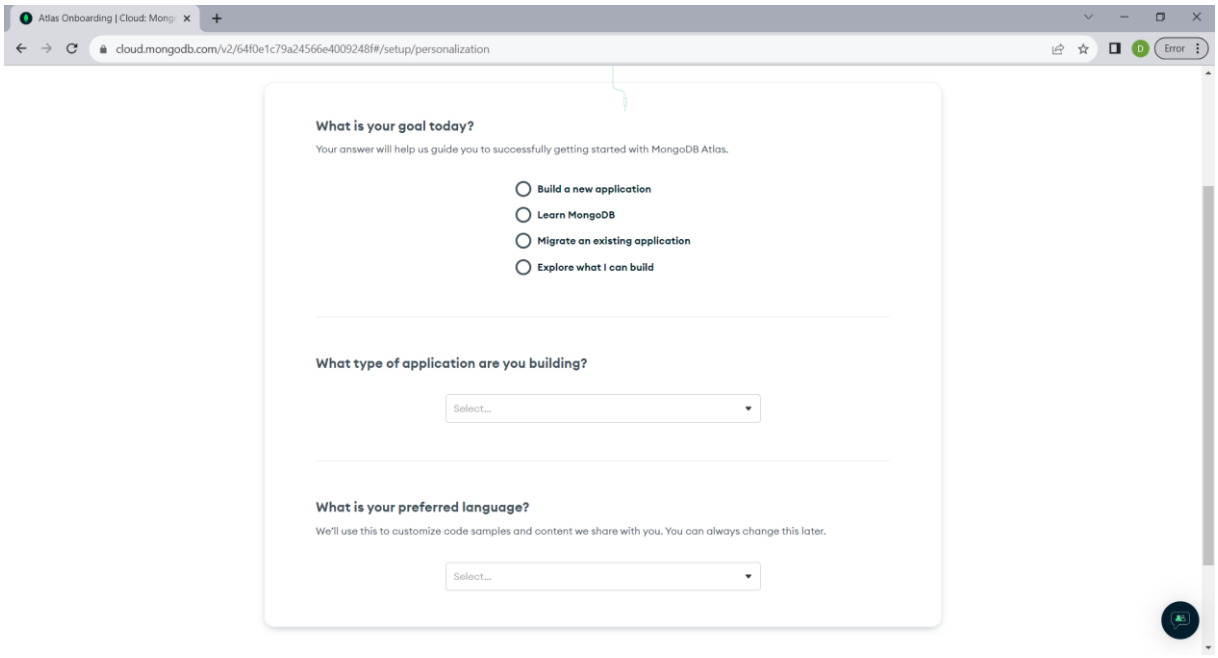
Slika 4. Početni ekran MongoDB-a [2]

Može se vidjeti početna stranica MongoDB-a.



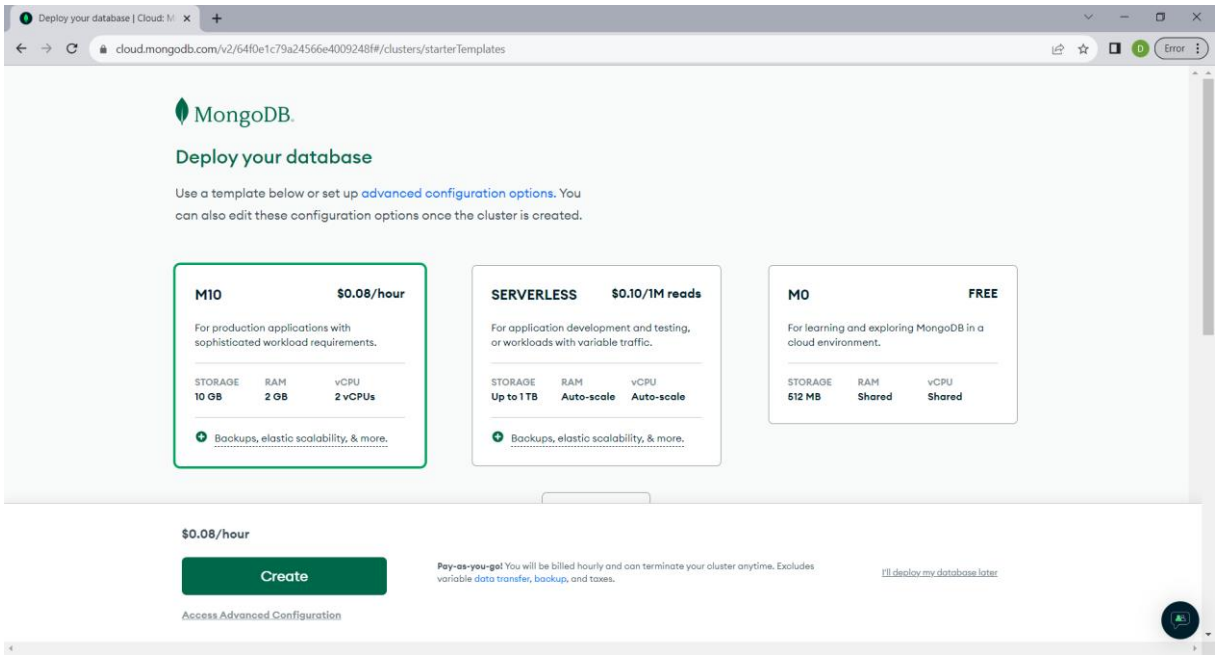
Slika 5. Prijava u MongoDB [2]

Nakon što se odabere opcija sign in, može se odabrati google račun opcija.



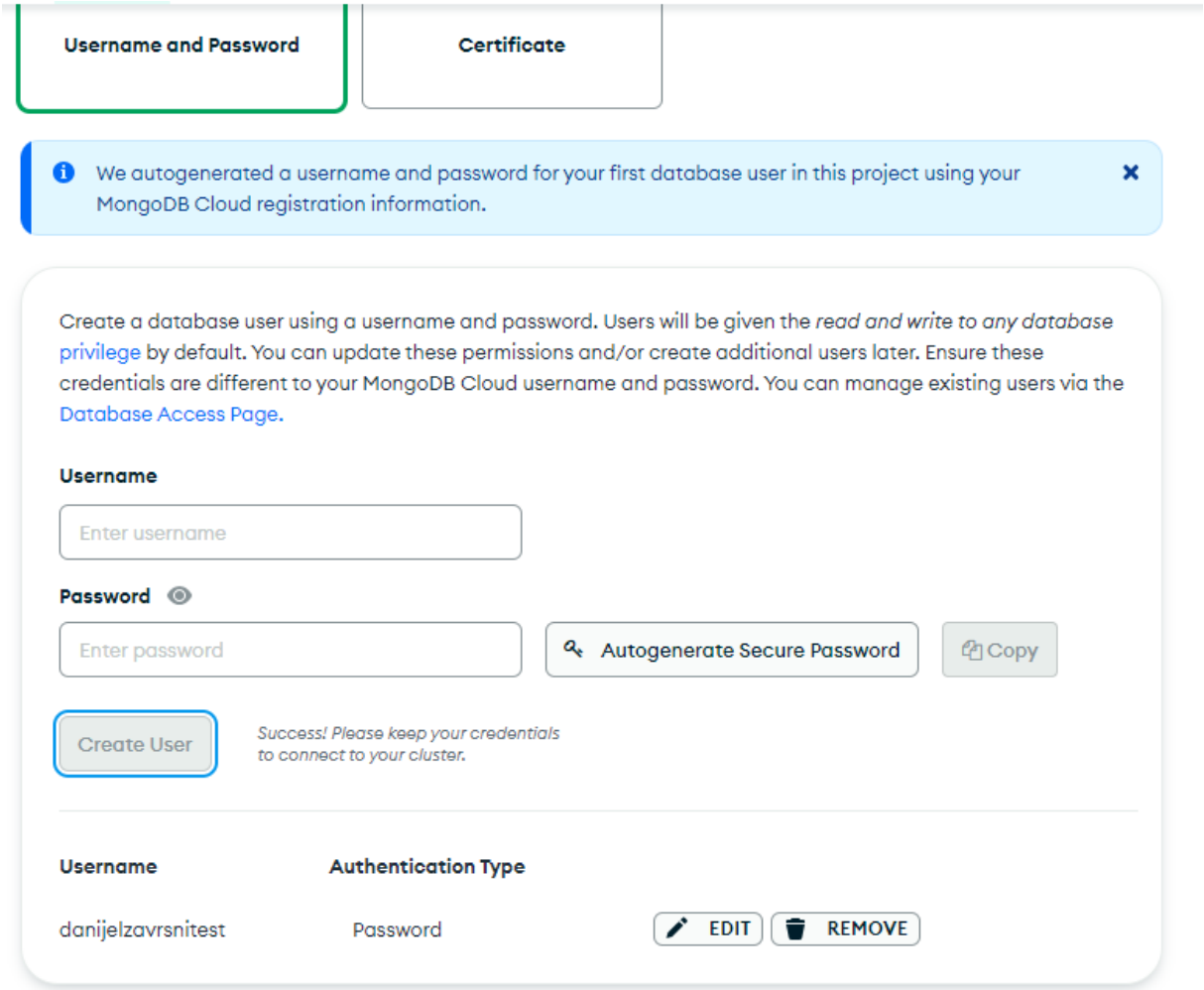
Slika 6. Ispunjavanje ankete [2]

Potom se od korisnika traži da se ispuni kratka anketa čiji je cilj saznati zašto se koristi MongoDB.



Slika 7. Odabiranje plaćanja baze [2]

Nakon što se ispuni anketa, odabire se koju opciju MongoDB-a se želi koristiti. Za ovu aplikaciju je korištena besplatna verzija. Korisnik dobiva 512 MB besplatnog spremišta za podatke.



Slika 8. Kreiranje novog korisnika [2]

Nakon toga se upiše IP adresa te se doda novi korisnik. Te opcije se vrlo lako mogu izvršiti, polja se mogu automatski popuniti kao što se može vidjeti na slici iznad. Na kraju korisnik ima svoj Cluster kojem može pristupiti i raditi s njime.

3. WPF aplikacije

Windows Presentation Foundation je besplatni grafički podsustav otvorenog koda koji je izvorno razvio Microsoft za prikaz korisničkih sučelja u aplikacijama temeljenim na sustavu Windows. Prvi put je objavljen kao dio .NET Framework 3.0 2006. godine.

WPF koristi XAML, jezik temeljen na XML-u za definiranje i povezivanje različitih elemenata sučelja. WPF kao glavni cilj ima objediniti niz elemenata korisničkog sučelja, kao što su 2D/3D renderiranje, fiksni dokumenti itd. koji se povezuju i manipuliraju na temelju različitih događaja.

U arhitekturi WPF aplikacija često se koristi koncept Model-View-ViewModel (MVVM). Ova arhitektura olakšava upravljanje podacima, logikom i prikazom korisničkog sučelja putem odvojenih komponenata. MVVM povećava testabilnost, skalabilnost i održavanje koda, pružajući jasnu strukturu aplikacije.

Grafika se prikazuje pomoću Direct3D što omogućuje prikaz složenije grafike i prilagođenih tema. Osim toga, koristi se vektorska grafika koja omogućuje skaliranje većine kontrola i elemenata bez gubitka kvalitete(pikselizacija). Stoga, to je jedna od glavnih prednosti koje WPF aplikacije pružaju u odnosu na standardne Windows Forms aplikacije koje se već dugo smatraju zastarjele. Osim toga, sam proces dizajna grafičkog sučelja je puno zahvalniji kod WPF jer je on i sam novija aplikacija te stoga postoji puno više mogućnosti uređenja.

4. Izrada InventoryPro-a

4.1 Prijava i registracija

Proces izrade InventoryPro započeo je naravno sa kreiranjem WPF aplikacije. Korišten je WPF kao platforma za izradu budući da se to smatra novijom tehnologijom. Nakon kreiranja, krenio je proces dizajniranja početnog ekrana za prijavu u aplikaciju. Kao što je već rečeno, WPF aplikacije koriste XAML jezik za definiranje elemenata na prozoru. U tom jeziku, postoji sustav mreže, odnosno „Grid“ kako bi se lakše moglo upravljati elementima na prozoru. Tako se u prozoru za prijavu, odnosno „login“ može napraviti mreža sa 4 stupca i 10 redaka. To izgleda ovako:

```
<Grid>
     <Grid.ColumnDefinitions>
         <ColumnDefinition Width="20" />
         <ColumnDefinition Width="auto" />
         <ColumnDefinition Width="*" />
         <ColumnDefinition Width="20" />
     </Grid.ColumnDefinitions>
     <Grid.RowDefinitions>
         <RowDefinition Height="5"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="auto"/>
         <RowDefinition Height="20"/>
     </Grid.RowDefinitions>
 </Grid>
```

Može se vidjeti da je XAML jezik dosta sličan HTML-u.

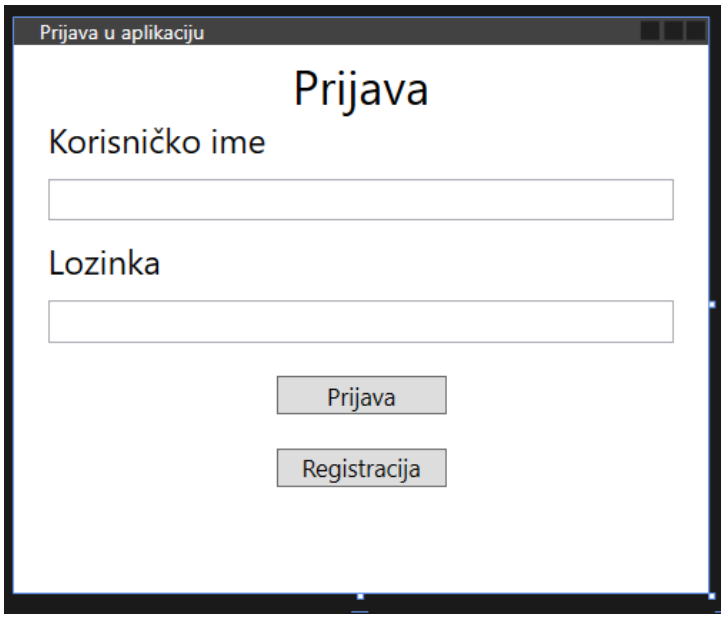
Kao vrijednost širine ili visine možemo zadati neku „hardcode“ vrijednost u pikselima, kao npr. za prvi stupac sam stavio da je širina 20 piksela. Auto vrijednost znači da će širina stupca automatski se proširiti ovisno o tome koje elemente dodamo. Ako pak stavimo vrijednost zvjezdice(*), to znači da će taj stupac zauzeti sav preostali prostor na prozoru. Elementi se

mogu definirati i izvan Grid-a, ali ja ću ih sve stavljati unutar budući da je tako proces dizajniranja najlakši, što će se i vidjeti u nastavku.

Nakon što se definiraju stupci i retci, mogu se definirati i ostali elementi. WPF aplikacije su slične Windows Forms aplikacijama, stoga će postojati TextBox-ovi i TextBlock-ovi(labele).

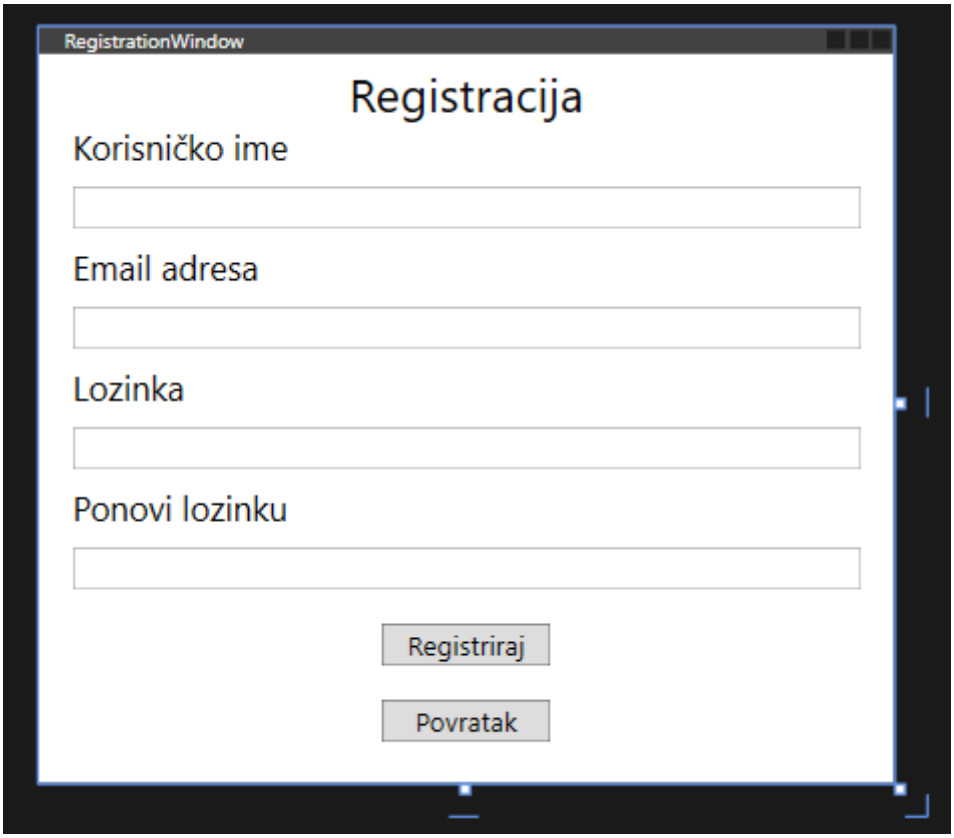
```
<TextBlock HorizontalAlignment="Center" Grid.Column="1" Grid.Row="1"
Grid.ColumnSpan="2" Text="Prijava" FontSize="28"/>
        <TextBlock Grid.Column="1" Grid.Row="2" Text="Korisničko ime"
FontSize="20"/>
        <TextBox x:Name="usernameText" Grid.Column="1" Grid.Row="3"
Grid.ColumnSpan="2" Height="25" Margin="0, 10"/>
        <TextBlock Grid.Column="1" Grid.Row="4" Text="Lozinka"
FontSize="20"/>
        <TextBox x:Name="passwordText" Grid.Column="1" Grid.Row="6"
Grid.ColumnSpan="2" Height="25" Margin="0, 10"/>


        <Button x:Name="loginButton" Content="Prijava" Grid.Column="1"
Grid.Row="7" Grid.ColumnSpan="2" MaxWidth="100" Margin="0, 10"
FontSize="14" Click="loginButton_Click"/>
        <Button x:Name="registrationButton" Content="Registracija"
Grid.Column="1" Grid.Row="8" Grid.ColumnSpan="2" MaxWidth="100" Margin="0,
10" FontSize="14" Click="registrationButton_Click"/>
```

TextBlock označava običan tekst, dakle služi kao labela. TextBox služi za čitanje unosa podataka od korisnika. Unutar ovog koda mogu se dodati i ostale postavke kao što margine itd., no zasad to ne treba. Na Button elementima je stavljen „Click“ tag, označava funkciju u kodu koja će se pozvati kad se klikne gumb. Gumb ima puno različitih događaja koji se vežu za njega, no u ovom slučaju potrebam je samo treba click event. Također se može vidjeti da neki elementi imaju „ColumnSpan“ definiran. To znači da se element može protezati kroz nekoliko stupaca, a taj broj se definira kao gore navedeno. Brojanje definiranih stupaca i redaka se radi kao u array-evima u programiranju: kreće se od nule i ide se dalje. To znači da he prvi redak na mjestu 0, drugi na mjestu 1 itd. Nakon svega ovoga, ekran za login izgleda ovako:



Slika 9. Prozor za prijavu (samostalna izrada 2023.)

Nakon toga je išao proces dizajniranja prozora za registraciju. Proces je isti kao i za login prozor, korišten je grid, stupci, retci i ostali elementi. Nakon dizajniranja taj ekran izgleda ovako:



Slika 10. Prozor za registraciju (samostalna izrada 2023.)

Klasa user sadrži sljedeća svojstva:

```
public class User
    {

        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]

        public string Id { get; set; }

        public string Username { get; set; }

        public string Password { get; set; }

        public string Email { get; set; }

        public string Salt { get; set; }
    }
```

Sva gore navedena svojstva je potrebno zapisati tijekom kreiranja novog korisnika.

Poslije ovoga je trebalo napraviti da klikom na gumb „Registracija“ na prozoru za login, da se otvori prozor za registraciju. To je napravljeno tako da se samo kreira instanca klase i pozove metoda show nad tom klasom.

Ovako izgleda kod przozora za dodavanje novog korisnika u bazu:

```
    private async void registrationButton_Click(object sender,
RoutedEventArgs e)
        {
            MongoRepository repository = new MongoRepository();
            if (usernameText.Text != "" && emailText.Text != "" &
passwordText.Text != "")
            {
                if (passwordText.Text == password2Text.Text)
                {
                    repository.AddUser(usernameText.Text, emailText.Text,
passwordText.Text);
                    LoginWindow newWindow = new LoginWindow();
                    newWindow.Show();
                    this.Close();
                }
                else
                {
```

```
                MessageBox.Show("Lozinke moraju biti iste!");
            }
        }
        else
        {
            MessageBox.Show("Niste upisali sve podatke!");
        }
    }
```

Nakon što korisnik klikne na gumb za registraciju, provjerava se sadrže li sva polja nekakvu vrijednost. Nakon toga slijedi provjera jesu li te dvije unesene lozinke iste. Ako jesu, poziva se repozitorij metoda dodavanje korisnika. U slučaju da korisnik krivo unese podatke, prikaže se MessageBox. Linija this.Close služi kako bi se zatvorio već otvoreni prozor kako ne bi bilo previše prozora otvoreno odjednom, čime se poboljšava user experience.

Nakon toga slijedi kod za registraciju korisnika. Potrebno je prvo spojiti se sa bazom. Prvo se u NuGet paketima instalira MongoDB driver kako bi se mogli koristiti alati potrebni za korištenje MongoDB-a. Potom je kreirana klasa MongoRepository koja će služiti za rad s bazom. U toj klasi napravljena je metoda AddUser:

```
    public static string connectionString =
"mongodb+srv://dzebcevic:WaivXdKY77dCWhDk@cluster0.jn3wc2c.mongodb.net/";
        public static string databaseName = "InventoryProDB";
        public static MongoClient client = new
MongoClient(connectionString);
        public IMongoDatabase database = client.GetDatabase(databaseName);


        public async void AddUser(string username, string email, string
password)
        {
            string collectionName = "users";
            var collection = database.GetCollection<User>(collectionName);


            string salt = BCrypt.Net.BCrypt.GenerateSalt();
            string hashedPassword = BCrypt.Net.BCrypt.HashPassword(password
+ salt);


            var person = new User { Username = username, Email = email,
Password = hashedPassword, Salt = salt };
            await collection.InsertOneAsync(person);
        }
```

Proces spajanja sa MongoDB je jako jednostavan. Kreira se connectionString koji se može dobiti na njihovoj stranici. Potom se napravi string za dohvaćanje imena baze podataka, ime kolekcije te se kreira novog klijenta koji se spaja pomoću već navednog connectionString-a. U metodi AddUser se kreira varijabla collection koja je rezultat pozivanja GetCollection na database. Metodi se prosljedi ime collectiona te tip objekta koji se vraća, u ovom slučaju user.

Može se vidjeti da je i napravljeno hashiranje lozinke prilikom registracije tako da se dodaje sol na lozinku. Korištena je biblioteka Bcrypt.

Nakon toga, napravljen je proces prijave u aplikaciju. Dohvaćaju se svi korisnici iz baze sa tim korisničkim imenima što ubrzava proces nego da se moraju dohvatiti svi korisnici, te se uspoređuje unesena lozinka i korisničko ime sa već postojećim u bazi. Kod izgleda ovako:

```
public async Task<bool> LoginUser(string userProvidedPassword, string
userProvidedUsername)
        {
            string collectionName = "users";
            var userCollection =
database.GetCollection<User>(collectionName);
            var filter = Builders<User>.Filter.Eq("Username",
userProvidedUsername);
            var foundUsers = await
userCollection.Find(filter).ToListAsync();
            if (foundUsers.Count > 0)
            {
                string retrievedSalt = foundUsers[0].Salt;
                bool isPasswordCorrect =
BCrypt.Net.BCrypt.Verify(userProvidedPassword + retrievedSalt,
foundUsers[0].Password);
                if (isPasswordCorrect)
                {
                    return true;

                }
                else
                {
                    return false;
                }
            }
            else
            {
                return false;
            }
```

```
        }
```

Prilikom uspoređivanja moraju se usporediti soli. Ako se svi podaci podudaraju, znači da se korisnik može prijaviti u aplikaciju.

U formi za prijavu se poziva metoda iz repozitorija za login te ako se vrati true, otvara se glavni prozor aplikacije:

```
    private async void loginButton_Click(object sender, RoutedEventArgs e)
        {
            MongoRepository repository = new MongoRepository();
            bool passwordIsCorrect = false;

            if (usernameText.Text != "" && passwordText.Text != "")
            {
                passwordIsCorrect = await
repository.LoginUser(passwordText.Text, usernameText.Text);
                if (passwordIsCorrect)
                {
                    HomeWindow homeWindow = new HomeWindow();
                    homeWindow.Show();
                    this.Close();
                }
                else
                {
                    MessageBox.Show("Netočni podaci tijekom prijave!");

                }
            }
            else
            {
                MessageBox.Show("Niste upisali sve podatke!");
            }
        }
```

Prilikom prijave je potrebno provjeriti sadrže li polja za korisničko ime i lozinku nekakvu tekstualnu vrijednost. U slučaju da ne sadrže, ispiše se greška pomoću MessageBox-a.

4.2 Izrada Home prozora

Sljedeće je potrebno urediti početni prozor aplikacije tako da postoji navigacija za sve druge prozore koji će postojati. Slijedi XAML kod za početni prozor:

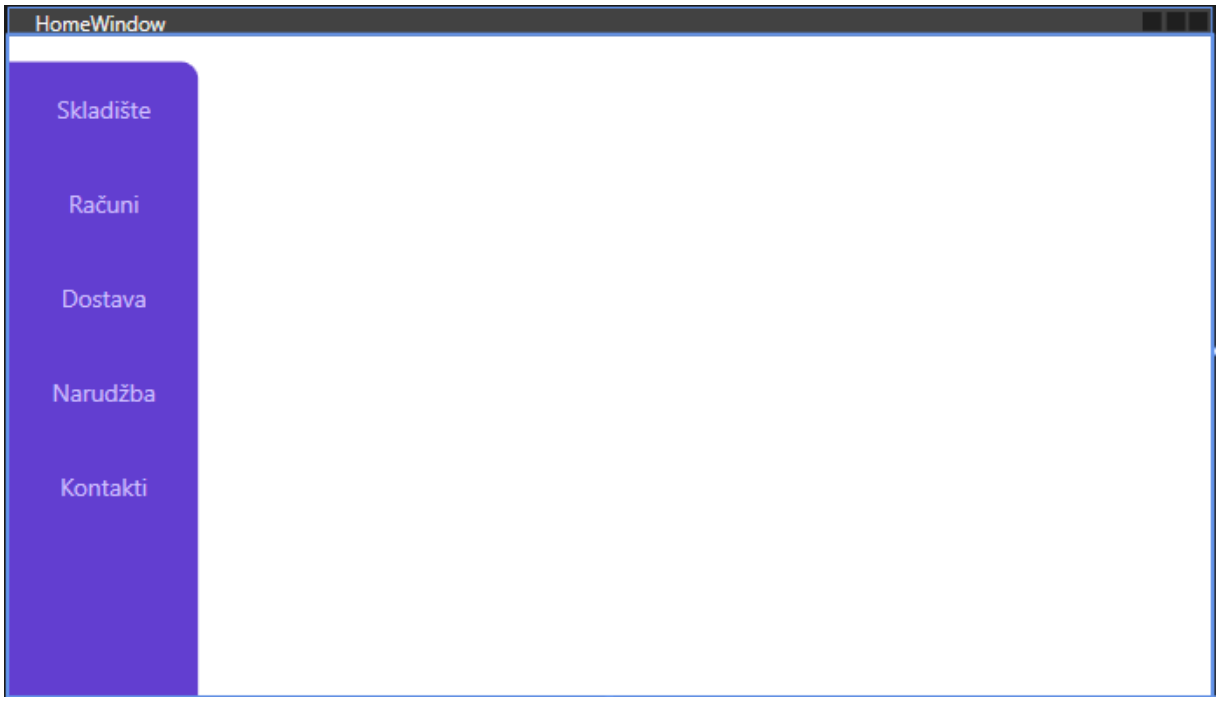
```
<Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="auto" />
            <ColumnDefinition Width="30" />
            <ColumnDefinition Width="160" />
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="160" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="15"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Border Grid.Row="1" Width="110" Grid.RowSpan="7"
Background="#623ed0" CornerRadius="0 10 0 0">


        </Border>


        <Button Style="{StaticResource menuButton}" x:Name="homeButton"
Content="Skladište" Grid.Column="0" Grid.Row="1"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="homeButton_Click"/>
        <Button Style="{StaticResource menuButton}" x:Name="billsButton"
Content="Računi" Grid.Column="0" Grid.Row="2"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="billsButton_Click"/>
        <Button Style="{StaticResource menuButton}" x:Name="deliveryButton"
Content="Dostava" Grid.Column="0" Grid.Row="3"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="deliveryButton_Click"/>
        <Button Style="{StaticResource menuButton}" x:Name="orderButton"
Content="Narudžba" Grid.Column="0" Grid.Row="4"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="orderButton_Click"/>
        <Button Style="{StaticResource menuButton}" x:Name="contactButton"
Content="Kontakti" Grid.Column="0" Grid.Row="5"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="contactButton_Click"/>
    </Grid>
```

U kodu se može vidjeti definiran broj redova i stupaca, kao i gumbovi koji će služiti za navigaciju. Svakom gumbu je potrebno dati ime, dodijeliti mu event koji će odrediti što će se dogoditi kad se klikne na njega. Osim toga, može se vidjeti da se koristio stil zvan menuButton.

U WPF aplikacijama postoje stilovi koji se mogu dodjeljivati elementima na prozoru. Stilovi su u ovom slučaju stavljeni u app.xaml file. Slijedi kod stila menuButton:

```
<Style x:Key="menuButton" TargetType="Button">
            <Setter Property="Background" Value="Transparent" />
            <Setter Property="Foreground" Value="#d0c0ff" />
            <Setter Property="FocusVisualStyle" Value="{x:Null}" />
            <Setter Property="Height" Value="35" />
            <Setter Property="Margin" Value="15 3" />
            <Setter Property="FontSize" Value="13" />
            <Setter Property="Background" Value="Transparent" />
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="Button">
                        <Border Background="{TemplateBinding Background}"
CornerRadius="7">
                            <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>

            <Style.Triggers>
                <Trigger Property="IsMouseOver" Value="True">
                    <Setter Property="Background" Value="#7b5cd6" />
                    <Setter Property="Foreground" Value="#ffffff" />
                </Trigger>
            </Style.Triggers>

        </Style>
```

Može se vidjeti da je proces pisanja stila poprilično jednostavan, slično je kao da korisnik svakom gumbu dodjeljuje svojstva i uređuje izgled, ovdje je samo taj kod odvojen i može se ponovno koristiti. Stavljene su margine, uređena je boja u ljubičastu koja se mijenja kad se mišem prelazi preko njega. Nakon što se taj stil dodijeli, početna stranica izgleda ovako:



Slika 11. Početni prozor aplikacije (samostalna izrada 2023.)

4.3 Rad sa proizvodima

Nakon što je napravljen početni prozor, može se napraviti prozor koji će služiti za prikaz svih proizvoda u skladištu.

Napravljena je klasa Product, koja predstavlja proizvod. Dodana su joj svojstva kao što su id, količina, naziv i količina.

```
public class Product
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]

    public string Id { get; set; }

    public string Name { get; set; }

    public float PricePerUnit { get; set; }

    public int Amount { get; set; }
}
```

U repozitoriju je napravljena metoda GetProducts koja dohvaća sve proizvode iz baze.

```
public async Task<List<Product>> GetProducts()
```

```
        {
            List<Product> products = new List<Product>();
            string collectionName = "products";
            var collection =
database.GetCollection<Product>(collectionName);
            var results = await collection.FindAsync(_ => true);

            foreach (var product in results.ToList())
            {
                products.Add(product);
            }
            return products;
        }
```

Može se primjetiti da metode ne vraćaju jednostavno bool ili List<Product> već vraćaju task sa tim vrijednostima. To je zato što asinkrone funkcije u C# moraju vraćati Task, a budući da se za rad sa podacima koristi await, moraju se koristiti asinkrone funkcije. Kako bi dobili vrijednost asinkrone funkcije, mora ih se također dozvati sa await ključnom riječi.

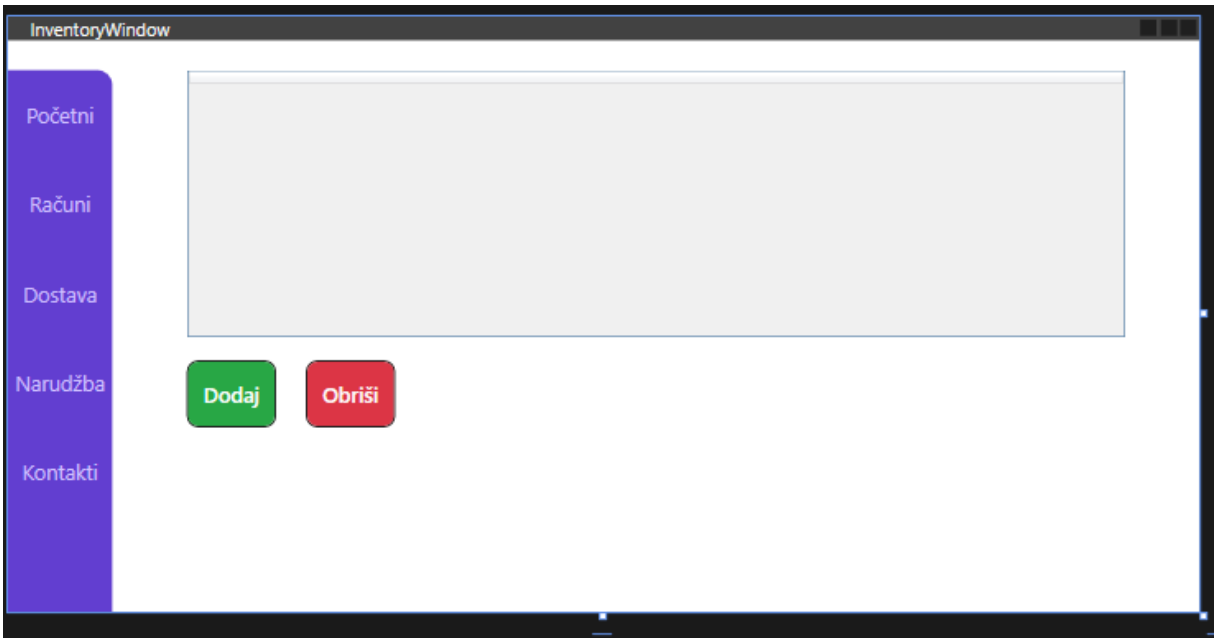
Također je odlučeno optimizirati arhitekturu aplikacije, stoga je napravljena mapa Windows u koju će se smještati svi prozori.

Prvo je potrebno dizajnirati početni Inventory prozor sa gumbovima za dodavanje i brisanje, tablicom, kao i navigacijski gumbovi kojima će korisnik moći navigirati do ostalih stranica aplikacije kao što to može na početnoj stranici. Ovako izgleda XAML kod nakon što se dodaju ti elementi:

```
<Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="auto" />
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="60" />
            <ColumnDefinition Width="20" />
            <ColumnDefinition Width="60" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="50" />

        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>

            <RowDefinition Height="20"/>
```

```
            <RowDefinition Height="60"/>
            <RowDefinition Height="60"/>
            <RowDefinition Height="60"/>
            <RowDefinition Height="60"/>
            <RowDefinition Height="60"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <DataGrid Name="dataGrid" Grid.Column="2" Grid.ColumnSpan="4"
Grid.Row="1" Grid.RowSpan="3" MinHeight="100" IsReadOnly="True"
AutoGenerateColumns="True" MouseDoubleClick="dataGrid_MouseDoubleClick" />
        <Button Style="{StaticResource SuccessButton}" x:Name="addButton"
Content="Dodaj" Grid.Column="2" Grid.Row="4" Margin="0,15,0,0"
Click="addButton_Click" />
        <Button Style="{StaticResource DangerButton}" x:Name="deleteButton"
Content="Obriši" Grid.Column="4" Grid.Row="4" Margin="0,15,0,0"
Click="deleteButton_Click" />


        <Border Grid.Row="1" Width="70" Grid.RowSpan="6"
Background="#623ed0" CornerRadius="0 10 0 0">
        </Border>



        <Button Style="{StaticResource menuButton}" x:Name="homeButton"
Content="Početni" Grid.Column="0" Grid.Row="1"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="homeButton_Click" />
        <Button Style="{StaticResource menuButton}" x:Name="billButton"
Content="Računi" Grid.Column="0" Grid.Row="2"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="billButton_Click" />
        <Button Style="{StaticResource menuButton}" x:Name="deliveryButton"
Content="Dostava" Grid.Column="0" Grid.Row="3"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="deliveryButton_Click" />
        <Button Style="{StaticResource menuButton}" x:Name="orderButton"
Content="Narudžba" Grid.Column="0" Grid.Row="4"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="orderButton_Click" />
        <Button Style="{StaticResource menuButton}" x:Name="contactButton"
Content="Kontakti" Grid.Column="0" Grid.Row="5"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="contactButton_Click" />
    </Grid>
```

Može se vidjeti DataGrid, kao i gumbovi za dodavanje i brisanje proizvoda. Border služi za odvajanje gumbova za navigaciju u jednu cjelinu te se time poboljšava dizajn aplikacije. S tim kodom početni prozor za upravljanje proizvodima izgleda ovako:



Slika 12. Prozor za proizvode (samostalna izrada 2023.)

Svim gumbovima su dodani događaji za klik. Također postoje stilovi zvani SuccessButton i DangerButton. Ti stilovi izgledaju ovako:

```
<Style x:Key="SuccessButton" TargetType="{x:Type Button}">
            <Setter Property="Background" Value="#28A745"/>
            <Setter Property="Foreground" Value="White"/>
            <Setter Property="BorderBrush" Value="#28A745"/>
            <Setter Property="BorderThickness" Value="1"/>
            <Setter Property="Padding" Value="20"/>
            <Setter Property="FontSize" Value="14"/>
            <Setter Property="FontWeight" Value="SemiBold"/>
            <Setter Property="Cursor" Value="Hand"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="{x:Type Button}">
                        <Border BorderBrush="Black" BorderThickness="1"
Background="{TemplateBinding Background}" CornerRadius="7">
                            <ContentPresenter HorizontalAlignment="Center"
VerticalAlignment="Center" />
                        </Border>
                        <ControlTemplate.Triggers>
                            <Trigger Property="IsMouseOver" Value="True">
                                <Setter Property="Background"
Value="#218838"/>
                            </Trigger>
                            <Trigger Property="IsPressed" Value="True">
```

```
                                <Setter Property="Background"
Value="#1E7E34"/>
                            </Trigger>
                            <Trigger Property="IsEnabled" Value="False">
                                <Setter Property="Background"
Value="#D0D0D0"/>
                                <Setter Property="Foreground"
Value="#6E6E6E"/>
                            </Trigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
```

SuccessButton stil mijenja boju gumba u zelenu. Daje mu također zeleni border, te mu se mijenja boja kad se prelazi mišem preko njega i kad ga se klikne. To se može videti u property-u „IsMouseOver“ ili „IsPressed“. Također mu se daje težina i veličina fonta. Stil za DangerButton je više manje identičan ovome samo se koristi crvena boja.

Nakon što se dodao prozor za prikaz proizvoda, mora se dodati prozor i za dodavanje proizvoda. Njegov XAML kod izgleda ovako:

```
<Grid.ColumnDefinitions>
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="60" />
            <ColumnDefinition Width="20" />
            <ColumnDefinition Width="150" />
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="20" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="20" />
        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>
            <RowDefinition Height="20"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="20"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="20"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="20"/>
```

```
            <RowDefinition Height="30"/>
            <RowDefinition Height="20"/>
        </Grid.RowDefinitions>

        <TextBlock    Grid.Column="1"    Grid.ColumnSpan="4"    Grid.Row="1"
Text="Ime proizvoda:" FontSize="20"/>
        <TextBox x:Name="nameText" Grid.Column="6" Grid.Row="1"/>
        <TextBlock    Grid.Column="1"    Grid.ColumnSpan="4"    Grid.Row="3"
Text="Cijena proizvoda po jedinici:" FontSize="20"/>
        <TextBox x:Name="priceText" Grid.Column="6" Grid.Row="3"/>
        <TextBlock    Grid.Column="1"    Grid.ColumnSpan="4"    Grid.Row="5"
Text="Količina:" FontSize="20"/>
        <TextBox x:Name="amountText" Grid.Column="6" Grid.Row="5"/>
        <Button  Style="{StaticResource  SuccessButton}"  x:Name="addButton"
Content="Dodaj"       Grid.Column="1"       Grid.Row="7"       FontSize="16"
Click="addButton_Click"/>
        <Button Style="{StaticResource DangerButton}" x:Name="returnButton"
Content="Odustani"      Grid.Column="3"      Grid.Row="7"      FontSize="16"
Margin="0,0,60,0" Click="returnButton_Click"/>
    </Grid>
```

Ovdje se nalaze TextBox-ovi i TextBlock-ovi koji služe za čitanje inputa korisnika. Također su dodani događaji na gumbove. Događaj za gumb „addButton“ ima kod koji izgleda ovako:

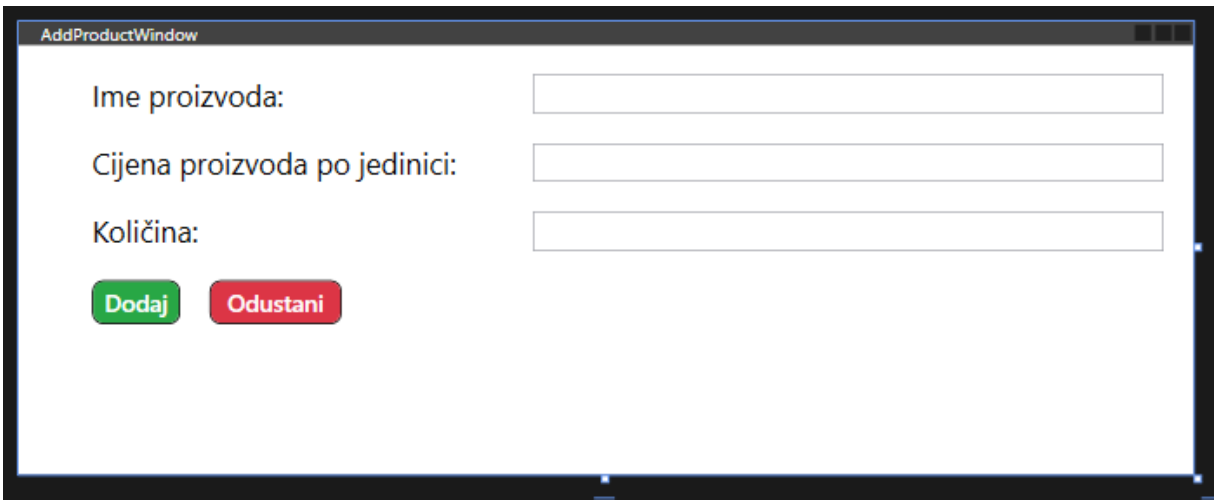
```
      private async void addButton_Click(object sender, RoutedEventArgs e)
        {
            MongoRepository mongoRepository = new MongoRepository();
            if (nameText.Text != "")
            {
                mongoRepository.AddProduct(nameText.Text,
int.Parse(amountText.Text), float.Parse(priceText.Text));
                InventoryWindow inventoryWindow = new InventoryWindow();
                inventoryWindow.Show();
                this.Close();
            }
            else
            {
                MessageBox.Show("Proizvod mora imati ime");
            }
        }
```

Kreira se instanca klase MongoRepository. Poziva se metoda te klase AddProduct i njoj se prosljeđuju podaci iz forme te postoji provjera da proizvod sadrži nekakvo ime. Nakon toga se

zatvara prozor za dodavanje i prikazuje prethodni prozor koji prikazuje proizvode. Kod za dodavanje proizvoda u repozitoriju izgleda ovako:

```
    public async void AddProduct(string name, int amount, float price)
        {
            string collectionName = "products";
            var                          collection                          =
database.GetCollection<Product>(collectionName);

            var product = new Product {Name = name, PricePerUnit = price,
Amount = amount };
            await collection.InsertOneAsync(product);

        }
```

Kod je sličan kodu dodavanja korisnika, samo se radi nad kolekcijom „products“. Prozor za dodavanje izgleda ovako:



Slika 13. Prozor za dodavanje proizvoda (samostalna izrada 2023.)

Na prozoru se nalaze 3 osnovna polja koja korisnik može ispuniti prije dodavanja novog proizvoda.

Osim dodavanja, potrebno je moći i izbrisati već postojeći proizvod. Kod za to izgleda ovako

```
private async void deleteButton_Click(object sender, RoutedEventArgs e)
        {
            var selected = dataGrid.SelectedItem as Product;
            MongoRepository mongoRepository = new MongoRepository();
            if (selected != null)
            {
                mongoRepository.DeleteProduct(selected);
```

```
                RefreshData();
            }
        }
```

Kao što se može vidjeti, kreira se varijabla koja predstavlja odabrani redak u tablici. Provjerava se sadrži li ta varijabla nekakvu vrijednost, ako sadrži poziva se metoda brisanja proizvoda i metoda osvježavanja podataka. Metoda RefreshData samo prikuplja proizvode iz baze i sprema ih u datagrid. Metoda brisanja u repozitoriju izgleda ovako:

```
public async void DeleteProduct(Product p)
      {
          var id = p.Id;
          var collection = database.GetCollection<Product>("products");
          var filter = Builders<Product>.Filter.Eq(p => p.Id, id);
          collection.DeleteOne(filter);
      }
```

Brisanje se radi pomoću ID-ja. Dohvate se svi proizvodi te se kreira filter pomoću kojeg se pronađe taj proizvod iz kolekcije preko zajedničkog identiteta. Nakon što se pronađe, poziva se metoda DeleteOne.

Naknadno, ostaje još samo mijenjanje podataka već postojećeg proizvoda kao jedna od osnovne 4 radnje CRUD-a. Prozor za uređivanje proizvoda ima više manje isti XAML kod kao i prozor za dodavanje, samo što će se kod otvaranja ovog polja popuniti već postojećim podacima. Stoga OnLoad metoda tog prozora izgleda ovako:

```
private static Product product;
      public UpdateProductWindow(Product p)
      {
          InitializeComponent();
          product = p;
      }

      private void Window_Loaded(object sender, RoutedEventArgs e)
      {
          nameText.Text = product.Name;
          priceText.Text = product.PricePerUnit.ToString();
          amountText.Text = product.Amount.ToString();
      }
```

Prozor prima u konstruktor odabrani proizvod i pomoću njegovih podataka popunjava text vrijednosti ovih textbox-ova.

Kod dugmeta za editiranje podataka izgleda ovako:

```
   private void editButton_Click(object sender, RoutedEventArgs e)
        {
            MongoRepository mongoRepository = new MongoRepository();
            Product newProduct = new Product { Amount = int.Parse(amountText.Text), Name = nameText.Text, PricePerUnit = float.Parse(priceText.Text) }
            if (nameText.Text != "")
            {
                mongoRepository.UpdateProduct(newProduct, product);
                InventoryWindow inventoryWindow = new InventoryWindow();
                inventoryWindow.Show();
                this.Close();
            }
            else
            {
                MessageBox.Show("Proizvod mora imati ime!");
            }
        }
```

Jednostavno se kreira novi product i prosljedi ga se u repozitorij ako proizvod sadrži nekakvo ime uneseno inače se pokaže greška. Metoda u repozitoriju je sljedeća:

```
public async void UpdateProduct(Product newProduct, Product oldProduct)
        {
            string collectionName = "products";
            var collection = database.GetCollection<Product>(collectionName);

            var updateDefinition = Builders<Product>.Update
                .Set(p => p.Name, newProduct.Name)
                .Set(p => p.PricePerUnit, newProduct.PricePerUnit)
                .Set(p => p.Amount, newProduct.Amount);

            var updateResult = await collection.UpdateOneAsync(
                    p => p.Id == oldProduct.Id,
                    updateDefinition);

        }
```

Metodi se moraju proslijediti stari i novi proizvod. Koristi se metoda update gdje se samo mijenjaju atributi proizvoda, taj specifični proizvod se pronađe pomoću metode

UpdateOneAsync kojoj se prosljedi id. S time su završene funkcionalnosti za proizvode. Prozor izgleda isti kao i prozor za dodavanje proizvoda.

4.4 Rad sa kontaktima

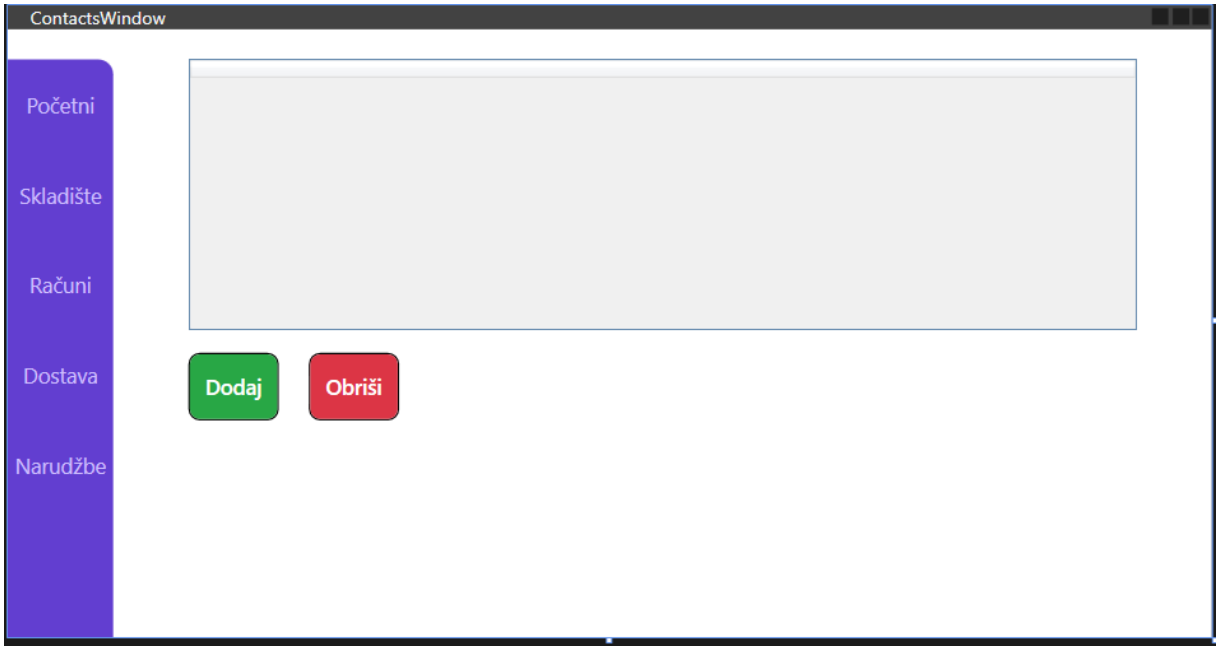
Slijede funkcionalnosti za upravljanje kontaktima. Kontakt klasa sadrži sljedeća svojstva:

```
public class Contact
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]

        public string Id { get; set; }

        public string Name { get; set; }

        public string Address { get; set; }

        public string PhoneNumber { get; set; }

        public string EmailAddress { get; set; }

    }
```

One su iste kao i funkcionalnosti za upravljanje proizvodima. Postoji prozor za pregled kontakta, editiranje i dodavanje novog. Ovako izgleda forma za dodavanje novog kontakta:

Slika 14. Prozor za dodavanje kontakta (samostalna izrada 2023.)

Kontakt klasa sadrži atribute koji se vide na formi: ime, adresa, broj mobitela i email adresa. Ako se klikne na gumb odustani, vraća se na prethodni prozor gdje se prikazuju kontakti. Forma za mijenjanje kontakta je ista kao i ova. Ovako izgleda početna forma za prikaz svih kontakta:



Slika 15. Prozor za kontakte (samostalna izrada 2023.)

Kao što se može vidjeti, forma je slična već navedenoj formi za proizvode.

Metode u repozitoriju koje služe za upravljanje kontaktima su dosta slične metodama za upravljanje proizvodima. Ovako izgleda metoda za dodavanje novog kontakta:

```
public async void AddContact(string name, string address, string phoneNumber, string emailAddress)
        {
            var collection = database.GetCollection<Contact>("contacts");
            var contact = new Contact { Address = address, EmailAddress = emailAddress, Name = name, PhoneNumber = phoneNumber };

            await collection.InsertOneAsync(contact);
        }
```

Kao što se može vidjeti, opet se dohvaća kolekcija, koja u ovom slučaju ima i svoj prikladni naziv te se u nju sa InsertOneAsync dodaje novi kontakt. Kod za brisanje i editiranje je također jako sličan već prethodno navedenome, samo je potrebno izmjeniti odgovarajuće nazive.

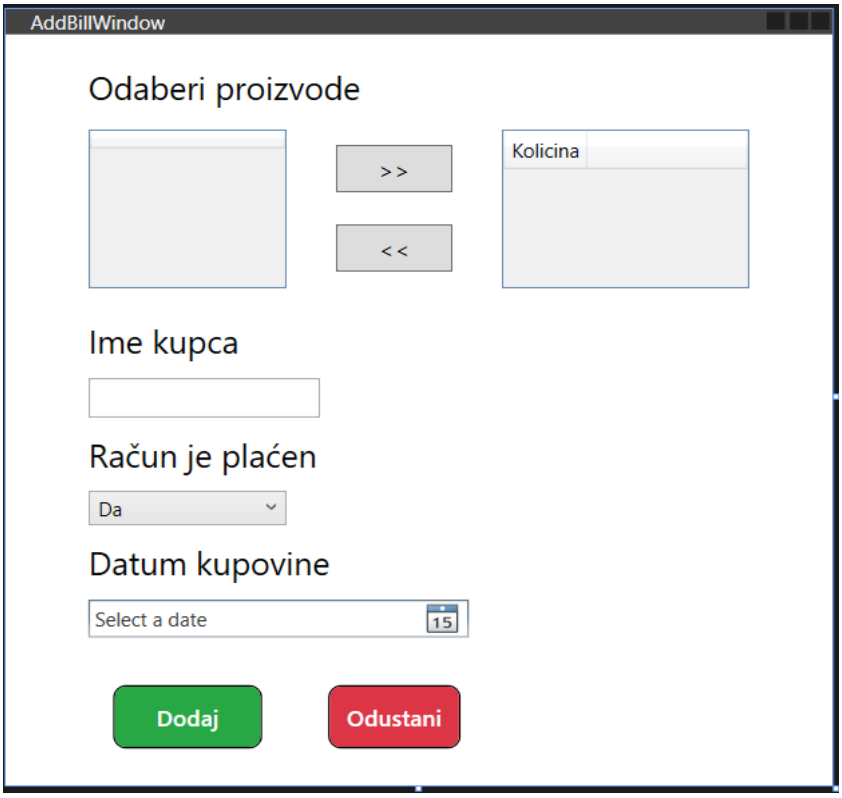
4.5 Rad sa računima

Osim toga, postoji funkcionalnost upravljanja računima koji služi kao sažet prikaz svih financijskih transakcija u poduzeću. Klasa račun sadrži sljedeća svojstva:

```
public class Bill
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }

        public List<Items> Items { get; set; }

        public string Buyer { get; set; }

        public bool IsPaid { get; set; }

        public DateTime DateOfPurchase { get; set; }

        public float Sum { get; set; }
    }
```

Može se vidjeti da klasa sadrži novu klasu zvanu „Items“. Ta klasa samo sadrži prozvod i njegovu količinu. Osim toga, klasa račun sadrži i ime kupca, status da li je plaćen, datum kupnje i ukupan trošak na proizvode.

Ovako izgleda forma za dodavanje novog računa:



Slika 16. Prozor za dodavanje računa (samostalna izrada 2023.)

Prozor je zamišljen tako da se s lijeve strane nalaze proizvodi koje se može prebaciti u desnu stranu, odnosno u desni datagrid. Svi proizvodi s desne strane se nalaze na računu. Osim toga, za račun se unosi i ime kupca, da li je plaćen te datum kupovine. XAML kod ovog prozora izgleda na sljedeći način:

```
<Grid>
      <Grid.ColumnDefinitions>
          <ColumnDefinition Width="50" />
          <ColumnDefinition Width="120" />
          <ColumnDefinition Width="20" />
          <ColumnDefinition Width="*" />
          <ColumnDefinition Width="20" />
          <ColumnDefinition Width="150" />
          <ColumnDefinition Width="50" />
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
          <RowDefinition Height="20"/>
          <RowDefinition Height="40"/>
          <RowDefinition Height="100"/>
```

```
            <RowDefinition Height="20"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="20"/>
            <RowDefinition Height="50"/>
            <RowDefinition Height="50"/>
        </Grid.RowDefinitions>
        <DataGrid   Name="dataGrid"   Grid.Column="1"   Grid.Row="2"
MinHeight="100"                              AutoGenerateColumns="True"
MouseDoubleClick="dataGrid_MouseDoubleClick">
        </DataGrid>


        <TextBlock  Grid.Column="1"  Grid.ColumnSpan="3"  Grid.Row="1"
Text="Odaberi proizvode" FontSize="20"/>
        <DataGrid   Name="dataGrid2"   Grid.Column="5"   Grid.Row="2"
MinHeight="100"                              AutoGenerateColumns="True"
MouseDoubleClick="dataGrid2_MouseDoubleClick">
            <DataGrid.Columns>
                <DataGridTextColumn                 Header="Kolicina"
Binding="{Binding Amount}" />
            </DataGrid.Columns>
        </DataGrid>
        <TextBlock  Grid.Column="1"  Grid.Row="4"  Text="Ime  kupca"
FontSize="20"/>
        <TextBox  x:Name="buyerText"  Grid.Column="1"  Grid.Row="5"
Grid.ColumnSpan="2" Height="25" Margin="0, 10"/>
        <TextBlock   Text="Račun   je   plaćen"   Grid.ColumnSpan="2"
Grid.Column="1" Grid.Row="6" FontSize="20" />
        <ComboBox       x:Name="billIsPaidText"       Grid.Column="1"
Grid.Row="7" SelectedIndex="0" Margin="0, 10">
            <ComboBoxItem Content="Da" />
            <ComboBoxItem Content="Ne" />
        </ComboBox>
        <TextBlock Text="Datum kupovine" Grid.Column="1" Grid.Row="8"
Grid.ColumnSpan="3" FontSize="20" />
        <DatePicker    x:Name="dateOfPurchaseText"    Grid.Column="1"
Grid.Row="9" Grid.ColumnSpan="3" Margin="0, 10"/>
```

```
        <Button  x:Name="toRightButton" Grid.Column="3" Grid.Row="2"
Content=">>"           FontSize="14"           Margin="10,10,10,60"
Click="toRightButton_Click"/>
        <Button   x:Name="toLeftButton" Grid.Column="3" Grid.Row="2"
Content="&lt;&lt;"         FontSize="14"         Margin="10,60,10,10"
Click="toLeftButton_Click" />


        <Button        Style="{StaticResource        SuccessButton}"
x:Name="addButton"  Content="Dodaj"  Grid.Column="1"  Grid.Row="11"
Click="addButton_Click" Margin="15,0,15,10" />
        <Button         Style="{StaticResource         DangerButton}"
x:Name="backButton" Content="Odustani" Grid.Column="3" Grid.Row="11"
Margin="5,0,5,10" Click="backButton_Click" />
    </Grid>
```

Može se primjetiti da je kod dosta sličan svim prethodnim XAML kodovima, samo se mijenja broj stupaca i redak ovisno o tome gdje korisnik želi pozicionirati određeni element na formi. Početna forma računa njihov prikaz je identična prethodnim formama za prikaz, dok forma za mijenjanje podataka već postojećeg računa je identična formi za dodavanje računa koja je maloprije opisana. Kod za dodavanje računa je drukčiji od postojećih budući da se kreira novi račun unutar koda samog prozora, koji se prosljeđuje repozitoriju, umjesto da mu se samo prosljede podaci i on se kreira tamo. To izgleda ovako:

```
    private async void addButton_Click(object sender, RoutedEventArgs e)
        {
            MongoRepository mongoRepository = new MongoRepository();
            List<Product> oldProducts = await mongoRepository.GetProducts();
            List<Items> items = new List<Items>();
            foreach (var product in dataGrid2.Items)
            {
                if (product is Product helpProduct)
                {
                    foreach (var item in oldProducts)
                    {
                        if (item.Id == helpProduct.Id)
                        {
                            items.Add(new    Items    {    AmountBought    =
helpProduct.Amount,   Product   =   item,   Sum   =   helpProduct.Amount   *
item.PricePerUnit });
                        }
                    }
                }
```

```
            }
            float total = 0;
            Bill bill = new Bill();
            foreach (var item in items)
            {
                total += item.Sum;
            }
            bill.Sum = total;
            bill.Items = items;
            bill.Buyer = buyerText.Text;

            ComboBoxItem selectedComboBoxItem =
(ComboBoxItem)billIsPaidText.Items[billIsPaidText.SelectedIndex];
            string selectedContent =
selectedComboBoxItem.Content.ToString();

            if (selectedContent == "Da")
            {
                bill.IsPaid = true;
            }
            else
            {
                bill.IsPaid = false;
            }

            DateTime? selectedDate = dateOfPurchaseText.SelectedDate;
            if (selectedDate.HasValue)
            {
                bill.DateOfPurchase = selectedDate.Value.Date;
            }
             if (bill.Items.Count < 1)
            {
                MessageBox.Show("Račun ne može biti bez proizvoda!");
            }
            else
            {
                mongoRepository.AddBill(bill);
                BillsWindow billWindow = new BillsWindow();
                billWindow.Show();
                this.Close();
```

```
            }
        }
```

U foreach petlji se provjerava id odabranih proizvoda, odnosno onih koji se nalaze u desnom datagridu. Ako ima isti id, taj proizvod se dodaje u svojstvo items. Isto tako se ostalim svojstvima dodajeljuje vrijednost. Na isti način se odvija i mijenjanje podataka već postojećeg računa, za kojeg postoji poseban prozor, kao što je već prije objašnjeno.

4.6 Rad sa dostavama

Svako skladište mora imati mogućnost pratiti narudžbe koje ono zaprimi. U ovom slučaju, zamišljeno je da postoje Order i Delivery klase. Delivery predstavlja sve proizvode koje dolaze u skladište, a Order predstavlja proizvode koje izlaze iz skladišta, odnosno one koji se prodaju.

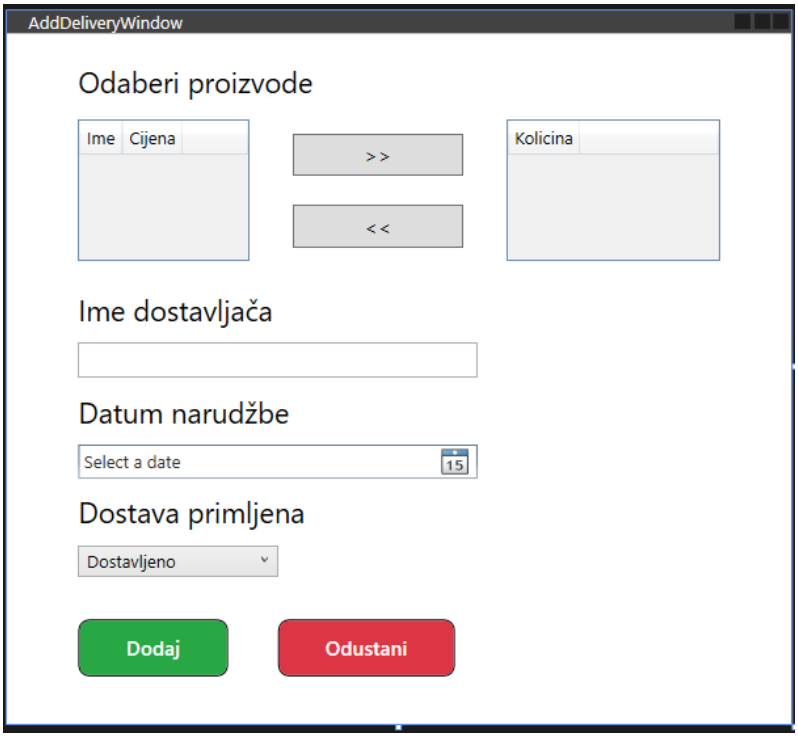
Klasa Delivery izgleda ovako:

```
public class Delivery
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }

        public string Deliverer { get; set; }

        public DateTime? DeliveryDate { get; set; }

        public List<Items> DeliveredItems { get; set; }

        public bool OrderIsDelivered { get; set; }

    }
```

Može se vidjeti da klasa sadrži ime dostavljača, datum dostave, status je li dostavljeno te popis svih dostavljenih proizvoda.

Ovako izgleda dizajn prozora za pregled dostupnih dostava i za dodavanje nove dostave:

Slika 17. Prozor za dostave (samostalna izrada 2023.)



Slika 18. Prozor za dodavanje dostava (samostalna izrada 2023.)

Kao što se može vidjeti, dizajn je sličan prozoru za dodavanje i pregled novog računa, dok je prozor za editiranje dostave jednak prozoru za dodavanje. Ovako izgleda kod za dodavanje nove dostave:

```
private async void addButton_Click(object sender, RoutedEventArgs e)
        {
            MongoRepository mongoRepository = new MongoRepository();
            List<Product> oldProducts = await mongoRepository.GetProducts();
            List<Items> items = new List<Items>();
            foreach (var product in dataGrid2.Items)
            {
                if (product is Product helpProduct)
                {
                    foreach (var item in oldProducts)
                    {
                        if (item.Id == helpProduct.Id)
                        {
                            items.Add(new    Items    {    AmountBought    =
helpProduct.Amount,   Product   =   item,   Sum   =   helpProduct.Amount   *
item.PricePerUnit });
                            break;
                        }
                    }
                }
            }
            Delivery delivery = new Delivery();
            delivery.DeliveredItems = items;
            delivery.Deliverer = delivererText.Text;
            DateTime? selectedDate = deliveryDateText.SelectedDate;
            if (selectedDate.HasValue)
            {
                delivery.DeliveryDate = selectedDate.Value.Date;
            }
            ComboBoxItem                selectedComboBoxItem                =
(ComboBoxItem)deliveredText.Items[deliveredText.SelectedIndex];
            string                      selectedContent                     =
selectedComboBoxItem.Content.ToString();


            if (selectedContent == "Dostavljeno")
            {
                delivery.OrderIsDelivered = true;
            }
            else
            {
                delivery.OrderIsDelivered = false;
            }
```

```
            if (delivery.DeliveredItems.Count < 1)
            {
                MessageBox.Show("Mora biti unesen barem 1 proizvod u
dostavu!");
            }
            else
            {
                mongoRepository.AddDelivery(delivery);
                DeliveryWindow deliveryWindow = new DeliveryWindow();
                deliveryWindow.Show();
                this.Close();
            }
        }
```

Proces dodavanja dostave je kao i kod računa, u kodu od prozora se kreira novi objekt te se na njega mapiraju vrijednosti, zatim se novokreirani objekt prosljeđuje klasi u repozitorij koja dodaje novi objekt u bazu. Kod editiranja se samo tekstualna polja popune sa vrijednostima prije nego što se prozor učita.

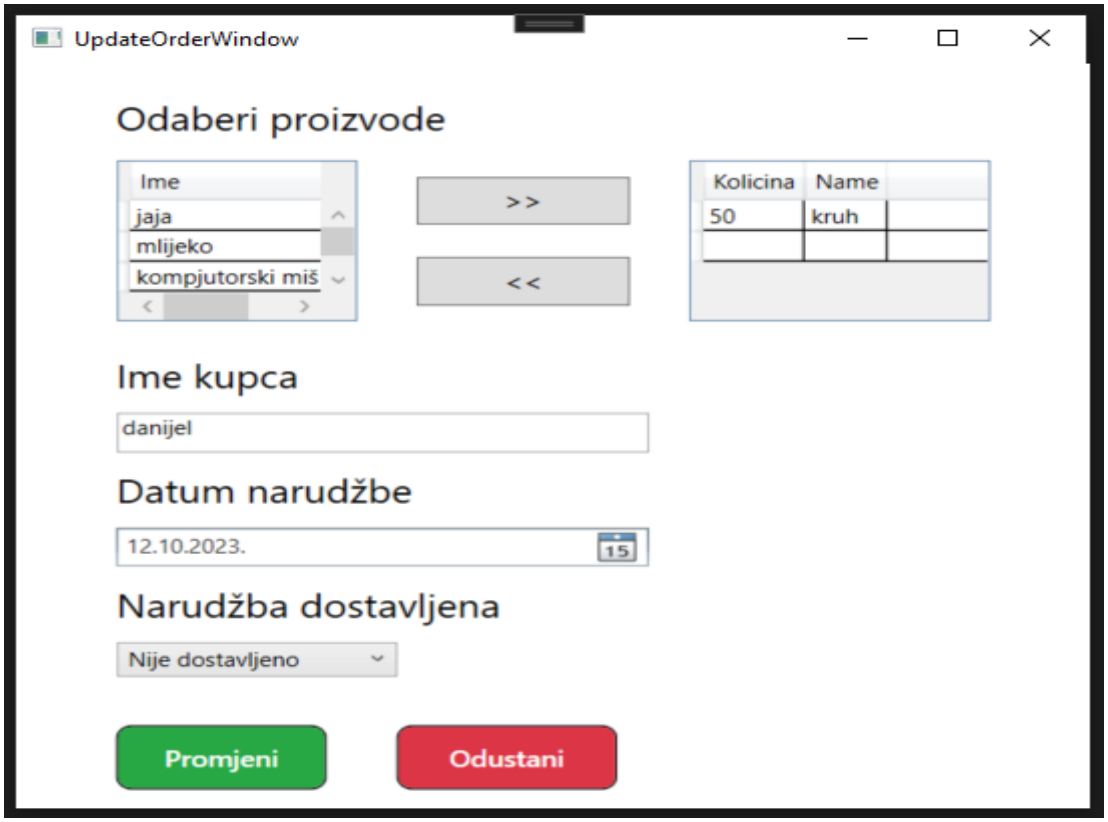
4.7 Rad sa narudžbama

Isto vrijedi i za klasu Order, odnosno narudžba, koja izgleda ovako:

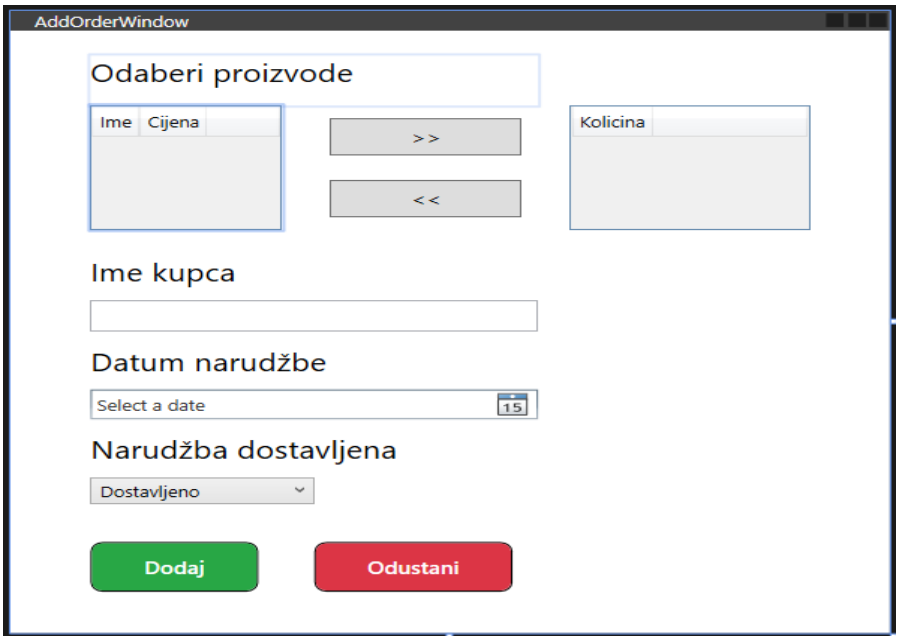
```
    public class Order
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }

        public string Customer { get; set; }

        public DateTime? DeliveryDate { get; set; }

        public List<Items> OrderedItems { get; set; }

        public bool OrderIsDelivered { get; set; }
    }
```

Svojstva klase su kao i kod dostave, samo se ovdje zovu OrderedItems kako bi se lakše raspoznala razlika između te dvije klase. Također su kodovi za dodavanje, editiranje, kao i XAML kodovi slični već prethodno opisanima.

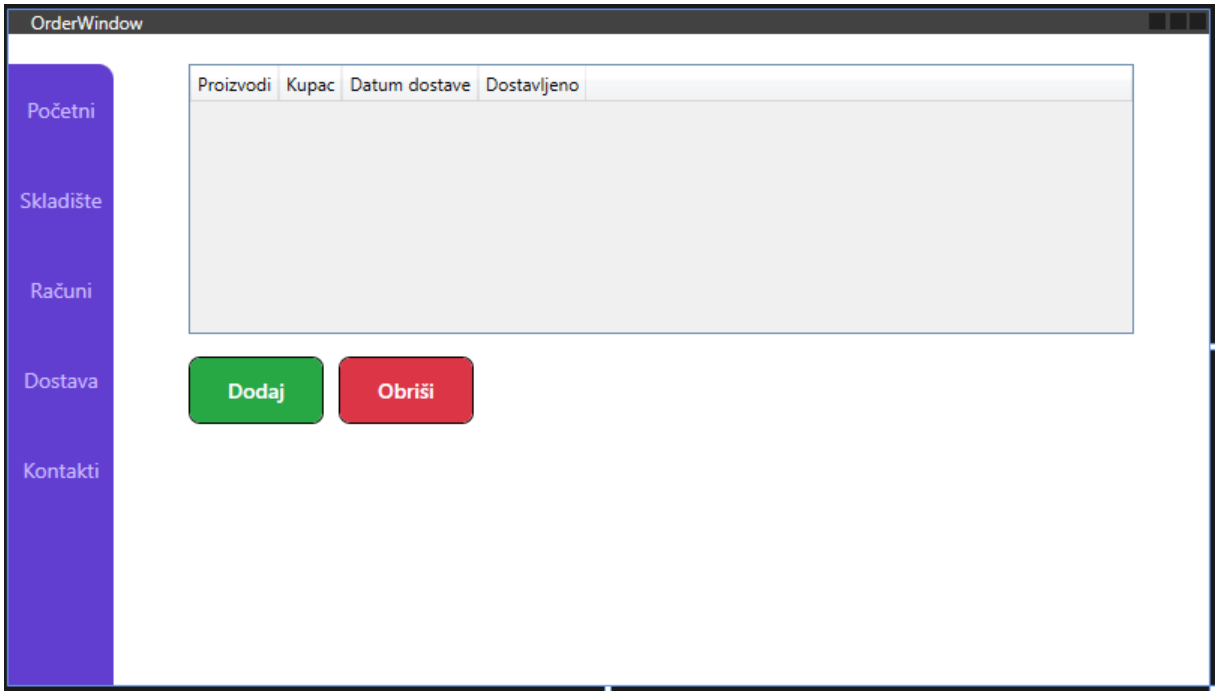
Ovako izgledaju ti prozori:



Slika 19. Prozor za izmjenu narudžbi (samostalna izrada 2023.)



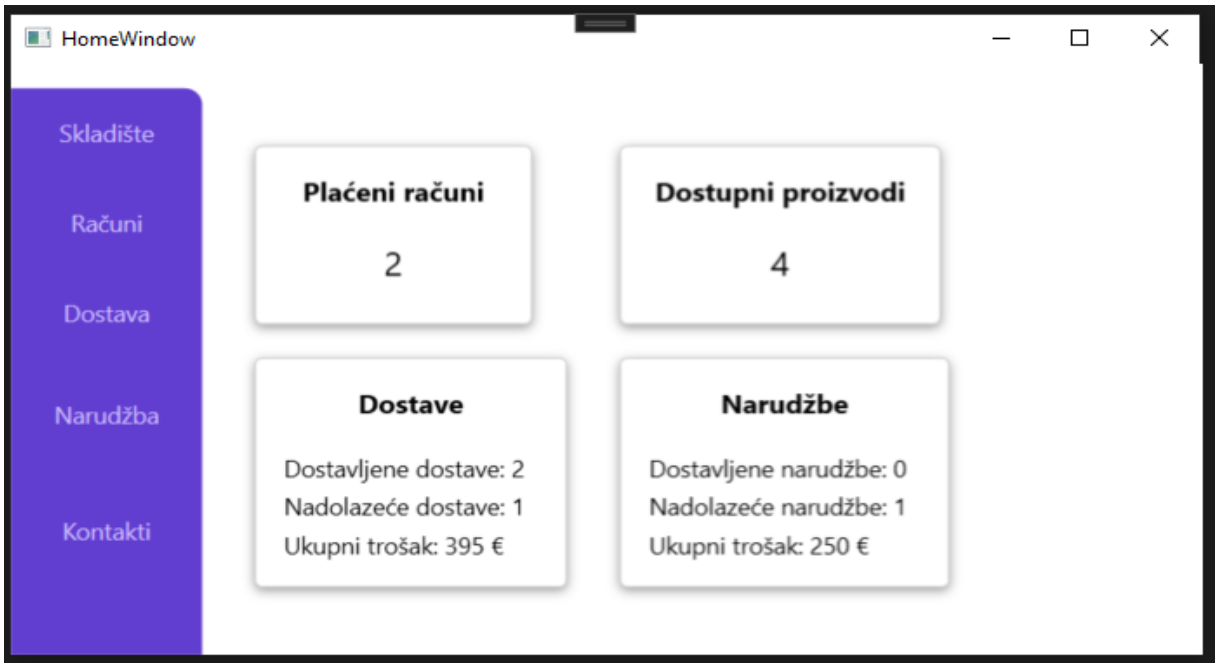
Slika 20. Prozor za dodavanje narudžbi (samostalna izrada 2023.)



Slika 21. Prozor za narudžbe (samostalna izrada 2023.)

4.8 Uređivanje početnog ekrana

Nakon toga preostaje još samo urediti početni ekran. Zamišljeno je da početni ekran sadrži sažetak općenitih informacija o stanju na skladištu, kao što je broj narudžbi koje dolaze ili koje su tek naručene ili općenito broj različitih proizvoda u skladištu. Stoga, početni ekran se može dizajnirati da izgleda ovako:



Slika 22. Uređeni početni prozor (samostalna izrada 2023.)

Može se vidjeti da se sažeto prikazuje broj plaćenih računa, dostupni proizvodi te osnovne informacije o dostava i narudžbama. Sve informacije se mogu detaljnije pogledati pomoću navigacije s lijeve strane koje su već opisane. Kako bi se postigao ovaj izgled početnog ekrana, koristi se sljedeći XAML kod:

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="auto" />
        <ColumnDefinition Width="30" />
        <ColumnDefinition Width="160" />
        <ColumnDefinition Width="50" />
        <ColumnDefinition Width="160" />
        <ColumnDefinition Width="*" />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="15"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="auto"/>
        <RowDefinition Height="*"/>
```

```
        </Grid.RowDefinitions>


        <Border       Grid.Row="1"       Width="110"       Grid.RowSpan="7"
Background="#623ed0" CornerRadius="0 10 0 0">

        </Border>

        <Button  Style="{StaticResource  menuButton}"  x:Name="homeButton"
Content="Skladište" Grid.Column="0" Grid.Row="1"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="homeButton_Click"/>

        <Button  Style="{StaticResource  menuButton}"  x:Name="billsButton"
Content="Računi" Grid.Column="0" Grid.Row="2"  MaxWidth="100" Margin="0, 10"
FontSize="14" Click="billsButton_Click"/>

        <Button Style="{StaticResource menuButton}" x:Name="deliveryButton"
Content="Dostava" Grid.Column="0" Grid.Row="3"  MaxWidth="100" Margin="0, 10"
FontSize="14" Click="deliveryButton_Click"/>

        <Button  Style="{StaticResource  menuButton}"  x:Name="orderButton"
Content="Narudžba" Grid.Column="0" Grid.Row="4"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="orderButton_Click"/>

        <Button Style="{StaticResource menuButton}" x:Name="contactButton"
Content="Kontakti" Grid.Column="0" Grid.Row="5"  MaxWidth="100" Margin="0,
10" FontSize="14" Click="contactButton_Click"/>


        <Border      Grid.Column="2"      Grid.Row="2"      Grid.RowSpan="2"
Style="{StaticResource CardStyle}" Margin="0,-20,0,20">

            <StackPanel>

                <TextBlock    Text="Plaćeni    računi"    FontWeight="Bold"
FontSize="16" HorizontalAlignment="Center" VerticalAlignment="Center"/>

                <TextBlock    x:Name="paidBillsNumberText"    FontSize="20"
TextAlignment="Center"  Margin="0,20,0,0"/>

            </StackPanel>

        </Border>

        <Border      Grid.Column="4"      Grid.Row="2"      Grid.RowSpan="2"
Style="{StaticResource CardStyle}" Margin="0,-20,-25,20" >

            <StackPanel>

                <TextBlock   Text="Dostupni   proizvodi"   FontWeight="Bold"
FontSize="16" HorizontalAlignment="Center" VerticalAlignment="Center"/>

                <TextBlock     x:Name="productsNumberText"     FontSize="20"
TextAlignment="Center" Margin="0,20,0,0" />

            </StackPanel>

        </Border>

        <Border      Grid.Column="2"      Grid.Row="4"      Grid.RowSpan="2"
Style="{StaticResource CardStyle}" Margin="0,0,-20,0">

            <StackPanel>

                <TextBlock  Text="Dostave"  FontWeight="Bold"  FontSize="16"
HorizontalAlignment="Center" VerticalAlignment="Center"/>

                <TextBlock   FontSize="14"   x:Name="orderedDeliveriesText"
Margin="0,20,0,0"/>

                <TextBlock   FontSize="14"   x:Name="incomingDeliveriesText"
Margin="0,5,0,0"  />
```

```
                <TextBlock   FontSize="14"   x:Name="deliveriesCostSumText"
Margin="0,5,0,0"  />
            </StackPanel>
        </Border>
        <Border      Grid.Column="4"      Grid.Row="4"      Grid.RowSpan="2"
Style="{StaticResource CardStyle}" Margin="0,0,-30,0">
            <StackPanel>
                <TextBlock  Text="Narudžbe"  FontWeight="Bold"  FontSize="16"
HorizontalAlignment="Center" VerticalAlignment="Center"/>
                <TextBlock     FontSize="14"     x:Name="orderedOrdersText"
Margin="0,20,0,0"/>
                <TextBlock     FontSize="14"     x:Name="incomingOrdersText"
Margin="0,5,0,0"  />
                <TextBlock     FontSize="14"     x:Name="ordersCostSumText"
Margin="0,5,0,0"  />
            </StackPanel>
        </Border>
    </Grid>
```

Kako bi se postigao izgled pravokutnika, koristi se border unutar kojeg se stavlja stack panel koji sadrži obične textbox-ove. Stil CardStyle izgleda ovako:

```
<Style TargetType="Border" x:Key="CardStyle">
            <Setter Property="Background" Value="White"/>
            <Setter Property="BorderBrush" Value="LightGray"/>
            <Setter Property="BorderThickness" Value="1"/>
            <Setter Property="Padding" Value="16"/>
            <Setter Property="CornerRadius" Value="4"/>
            <Setter Property="Effect">
                <Setter.Value>
                    <DropShadowEffect      Color="Gray"      BlurRadius="10"
Opacity="0.5" Direction="270" ShadowDepth="2"/>
                </Setter.Value>
            </Setter>
        </Style>
```

U stilu se postavlja bijela pozadina, stavlja se efekt sjene te vrijednosti koje upravljaju izgledom bordera, odnosno granice.

Kako bi se učitali svi podaci za početni prozor i izračunale vrijednosti, koristi se sljedeći kod:

```
  private async void Window_Loaded(object sender, RoutedEventArgs e)
        {
```

```
            MongoRepository mongoRepository = new MongoRepository();
            List<Bill> bills = await mongoRepository.GetPaidBills();
            paidBillsNumberText.Text = bills.Count.ToString();
            List<Product> products = await
mongoRepository.GetAvailableProducts();
            productsNumberText.Text = products.Count.ToString();
            int numOfOrderedDeliveries = await
mongoRepository.GetNumberOfDeliveredDeliveries();
            int numOfIncomingDeliveries = await
mongoRepository.GetNumberOfIncomingDeliveries();
            float totalSum = await mongoRepository.GetSumOfDeliveriesCost();
            orderedDeliveriesText.Text = "Dostavljene dostave: " +
numOfOrderedDeliveries.ToString();
            incomingDeliveriesText.Text = "Nadolazeće dostave: " +
numOfIncomingDeliveries.ToString();
            deliveriesCostSumText.Text = "Ukupni trošak: " +
totalSum.ToString() + " €";


            int numOfOrderedOrders = await
mongoRepository.GetNumberOfDeliveredOrders();
            int numOfIncomingOrders = await
mongoRepository.GetNumberOfIncomingOrders();
            float totalSum2 = await mongoRepository.GetSumOfOrdersCost();
            orderedOrdersText.Text = "Dostavljene narudžbe: " +
numOfOrderedOrders.ToString();
            incomingOrdersText.Text = "Nadolazeće narudžbe: " +
numOfIncomingOrders.ToString();
            ordersCostSumText.Text = "Ukupni trošak: " + totalSum2.ToString()
+ " €";


        }
```

Iz repozitorija se dohvaća broj proizvoda te se dohvaća broj računa koji imaju vrijednost IsPaid stavljeno na true, dakle da su plaćeni. Ispod toga se može vidjeti dohvaćanje dostavi i narudžbi koje su već dostavljene i one koje tek trebaju biti dostaljene.

Ovo je primjer koda koji dohvaća narudžbe na temelju nekog atributa:

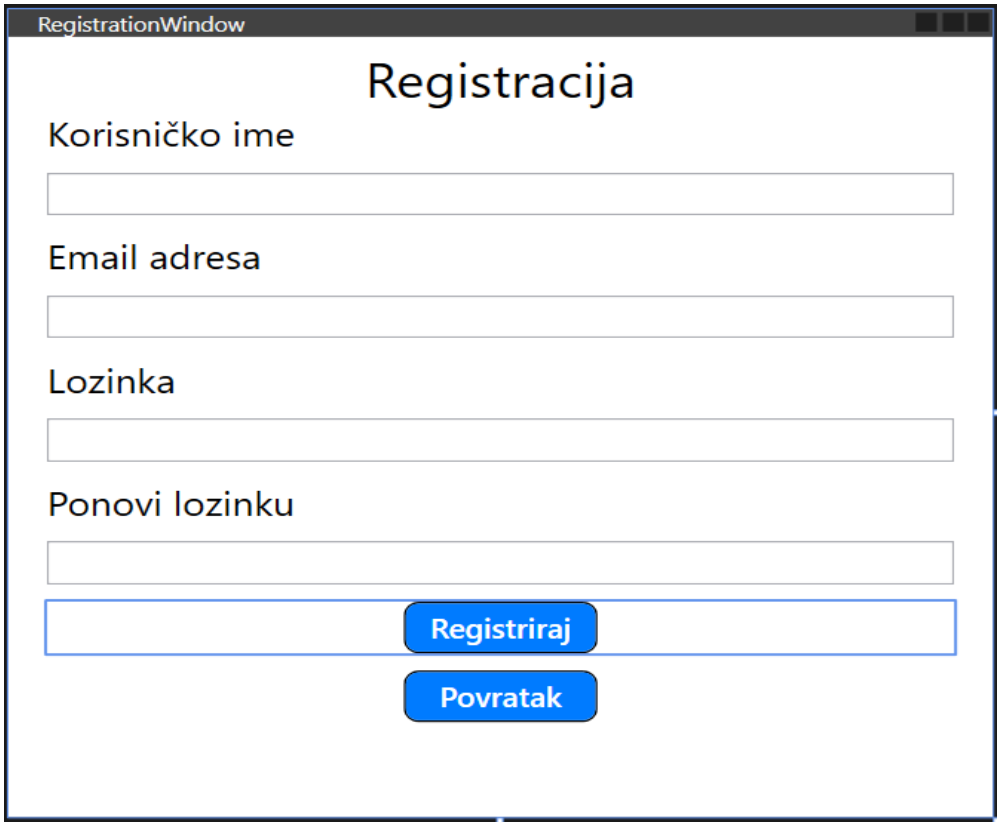
```
public async Task<int> GetNumberOfDeliveredOrders()
        {
            List<Order> orders = new List<Order>();
            var collection = database.GetCollection<Order>("orders");
            var filter = Builders<Order>.Filter.Eq(del =>
del.OrderIsDelivered, true);
            var results = await collection.FindAsync(filter);
```

```
        await results.ForEachAsync(bill => orders.Add(bill));
        return orders.Count;
    }
```

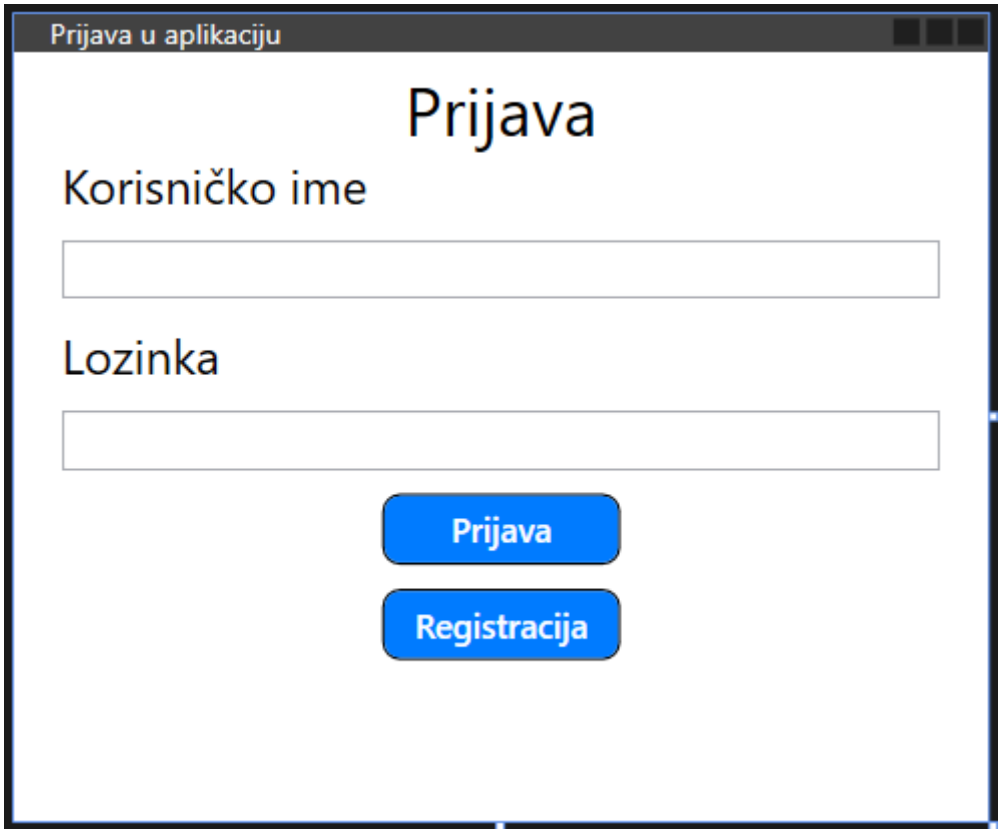
Samo je potrebno kreirati filter kojem se prosljedi atribut te se poziva funkcija FindAsync i njoj se prosljedi filter. Na isti način se dohvaćaju ostale narudžbe i dostave.

4.9 Ostali stilovi

Na samom kraju je kreiran stil za gumbove za početne ekrane registracije i prijave te stil koji će urediti običan datagrid koji poboljšavanje njegov izgled. Taj stil za gumbove je isti kao već opisani danger i success button stilovi samo je boja promjenjena u plavu. S time ekrani za prijavu i registraciju izgledaju ovako:



Slika 23. Uređeni prozor za registraciju (samostalna izrada 2023.)

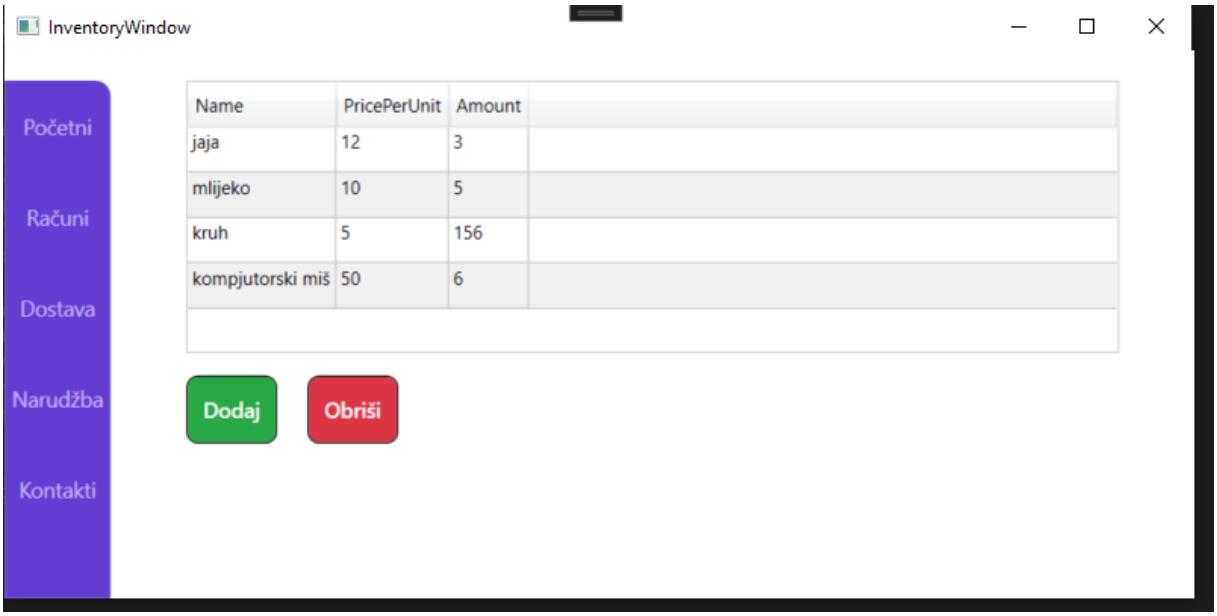


Slika 24. Uređeni prozor za prijavu (samostalna izrada 2023.)

Stil za datagrid izgleda na sljedeći način:

```
<Style x:Key="CustomDataGridStyle" TargetType="DataGrid">
            <Setter Property="AutoGenerateColumns" Value="True" />
            <Setter Property="AlternatingRowBackground" Value="#F0F0F0" />
            <Setter Property="Background" Value="White" />
            <Setter Property="RowHeaderWidth" Value="0" />
            <Setter Property="BorderBrush" Value="#CCCCCC" />
            <Setter Property="BorderThickness" Value="1" />
            <Setter Property="ColumnHeaderHeight" Value="30" />
            <Setter Property="RowHeight" Value="30" />
            <Setter Property="HorizontalGridLinesBrush" Value="#CCCCCC" />
            <Setter Property="VerticalGridLinesBrush" Value="#CCCCCC" />
            <Setter Property="HeadersVisibility" Value="Column" />
            <Setter Property="SelectionMode" Value="Single" />
            <Setter Property="CanUserAddRows" Value="False" />
        </Style>
```

Dakle, mijenjanju se samo osnovne stvari kao visina retka, pozadinska boja itd. te s time datagrid izgleda ovako:



Slika 25. Uređeni datagrid (samostalna izrada 2023.)

5. Zaključak

U ovom radu je opisan postupak izrade aplikacije za upravljanja skladištem u C# koristeći WPF tehnologiju. Postupak je detaljno razrađen, prikazan je XAML kod koji pokazuje kako dizajnirati prozore, kod za komuniciranje s bazom te kod za upravljenje tim podacima.

U procesu razvoja, prikazan je osnovan CRUD sustav koji se danas najčešće koristi u programerskom svijetu, to je kreiranje(create), čitanje(read), mijenjanje(update) i brisanje(delete). Pomoću te 4 temeljne paradigme je razvijena aplikacija koja može poslužiti u stvarnom svijetu poduzetništva. Osim toga, korištenje WPF-a kao tehnologije je omogućilo kreiranje atraktivnog i suvremenijeg grafičkog sučelja koje je intuitivno i jednostavno za korištenje.

Korištenje MongoDB za bazu podataka je uvelike olakšao postupak razvoja aplikacije te je pokazano da i početnici mogu lako krenuti programirati koristeći NoSQL baze koje su skalabilne.

Zaključno se može reći da je rad detaljno opisao razvoj ovakvog tipa aplikacije te može pomoći svim početnicima u programiranju kako bi i sama mogli započeti razvijati svoje aplikacije.

Poveznica na Github:

https://github.com/DanijelZebcevic/InventoryPro

Literatura:

[1] MongoDB (bez dat.) *What is NoSQL?* [Na internetu] Dostupno: https://www.mongodb.com/nosql-explained

[2] MongoDB (bez dat.) [Na internetu] Dostupno: https://www.mongodb.com/

[3] Alexander S. Gillis, „What is MongoDB?“ (bez dat.) [Na internetu] Dostupno: https://www.techtarget.com/searchdatamanagement/definition/MongoDB

[4] Mahesh Chand, "What WPF?", 13.6.2019.. [Na internetu]. Dostupno: https://www.c-sharpcorner.com/blogs/what-wpf-is1

[5] IAmTimCorrey, (06.12.2021) "How to Connect MongoDB to C# the Easy Way," Youtube [Video datoteka]. Dostupno: https://www.youtube.com/watch?v=exXavNOqaVo&t=919s&ab_channel=IAmTimCorey

[6] C# WPF UI Academy, (01.06.2022) " C# WPF UI | How to Design Flat Data Table Dashboard in WPF (Customize Datagrid)" Youtube [Video datoteka]. Dostupno:

https://www.youtube.com/watch?v=mlmyFXJy8gQ&ab_channel=C%23WPFUIAcademy

Popis slika