Sustav za upravljanje restoranom u C#

Gaši, Juraj

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Juraj Gaši

SUSTAV ZA UPRAVLJANJE RESTORANOM U C#

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Juraj Gaši

JMBAG: 0016151946

Studij: Informacijski i poslovni sustavi

SUSTAV ZA UPRAVLJANJE RESTORANOM U C#

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, kolovoz 2024.

Juraj Gaši

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

Ovaj završni rad bavi se razvojem desktop aplikacije za restorane, koja integrira funkcionalnosti upravljanja korisnicima, jelovnicima, stolovima i rezervacijama, narudžbama hrane i pića i zalihama te prikazivanje statistika. Aplikacija omogućuje različite razine pristupa i ovlasti za tri vrste korisnika: administratora, osoblje i običnog korisnika. Administratori imaju širok spektar mogućnosti, uključujući upravljanje korisnicima i resursima restorana, dok obični korisnici mogu vršiti rezervacije, narudžbe i pregledavati ponudu te unositi recenzije. Osoblje upravlja narudžbama i priprema ih za realizaciju.

Završni rad se temelji na principima troslojne arhitekture koja se sastoji od prezentacijskog sloja, sloja za pristup podacima, poslovne logike i entiteta. U sklopu rada korišten je sustav za upravljanje korisničkim sučeljem putem Window prozora, User Controla, GUI Managera i navigacijskog bara. Svaki tip korisnika ima vlastiti Window koji sadrži prilagođeni navigacijski bar, omogućujući pristup različitim dijelovima aplikacije. Putem ovog navigacijskog bara, korisnik se kreće kroz aplikaciju, pri čemu se User Control komponente dinamički mijenjaju unutar glavnog Windowa. Window konstantno ostaje vidljiv, dok se sadržaj mijenja ovisno o odabiru u navigacijskom baru ili odabirom tipke unutar User Controle. Upravljanje izmjenom User Controla unutar Windowa obavlja GUI manager, koji dinamički postavlja odgovarajuće kontrole na temelju korisničkih interakcija, omogućujući glatko i intuitivno kretanje kroz aplikaciju.

Integracija ovih funkcionalnosti u jedinstveni sustav može značajno poboljšati učinkovitost i organizaciju rada u restoranu. Zaključci potvrđuju da ovakav sustav pruža sveobuhvatan alat za upravljanje restoranom, optimizirajući radne procese i poboljšavajući korisničko iskustvo.

Ključne riječi: sustav za upravljanje restoranom; C#; WPF; SQL Server; Troslojna arhitektura; Microsoft Azure; Entity Framework; desktop aplikacija;

Sadržaj

Uvod

Tema ovog završnog rada je razvoj desktop aplikacije specijalizirane za restorane, koja omogućava cjelovito upravljanje različitim aspektima restoranskog poslovanja, uključujući, korisnike, rezervacije i stolove, narudžbe, jelovnike, recenzije, zalihe i statistike. S obzirom na sve veće zahtjeve tržišta i konkurencije u ugostiteljskom sektoru, učinkovita organizacija i optimizacija poslovnih procesa postala je ključna za održavanje i unapređenje kvalitete usluge. Razvoj ovakvih aplikacija pruža restoranima ne samo mogućnost bolje organizacije rada, već i značajno poboljšanje korisničkog iskustva kroz brže i preciznije upravljanje narudžbama, rezervacijama, jelovnicima, korisnicima, recenzijama i zalihama.

Značaj ove teme ogleda se u potrebi za modernizacijom poslovanja u ugostiteljstvu, koje se sve više oslanja na digitalna rješenja. Kroz automatizaciju i integraciju različitih funkcionalnosti, restorani mogu optimizirati svoje operacije, smanjiti rizik od ljudskih pogrešaka, te omogućiti bolju koordinaciju među osobljem. Osim toga, korisnici dobivaju brži i jednostavniji pristup uslugama, što može povećati njihovo zadovoljstvo i lojalnost.

Moja motivacija za odabir ove teme proizlazi iz osobnog interesa za kombinaciju tehnologije i poslovnih procesa, kao i iz želje za stvaranjem praktičnog i korisnog rješenja koje može unaprijediti rad restorana. Smatram da tehnologija ima potencijal značajno poboljšati svakodnevni rad u ugostiteljstvu, a ovaj projekt pruža priliku za primjenu stečenog znanja iz područja programiranja, dizajna softverskih sustava i upravljanje podacima. Osim tehničkih izazova, rad na ovoj temi omogućuje i razumijevanje stvarnih potreba i problema s kojima se suočavaju restorani, čime doprinosi razvoju relevantnih i primjenjivih rješenja u praksi.

Tehnologije

2.1. C#

C# je objektno-orijentirani programski jezik razvijen od strane Microsofta, koji se koristi u okviru .NET platforme. Osmišljen je da bude jednostavan, moderan i svestran, omogućujući programerima razvoj raznih vrsta aplikacija, od desktop i web aplikacija do mobilnih aplikacija i igara. C# podržava brojne programske paradigme, uključujući proceduralno, objektno-orijentirano i funkcionalno programiranje. Njegov snažan sustav pomaže u smanjenju grešaka i poboljšava sigurnost koda. Jezik također nudi bogat skup biblioteka i alata kroz .NET Framework i .NET Core, što omogućava efikasno upravljanje resursima i rad s različitim tehnologijama. Korištenje C#-a omogućuje visoku produktivnost i fleksibilnost, čineći ga popularnim izborom za razvoj složenih i skalabilnih aplikacija. [1]

2.2. Entity Framework

Entity Framework (EF) je objektno-orijentirani relacijski mapper (ORM) za .NET platformu koji omogućava programerima da rade s podacima u obliku objekata specifičnih za domenu bez potrebe za pisanjem složenih SQL upita. EF pojednostavljuje rad s bazama podataka kroz automatsko mapiranje objekata na tablice u bazi podataka i omogućava pristup podacima koristeći LINQ (Language Integrated Query). Podržava razne načine rada, uključujući Code First, Database First i Model First pristupe, čime pruža fleksibilnost u načinima modeliranja i pristupu podacima. Entity Framework olakšava rad s podacima i ubrzava razvoj aplikacija smanjujući potrebu za ručnim upravljanjem SQL kodom. [2]

2.3. WPF

Windows Presentation Foundation (WPF) je grafički API za razvoj bogatih korisničkih sučelja (UI) na Windows platformi. Dio je .NET framework-a i omogućuje razvoj desktop aplikacija s modernim i dinamičkim korisničkim sučeljima. WPF pruža mogućnosti za napredne vizualne efekte, animacije, 3D grafiku i detaljno stiliziranje kroz bogat skup kontrola i elemenata. Koristi se za izgradnju aplikacija koje mogu imati kompleksne interakcije i vizualizacije, s podrškom za različite vrste prikaza i integraciju s drugim .NET tehnologijama.

WPF omogućuje visoku fleksibilnost i prilagodbu u dizajnu korisničkog sučelja, čime doprinosi stvaranju profesionalnih i privlačnih desktop aplikacija. [3]

2.3.1. XAML

XAML (Extensible Application Markup Language) je deklarativni jezik za opisivanje korisničkog sučelja u aplikacijama temeljenim na WPF-u i drugim .NET tehnologijama. XAML omogućuje dizajnerima i programerima da definiraju vizualne elemente i njihov raspored u aplikaciji koristeći sintaksu sličnu kao XML. Omogućuje odvajanje dizajna od aplikacijske logike, što olakšava rad u timovima i ubrzava razvoj aplikacija. Korištenjem XAML-a, korisnici mogu jednostavno definirati stilove, predloške i resurse za aplikaciju, što doprinosi dosljednom i profesionalnom izgledu sučelja. [4]

2.4. SQL Server

SQL Server je sustav za upravljanje relacijskim bazama podataka (RDBMS) razvijen od strane Microsofta. Pruža robusno rješenje za pohranu, upravljanje i analizu podataka koristeći SQL (Structured Query Language). SQL Server nudi napredne značajke kao što su transakcije, pohranjene procedure, triggere i indekse, koje omogućuju visoku dostupnost, sigurnost i optimalne performanse. Uključuje alate za analizu podataka, poslovnu inteligenciju i izvještavanje kroz komponente kao što su SQL Server Analysis Services (SSAS), SQL Server Integration Services (SSIS) i SQL Server Reposting Services (SSRS). SQL Server je široko korišten za razvoj složenih aplikacija i sustava koji zahtijevaju pouzdanu i skalabilnu bazu podataka. [5]

2.5. Microsoft Azure

Microsoft Azure je sveobuhvatna cloud platforma koja omogućuje razvoj, implementaciju i upravljanje aplikacijama i uslugama kroz globalnu mrežu podatkovnih centara. Azure pruža širok raspon usluga uključujući pohranu podataka, analitiku, virtualne poslužitelje, mrežne usluge, baze podataka, umjetnu inteligenciju, Internet stvari (IoT) i brojne druge. Omogućuje korisnicima skalabilnost, fleksibilnost i plaćanje samo za stvarno korištene resurse. Azure također osigurava visoku dostupnost i sigurnost, te se lako integrira s drugim

Microsoftovim proizvodima, čime se omogućuje jednostavna integracija s postojećom IT infrastrukturom i sustavima. [6]

2.6. MySQL Workbench

MySQL Workbench je sveobuhvatan alat za upravljanje bazama podataka koji se koristi s MySQL sustavom za upravljanje relacijskim bazama podataka. Pruža grafičko korisničko sučelje za dizajniranje, modeliranje, upravljanje i administraciju MySQL baza podataka. MySQL Workbench uključuje alate za vizualno modeliranje podataka, kreiranje i upravljanje bazama podataka, izvođenje SQL upita, analizu performansi i rad s podacima. Također omogućuje jednostavno stvaranje ER dijagrama (entitet-relacija) i pruža funkcionalnosti za migraciju podataka, što olakšava rad s kompleksnim bazama podataka i poboljšava produktivnost razvoja. [7]

2.7. Visual Studio

Visual Studio je integrirano razvojno okruženje (IDE) koje je razvila kompanija Microsoft. Koristi se za razvoj softvera, uključujući aplikacija za web, mobilne uređaje, desktop aplikacije, igre i cloud rješenja. Visual Studio podržava više programskih jezika, uključujući C#, C++, Python, JavaScript i mnoge druge, te pruža napredne alate za uređivanje koda, otklanjanje pogrešaka, testiranje i implementaciju. IDE također omogućuje integraciju s različitim alatima za verzioniranje koda kao što je Git, te nudi funkcionalnosti za suradnju u timu, poput zajedničkog uređivanja koda u stvarnom vremenu. Visual Studio je dizajniran da povećava produktivnost programera i omogućuje im stvaranje složenih aplikacija na jednostavan i učinkovit način. [8]

2.8. GitHub

GitHub je popularna platforma za verzioniranje koda i suradnju koja omogućuje programerima da pohranjuju, dijele i upravljaju svojim projektima. GitHub koristi Git, distribuirani sustav za kontrolu verzija, koji omogućuje praćenje promjena u kodu i olakšava suradnju među timovima. Programeri mogu kreirati repozitorije, raditi na različitim granama projekta, te slati „pull requestove“ za pregled i integraciju promjena u glavnu verziju koda. GitHub također nudi funkcionalnosti za upravljanje projektima, uključujući bug tracking,

upravljanje zadacima, te alate za automatizaciju procesa implementacije (CI/CD). Platforma je ključna za open-source projekte, ali se široko koristi i u komercijalnom razvoju softvera. [9]

2.9. Microsoft Word

Microsoft Word je vodeći alat za obradu teksta, razvijen od strane Microsofta, koji omogućuje korisnicima da kreiraju, uređuju i oblikuju dokumente. Word pruža raznovrsne funkcionalnosti uključujući automatsko ispravljanje pravopisa i gramatike, umetanje slika, tablica, grafikona i drugih objekata, kao i napredne opcije oblikovanja teksta. Program nudi brojne predloške koji olakšavaju izradu profesionalnih dokumenata kao što su pisma, izvještaji, životopisi i poslovne ponude. Osim toga, Microsoft Word omogućuje suradnju u stvarnom vremenu putem integracije s OneDrive-om i SharePoint-om, omogućujući više korisnika da istovremeno rade na istom dokumentu. Word je široko korišten u poslovnom, obrazovnom i osobnom okruženju zbog svoje fleksibilnosti i jednostavnosti uporabe. [10]

Arhitektura sustava

3.1. Troslojna arhitektura sustava

Arhitektura koju sam koristio za svoj sustav je troslojna arhitektura sustava. Troslojna arhitektura sustava predstavlja metodologiju dizajna softverskih aplikacija koja se temelji na razdvajanju aplikacije u tri međusobno povezana, ali jasno odvojena sloja. Ova arhitektura osigurava modularnost i skalabilnost aplikacije, omogućujući jednostavnije održavanje i unapređenje sustava tijekom njegovog životnog ciklusa. Svaki sloj unutar arhitekture ima specifičnu ulogu, a komunikacija između slojeva se odvija na način koji minimizira međusobnu zavisnost. [11]

3.1.1. Struktura troslojne arhitekture

3.1.1.1. Odvajanje odgovornosti

Svaki sloj u troslojnoj arhitekturi ima jasno definirane odgovornosti i zadatke, čime se omogućuje specijalizacija i fokus unutar svakog sloja. Ova segmentacija pridonosi boljoj organizaciji koda i većoj fleksibilnosti u razvoju i održavanju aplikacije.

3.1.1.2. Smanjenje međusobne zavisnosti

Troslojna arhitektura smanjuje međusobnu zavisnost između komponenti sustava. Svaki sloj je neovisan o ostalim slojevima i komunicira s njima putem jasno definiranih sučelja. To omogućava lakšu modifikaciju i nadogradnju bez utjecaja na cijeli sustav.

3.1.1.3. Poboljšana skalabilnost

Zbog svoje modularne prirode, troslojna arhitektura omogućuje skaliranje svakog sloja neovisno, što je posebno korisno u složenim i velikim sustavima. Na primjer, ako je potreban veći kapacitet za rukovanje korisničkim zahtjevima, može se jednostavno proširiti prezentacijski sloj, dok ostali slojevi ostaju nepromijenjeni.

3.1.1.4. Olakšano testiranje i održavanje

Razdvajanje na tri sloja olakšava testiranje aplikacije jer se svaki sloj može testirati neovisno o ostalima. Ovo odvajanje također pojednostavljuje proces otklanjanja pogrešaka, jer je lakše locirati i ispraviti greške unutar jasno definiranog sloja. [11]

3.2. Sloj za pristup bazi podataka

Sloj za pristup bazi podataka (Data Access Layer, DAL) predstavlja temelj aplikacije u kojem se odvija sve što se tiče rukovanja podacima. Ovaj sloj je odgovoran za izravnu komunikaciju s bazom podataka ili drugim sustavima za pohranu podataka. Njegova glavna uloga je omogućiti dohvaćanje, pohranu, ažuriranje i brisanje podataka na način koji je siguran, pouzdan i optimiziran za performanse.

DAL služi kao posrednik između baze podataka i poslovne logike, što znači da poslovni sloj ne mora poznavati specifične detalje o strukturi baze podataka niti načinu na koji se podaci dohvaćaju. Ovo omogućava izolaciju i apstrakciju pristupa podacima, čime se olakšava održavanje i prilagodba baze podataka. Na primjer, ako se baza podataka promijeni ili nadogradi, potrebno je samo prilagoditi sloj za pristup bazi podataka, dok poslovna logika i prezentacijski sloj ostaju nepromijenjeni.

Osim osnovnih operacija kao što su kreiranje, čitanje, ažuriranje i brisanje podataka (CRUD operacije), DAL također upravlja složenijim zadacima kao što su transakcije, povezivanje podataka između različitih entiteta, te implementacija sigurnosnih mjera kao što su autentifikacija i autorizacija pristupa podacima. Također, DAL često uključuje mehanizme za optimizaciju performansi kao što su keširanje podataka, indeksiranje i optimizacija upita prema bazi podataka. [11]

3.3. Sloj poslovne logike

Sloj poslovne logike (Business Logic Layer, BLL) središnji je i najvažniji dio troslojne arhitekture jer se u njemu odvija sva ključna obrada podataka prema poslovnim pravilima. Ovaj sloj je odgovoran za implementaciju svih poslovnih pravila i procesa koje aplikacija mora podržavati. Kada korisnik izvrši neku akciju u prezentacijskom sloju, ta akcija se obrađuje u poslovnom sloju prije nego što se podaci pošalju sloju za pristup baza podataka ili natrag korisniku.

BLL osigurava da su svi podaci i operacije usklađeni s poslovnim pravilima organizacije. To može uključivati validacije unosa, izračune, donošenje odluka temeljenih na zadanim kriterijima, te provjeru i provedbu poslovnih procesa. Na primjer, u aplikaciji kao što je sustav za upravljanje restoranom, poslovni sloj može biti odgovoran za procese poput upravljanja korisničkim računima, obrade zahtjeva korisnika, generiranja statistike, ili provedbe specifičnih poslovnih pravila koja su ključna za funkcioniranje aplikacije.

Sloj poslovne logike također služi kao zaštitni sloj između prezentacijskog sloja i sloja za pristup bazi podataka, omogućujući izolaciju poslovnih pravila od korisničkog sučelja. To znači da promjene u poslovnoj logici mogu biti implementirane bez potrebe za promjenama u prezentacijskom sloju ili u bazi podataka, što osigurava fleksibilnost i lakoću održavanja aplikacije. Osim toga, BLL može uključivati različite servise i poslovne objekte koji međusobno komuniciraju kako bi izvršili složene poslovne zadatke. [11]

3.4. Prezentacijski sloj

Prezentacijski sloj je onaj dio aplikacije koji je u izravnom kontaktu s krajnjim korisnikom. Njegova glavna funkcija je prikazivati podatke i omogućiti korisniku da unosi podatke ili pokreće akcije koje će se obrađivati u poslovnom sloju. Ovaj sloj se fokusira na korisničko iskustvo, pružajući sučelje koje je intuitivno, responzivno i vizualno privlačno.

U prezentacijskom sloju, svi podaci koji dolaze iz poslovnog sloja obrađuju se i prikazuju na način koji je razumljiv korisniku. Ovo uključuje formatiranje teksta, prikaz slika, tablica, grafikona i drugih vizualnih elemenata. Također, prezentacijski sloj upravlja navigacijom unutar aplikacije, omogućavajući korisnicima da se kreću između različitih dijelova aplikacije, unose podatke i komuniciraju s poslovnim procesima.

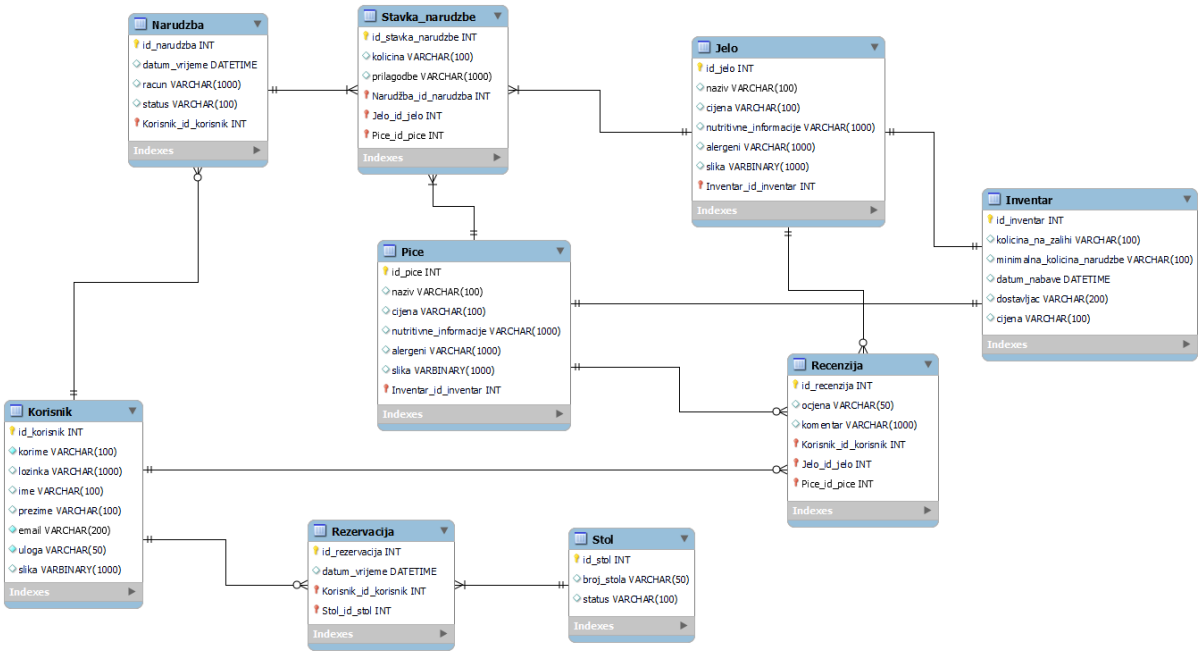
Prezentacijski sloj često uključuje i validaciju podataka prije nego što se oni pošalju u poslovni sloj. Na primjer unutar moje aplikacije, ako korisnik želi ažurirati svoje korisničko ime ili e-mail adresu, prezentacijski sloj može provjeriti ispravnost formata tih podataka prije nego što ih pošalje dalje na obradu. Ovo osigurava da su podaci koji ulaze u sustav već validirani, što smanjuje rizik od pogrešaka i osigurava konzistentnost unosa.

Osim prikaza i unosa podataka, prezentacijski sloj može pružati korisnicima povratne informacije u stvarnom vremenu, kao što su obavijesti o uspjehu ili greškama prilikom obrade podataka, što doprinosi boljem korisničkom iskustvu. [11]

Baza podataka

4.1. ERA model

ERA model (Entity-Relationship-Attribute model) je metodologija za dizajn baza podataka koja se koristi za modeliranje i strukturiranje podataka u obliku entiteta, njihovih odnosa i atributa. ERA model je proširenje Entity-Relationship (ER) modela, koji je razvio Peter Chen 1976. godine, i predstavlja jedan od najvažnijih konceptualnih modela u dizajnu baza podataka. [12]



Slika 1. ERA model (Izvor: samostalna izrada 2024.)

4.2. Opis strukture baze podataka

4.2.1. Entiteti i relacije

U ERA modelu sustava za upravljanje restoranom nalazi se 9 entiteta. Korisnik tablica pohranjuje informacije o korisnicima sustava, bilo da su administratori, osoblje ili obični korisnici. Narudžba tablica bilježi svaku narudžbu koju korisnik kreira. Jedan korisnik može kreirati nula ili više narudžbi. Povezana je i s tablicom Stavka_narudžbe koja predstavlja svaku jedinicu proizvoda koja se nalazi u narudžbi, odnosno, jedno jelo ili jedno piće je jedna stavka narudžbe.

Tablica Jelo i tablica Piće predstavljaju ponudu restorana te su povezane s tablicom Stavka_narudžbe s vezom jedan naprema više, jer isto jelo i isto piće se može više puta pojaviti u istoj narudžbi. Također, tablica Jelo i tablica Piće su povezane sa tablicom Inventar s vezom jedan na jedan. Inventar funkcionira na način da jedan inventar sadrži sve potrebne informacije za jedno Jelo ili jedno Piće, kao što su količina na zalihi, minimalna količina narudžbe, datum nabave, dostavljač i cijena dostave. Znači ako bi u bazi podataka postojalo 10 jela i 10 pića, moralo bi također postojati 20 različitih zapisa inventara u bazi podataka.

Kada korisnik kreira narudžbu, također će se kreirati i rezervacija, pa je tablica Korisnik također povezana i s tablicom Rezervacija. U rezervaciju se sprema datum i vrijeme kada je korisnik odlučio doći u restoran.

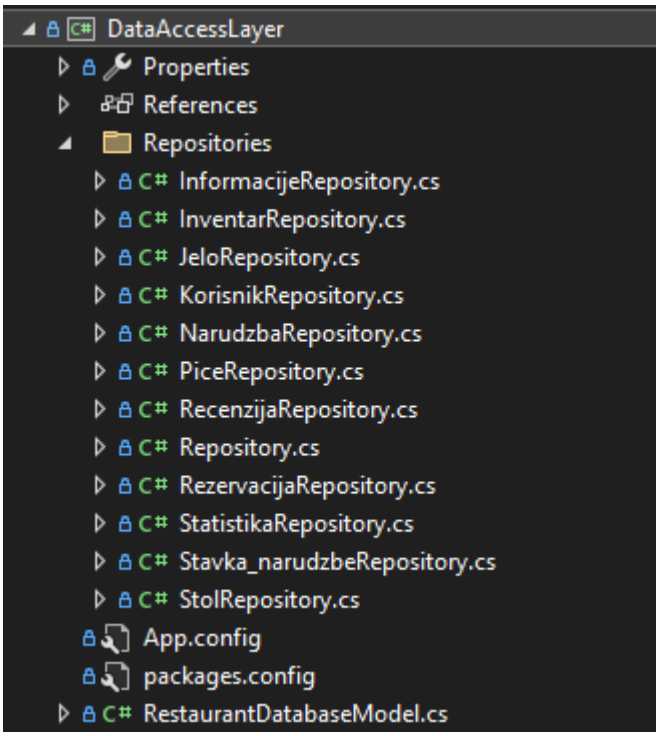
Također, tablica Rezervacija mora biti povezana s tablicom Stol. Stol je tablica u kojoj se nalazi 20 različitih stolova u bazi podataka. Kada korisnik kreira rezervaciju, dodjeljivanje stolova je automatizirano, znači stol će mu se automatski dodijeliti ovisno o zauzetosti stolova. Rezervacije korisnika traju 2 sata. To znači, ako na primjer, jedan korisnik napravi rezervaciju 12. listopada 2024. godine u 16:00 sati, dobit će stol s „Id = 1“. Ako drugi korisnik napravi rezervaciju 12. listopada 2024. godine u 16:30 sati, iz razloga što rezervacije traju 2 sata, dobit će stol s „Id = 2“. Ako treći korisnik napravi rezervaciju 12. listopada 2024. godine u 15:30 sati, dobit će stol s „Id = 3“. Zato što se sve tri narudžbe preklapaju u rasponu od 2 sata, što znači da se ograničava kapacitet restorana na 20 zauzetih stolova u isto vrijeme.

Korisnik također može kreirati recenzije jer je povezan s tablicom Recenzija jedan naprema više. Recenzije funkcioniraju na način da korisnik može dati ocjenu od 1 do 5 i komentar. Recenzije su također povezane s tablicama Jelo i Piće s vezom više naprema jedan. To znači da može postojati više recenzija za svako jelo i piće u bazi podataka.

Implementacija

5.1. Implementacija sloja za pristup bazi podataka

U sloju za pristup bazi podataka koristi se Repository pattern kako bi se omogućila centralizacija i ponovna upotreba koda za pristup podacima. Ovaj uzorak omogućava apstrahiranje operacija nad bazom podataka, čime se olakšava održavanje i proširivost aplikacije. [13]



Slika 2. Sloj za pristup bazi podataka (Izvor: samostalna izrada 2024.)

5.1.1. RestaurantDatabaseModel Klasa

RestaurantDatabaseModel je kontekstna klasa koja nasljeđuje DbContext, osnovnu klasu u Entity Frameworku za rad s bazom podataka. Ova klasa je odgovorna za upravljanje entitetima aplikacije i mapiranje tih entiteta na tablice u bazi podataka. Svaka tablica u bazi podataka predstavlja se kroz DbSet<TEntity> svojstvo unutar ove klase.

```
public partial class RestaurantDatabaseModel : DbContext
{
    public RestaurantDatabaseModel()
        : base("name=RestaurantDatabaseModel")
    {
    }

    public virtual DbSet<Informacije> Informacije { get; set; }
    public virtual DbSet<Inventar> Inventar { get; set; }
    public virtual DbSet<Jelo> Jelo { get; set; }
    public virtual DbSet<Korisnik> Korisnik { get; set; }
    public virtual DbSet<Narudzba> Narudzba { get; set; }
    public virtual DbSet<Pice> Pice { get; set; }
    public virtual DbSet<Recenzija> Recenzija { get; set; }
    public virtual DbSet<Rezervacija> Rezervacija { get; set; }
    public virtual DbSet<Statistika> Statistika { get; set; }
    public virtual DbSet<Stavka_narudzbe> Stavka_narudzbe { get; set; }
    public virtual DbSet<Stol> Stol { get; set; }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
    }
```

Kontekstna klasa RestaurantDatabaseModel sadrži DbSet svojstva koja predstavljaju kolekcije entiteta kao što su Korisnik, Narudžba, Jelo, Piće, itd. Kroz ova svojstva, Entity Framework može pratiti promjene na entitetima, izvršavati upite i ažurirati bazu podataka.

U konstruktoru klase poziva se bazni konstruktor s nazivom konekcijskog stringa RestarauntDatabaseModel, koji definira način povezivanja s bazom podataka. Metoda OnModelCreating može se koristiti za dodatnu konfiguraciju modela, kao što su odnosi između entiteta i ograničenja.

5.1.2. Repository Klasa

Klasa Repository<T> predstavlja apstraktni repozitorij koji sadrži osnovne metode za manipulaciju entitetima tipa „T“, gdje „T“ mora biti klasa. Ova klasa koristi RestaurantDatabaseModel za pristup podacima putem DbSet<T> objekta, koji predstavlja skup entiteta u kontekstu baze podataka.

```
public abstract class Repository<T> : IDisposable where T : class
{
    protected RestaurantDatabaseModel Context { get; set; }
    public DbSet<T> Entities { get; set; }

    public Repository(RestaurantDatabaseModel context)
    {
        Context = context;
        Entities = Context.Set<T>();
    }

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

    public int SaveChanges()
    {
        return Context.SaveChanges();
    }

    public async Task<int> SaveChangesAsync()
    {
        return await Context.SaveChangesAsync();
    }

    public virtual int Add(T entity, bool saveChanges = true)
    {
        Entities.Add(entity);
        if(saveChanges)
        {
            return SaveChanges();
        }
        else
        {
            return 0;
        }
    }
```

```
    public virtual async Task<int> AddAsync(T entity, bool saveChanges =
true)
    {
        Entities.Add(entity);

        if (saveChanges)
        {
            return await Context.SaveChangesAsync();
        }
        else
        {
            return 0;
        }
    }

    public abstract int Update(T entity, bool saveChanges = true);

    public virtual int Remove(T entity, bool saveChanges = true)
    {
        Entities.Attach(entity);
        Entities.Remove(entity);
        if (saveChanges)
        {
            return SaveChanges();
        }
        else
        {
            return 0;
        }
    }

    public void Dispose()
    {
        Context.Dispose();
    }
}
```

Repository klasa pruža osnovne operacije poput dodavanja, ažuriranja, brisanja i dohvaćanja podataka. Svaka metoda je virtualna, što znači da se može nadjačati (override) u izvedenim klasama. Ova klasa također implementira sučelje IDisposable kako bi se osiguralo pravilno otpuštanje resursa kada se repozitorij više ne koristi.

5.1.3. KorisnikRepository Klasa

Kao primjer korištenja metoda iz Repository klase u drugim klasama prikazati ću KorisnikRepository klasu. KorisnikRepository je specifična klasa koja nasljeđuje Repository<Korisnik> i implementira metode specifične za rad s Korisnik entitetom. U ovoj klasi su definirane metode koje omogućuju specifične operacije nad entitetom Korisnik.

Konstruktor KorisnikRepository poziva bazni konstruktor iz Repository klase, prolazeći novu instancu RestaurantDatabaseModel, koja predstavlja kontekst baze podataka. Ovaj kontekst se koristi za sve operacije nad entitetima Korisnik.

```
public class KorisnikRepository : Repository<Korisnik>
{
    public KorisnikRepository() : base(new RestaurantDatabaseModel())
    {

    }
```

GetKorisnikByKorime metoda omogućuje pretraživanje korisnika na temelju korisničkog imena (korime). Koristi se LINQ upit za filtriranje Korisnik entiteta čije „korime“ sadrži zadani string. Metoda vraća IQueryable<Korisnik>, što omogućuje daljnju manipulaciju ili izvršavanje upita prije konačnog izvršenja u bazi podataka.

```
    public IQueryable<Korisnik> GetKorisnikByKorime(string korime)
    {
        var query = from k in Entities
                    where k.korime.Contains(korime)
                    select k;
        return query;
    }
```

Update metoda nadjačava (override) apstraktnu metodu iz Repository klase i omogućuje ažuriranje postojećeg Korisnik entiteta. Metoda prvo dohvaća korisnika na temelju

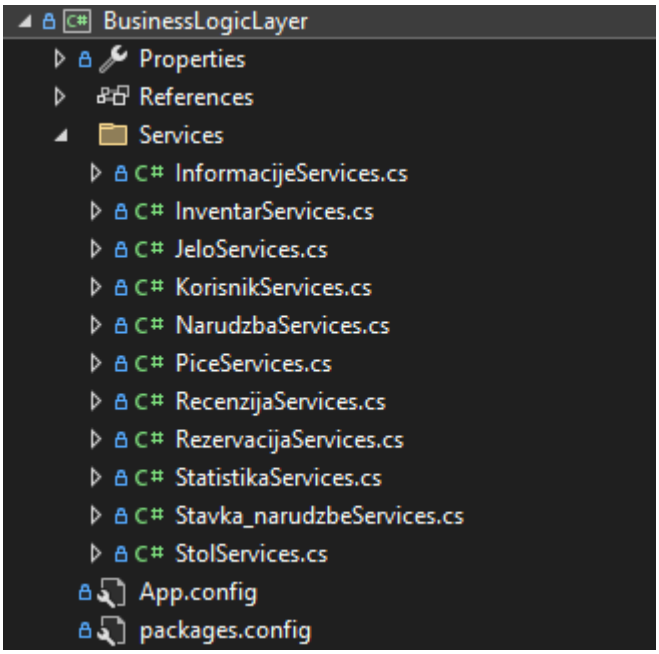
„id_korisnik“, zatim ažurira njegove atribute prema predanim vrijednostima iz „entity“ objekta. Ako je parametar „saveChanges“ postavljen na „true“, izmjene će biti spremljene u bazu podataka.

```
public override int Update(Korisnik entity, bool saveChanges = true)
{
    var korisnik = Entities.SingleOrDefault(k => k.id_korisnik == entity.id_korisnik);

    korisnik.korime = entity.korime;
    korisnik.lozinka = entity.lozinka;
    korisnik.ime = entity.ime;
    korisnik.prezime = entity.prezime;
    korisnik.email = entity.email;
    korisnik.uloga = entity.uloga;
    korisnik.slika = entity.slika;

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

Ovim pristupom implementaciji sloja za pristup bazi podataka postigla se modularnost i ponovna uporebljivost koda. Korištenje Repository patterna omogućava jednostavno proširenje i prilagodbu specifičnim poslovnim zahtjevima, dok istovremeno osigurava siguran i konzistentan pristup podacima. Ključnu ulogu u ovom sustavu igra RestaurantDatabaseModel klasa, koja djeluje kao središte za upravljanje svim entitetima aplikacije i njihovim vezama u bazi podataka.

5.2. Implementacija sloja poslovne logike

Sloj poslovne logike predstavlja srce aplikacije, gdje se implementiraju pravila i procesi poslovanja. Ovaj sloj djeluje kao posrednik između sloja za pristup podacima i korisničkog sučelja, osiguravajući da se poslovna pravila pravilno primjenjuju prilikom obrade podataka. U ovom dijelu aplikacije, poslovna logika se organizira u servisne klase koje grupiraju povezane funkcionalnosti. [13]



Slika 3. Sloj poslovne logike (Izvor: samostalna izrada 2024.)

5.2.1. KorisnikServices Klasa

Za primjer kako funkcioniraju servis klase prikazati ću KorisnikServices klasu. Ova klasa sadrži metode koje upravljaju poslovnim pravilima za entitet Korisnik, uključujući autentifikaciju korisnika, dohvaćanje podataka iz baze i manipulaciju korisničkim računima.

Metoda HashPassword koristi se za stvaranje kriptografski sigurnog sažetka (hasha) lozinke korisnika. Ova metoda kombinira lozinku s unaprijed definiranim „sol“ (salt) vrijednosti kako bi se smanjila ranjivost na napade poput Brute force napada.

```
public class KorisnikServices
{
    private const string fiksnaSol = "moja_fiksna_vrijednost_soli";

    public string HashPassword(string password)
    {
        using (var sha256 = SHA256.Create())
        {
            byte[] saltedPassword = Encoding.UTF8.GetBytes(fiksnaSol +
password);
            byte[] hash = sha256.ComputeHash(saltedPassword);
            return Convert.ToBase64String(hash);
        }
    }
```

Sol je unaprijed definirana vrijednost koja se dodaje na početak lozinke prije nego što se izvrši sažimanje. Na taj način dodaje dodatni sloj sigurnosti.

Algoritam SHA256 koristi se za stvaranje sažetaka lozinke. Ovaj sažetak predstavlja lozinku u obliku niza bajtova.

Konačni sažetak pretvara se u Base64 format, što olakšava njegovo pohranjivanje kao tekst u bazi podataka.

Metoda ValidatePassword koristi se za provjeru ispravnosti unesene lozinke od strane korisnika. Ova metoda prvo sažima unesenu lozinku koristeći isti postupak kao i HashPassword metoda, a zatim uspoređuje rezultat s već pohranjenim sažetkom lozinke.

```
public bool ValidatePassword(string inputPassword, string storedHashPassword)
{
    string hashInputPassword = HashPassword(inputPassword);
    return hashInputPassword == storedHashPassword;
}
```

Sažetak unesene lozinke uspoređuje se s pohranjenim sažetkom lozinke. Ako se podudaraju, lozinka je ispravna i metoda vraća „true“, u suprotnom vraća „false“.

Metoda GetKorisnikByKorime služi za dohvaćanje korisnika iz baze podataka prema korisničkom imenu (korime). Ova metoda koristi sloj za pristup podacima (KorisnikRepository) kako bi izvela upit na bazi podataka.

```
public List<Korisnik> GetKorisnikByKorime(string phrase)
{
    using (var repo = new KorisnikRepository())
    {
 List<Korisnik> korisnik = repo.GetKorisnikByKorime(phrase).ToList();

        return korisnik;
    }
}
```

Metoda koristi KorisnikRepository za filtriranje korisnika pomoću LINQ upita čije korisničko ime sadrži zadanu frazu (phrase). Vraćeni rezultat je lista korisnika koji odgovaraju pretraživanju. Korištenje KorisnikRepository osigurava da poslovna logika ostaje odvojena od direktnog pristupa bazi podataka, pridonoseći modularnosti i održivosti aplikacije.

Metoda AddKorisnik koristi se za dodavanje novog korisnika u bazu podataka. Prije nego što se korisnik doda, lozinka se sažima pomoću metode HashPassword kako bi se osigurala sigurnost pohranjenih podataka.

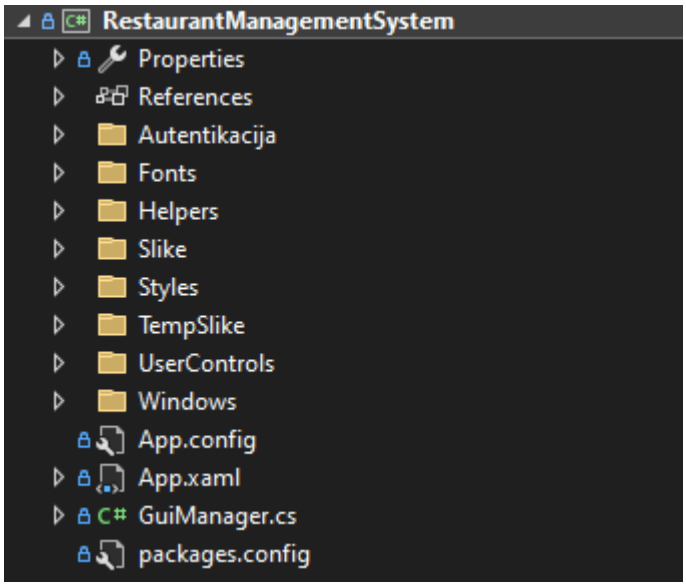
```
public bool AddKorisnik(Korisnik korisnik)
{
    bool isSuccesful = false;
    korisnik.lozinka = HashPassword(korisnik.lozinka);

    using (var repo = new KorisnikRepository())
    {
        int affectedRows = repo.Add(korisnik);
        isSuccesful = affectedRows > 0;

    }

    return isSuccesful;
}
```

Metoda koristi KorisnikRepository kako bi dodala novog korisnika u bazu podataka. Ako je dodavanje uspješno (broj zahvaćenih redaka je veći od 0), metoda vraća „true“.

Sloj poslovne logike ključan je za osiguranje da se svi poslovni procesi i pravila ispravno provode unutar aplikacije. Korištenjem servisnih klasa poput KorisnikServices, poslovna logika postaje centralizirana, modularna i lako održiva. Metode kao što su HashPassword, ValidatePassword, GetKorisnikByKorime i AddKorisnik pokazuju kako se poslovna logika implementiraju u interakciji s bazom podataka, istovremeno osiguravajući sigurnost i konzistentnost podataka. [13]

5.3. Implementacija prezentacijskog sloja

Prezentacijski sloj je dio aplikacije s kojim korisnici direktno komuniciraju. Ovaj sloj je zadužen za prikaz podataka korisnicima i za prikupljanje unosa od korisnika, te interakciju s poslovnom logikom kako bi osigurao pravilnu obradu podataka. Prezentacijski sloj u aplikaciji za upravljanje restoranom omogućuje korisnicima, bilo da su administratori, osoblje ili obični korisnici, da lako pristupe funkcionalnostima aplikacije putem grafičkog korisničkog sučelja (GUI). [13]



Slika 4. Prezentacijski sloj (Izvor: samostalna izrada 2024.)

5.3.1. Komunikacija između Windows-a, User Controla i GuiManagera

Prezentacijski sloj sustava za upravljanje restoranom osmišljen je tako da osigura jednostavnu i intuitivnu interakciju korisnika s aplikacijom. Ključni dijelovi ovog sloja su Windows, User Controle i GuiManager klasa, koja omogućuje centralizirano upravljanje prikazom različitih korisničkih sučelja.

U aplikaciji, svaki tip korisnika ima svoj vlastiti prozor (Window) koji predstavlja početnu točku za rad s aplikacijom. MainWindow za obične korisnike, AdminWindow za administratore te OsobljeWindow za osoblje restorana.

Unutar svakog od ovih prozora, postavljen je ContentControl, koji služi kao kontejner za učitavanje različitih korisničkih kontrola (UserControl). Ovaj ContentControl omogućuje

dinamičko mijenjanje sadržaja unutar prozora, čime se prilagođava trenutnim potrebama korisnika.

```
<ContentControl x:Name="contentPanel" Grid.Row="2" Margin="25">
</ContentControl>
```

GuiManager klasa je statička klasa koja centralizira upravljanje prikazom različitih UserControla unutar aplikacije. Ova klasa sadrži reference na različite prozore (Windows) i omogućuje prebacivanje između različitih kontrola unutar ContentControla.

```
public static class GuiManager
{
    public static MainWindow mainWindow { get; set; }
    public static AdminWindow adminWindow { get; set; }
    public static OsobljeWindow osobljeWindow { get; set; }

    private static UserControl currentContent;
    private static UserControl previousContent;

    public static void SetMainWindow(MainWindow window)
    {
        mainWindow = window;
    }

    public static void SetAdminWindow(AdminWindow window)
    {
        adminWindow = window;
    }

    public static void SetOsobljeWindow(OsobljeWindow window)
    {
        osobljeWindow = window;
    }

    public static void OpenContent(UserControl userControl)
    {
        if (mainWindow != null)
     {
     previousContent = mainWindow.contentPanel.Content as UserControl;
     mainWindow.contentPanel.Content = userControl;
```

```
        currentContent = mainWindow.contentPanel.Content as UserControl;
            }
            else if (adminWindow != null)
            {
        previousContent = adminWindow.contentPanel.Content as UserControl;
        adminWindow.contentPanel.Content = userControl;
        currentContent = adminWindow.contentPanel.Content as UserControl;
        }
        else if (osobljeWindow != null)
        {
      previousContent = osobljeWindow.contentPanel.Content as UserControl;
      osobljeWindow.contentPanel.Content = userControl;
      currentContent = osobljeWindow.contentPanel.Content as UserControl;
        }
        }
        public static void CloseContent()
        {
            if (mainWindow != null)
            {
                OpenContent(previousContent);
            }
            else if (adminWindow != null)
            {
                OpenContent(previousContent);
            }
            else if (osobljeWindow != null)
            {
                OpenContent(previousContent);
            }
        }
        public static void Logout()
        {
            mainWindow = null;
            adminWindow = null;
            osobljeWindow = null;
        }
    }
```

GuiManager omogućuje postavljanje aktivnog prozora pomoću metoda SetMainWindow, SetAdminWindow i SetOsobljeWindow. Na taj način, prozor se centralno

registrira u GuiManageru kako bi se omogućilo upravljanje sadržajem unutar tog prozora i upravljanje pristupom ovisno o ulozi korisnika. U kodu ispod možemo vidjeti kako se dohvaća uloga korisnika te se, ovisno o ulozi, otvara odgovarajući prozor (Window) korisniku, a LoginWindow se zatvara.

```
private void btnLogin_Click(object sender, RoutedEventArgs e)
{
	var korime = txtKorime.Text;
	var lozinka = txtLozinka.Password;
	var pronaden = korisnikServices.GetKorisnikByKorime(korime);

	if (pronaden.Any())
	{
		var korisnik = pronaden.FirstOrDefault();
		if (korisnik != null &&
		korisnikServices.ValidatePassword(lozinka, korisnik.lozinka))
		{
		if(korisnik.uloga == "Običan korisnik")
		{
			MainWindow mainWindow = new MainWindow(korisnik);
			mainWindow.Show();
			GuiManager.SetMainWindow(mainWindow);
			this.Close();
		}
		else if(korisnik.uloga == "Administrator")
		{
			AdminWindow adminWindow = new AdminWindow(korisnik);
			adminWindow.Show();
			GuiManager.SetAdminWindow(adminWindow);
			this.Close();
		}
		else if (korisnik.uloga == "Osoblje")
		{
		OsobljeWindow osobljeWindow = new OsobljeWindow(korisnik);
			osobljeWindow.Show();
			GuiManager.SetOsobljeWindow(osobljeWindow);
			this.Close();
		}
	}
```

Metoda OpenContent omogućuje otvaranje određene UserControle unutar aktivnog prozora. Ova metoda također pamti prethodni sadržaj kako bi se omogućio povratak na prethodno stanje pomoću metode CloseContent. Ja sam u svojoj aplikaciji većinom koristio navigacijski izbornik [18] kao način upravljanja sa UserControlama, ali ako se želi implementirati povratak na prethodni prozor, koristit će se metoda CloseContent.

Metoda Logout poništava sve reference na prozore, čime se osigurava da se korisnik može ispravno odjaviti iz aplikacije bez brige da će se prozori pomiješati.

Navigacija između različitih UserControla unutar prozora realizirana je na način da se na određeni događaj, u našem slučaju klikom na gumb, pokrene metoda koja koristi GuiManager za zamjenu sadržaja unutar ContentControla.

Na primjer, unutar MainWindow klase, navigacija do određene kontrole može izgledati ovako:

```
private void PregledJelovnikaButton_Click(object sender,
RoutedEventArgs e)
{
   pageIcon.Icon = IconChar.Utensils;
   pageTitle.Text = "Pregled jelovnika";
   var ucPregledJelovnika = new UcPregledJelovnika(TrenutniKorisnik);
   GuiManager.OpenContent(ucPregledJelovnika);
}
```

Unutar same UserControl kontrole, također je moguće realizirati navigaciju na drugu kontrolu. Najprije se postavi grid sa određenom identifikacijom, a zatim se dijete grida zamjeni sa novom UserControlom.

```
<Grid Name="glavniGrid">
</Grid>

private void Jelo_Click(object sender, RoutedEventArgs e)
{
    SelectedJelo = (sender as FrameworkElement)?.DataContext as Jelo;

    if (SelectedJelo != null)
    {
        UcDetaljiJela detaljiJela = new UcDetaljiJela(SelectedJelo,
trenutniKorisnik);
        detaljiJela.DataContext = SelectedJelo;
```

```
            glavniGrid.Children.Clear();
            glavniGrid.Children.Add(detaljiJela);
        }
    }
```

5.3.2. Implementacija spremanja slika u bazu podataka i prikazivanje u aplikaciji

U aplikaciji postoji mogućnost učitavanja, prikazivanja i spremanja slika u bazu podataka, kao i dohvaćanja i prikazivanja tih slika iz baze podataka. Ovdje ću objasniti kako se to postiže koristeći WPF kontrole i .NET klasu BitmapImage.

5.3.2.1. Učitavanje slike s diska

Prvi korak je omogućiti korisniku da odabere sliku sa svog računala i prikaže je u aplikaciji. Za to koristimo OpenFileDialog, koji omogućava korisniku odabir datoteke s određenim ekstenzijama, poput .jpg, .jpeg, .png,… Nakon što korisnik odabere sliku, ona se učitava kao BitmapImage i prikazuje se u kontroli Image. [21]

```
private void btnOdaberiSliku_Click(object sender, RoutedEventArgs e)
{
    var openFileDialog = new OpenFileDialog
    {
        Filter = "Image files (*.jpg, *.jpeg,
      *.png)|*.jpg;*.jpeg;*.png"
    };
    if (openFileDialog.ShowDialog() == true)
    {
        BitmapImage bitmap = new BitmapImage();
        bitmap.BeginInit();
        bitmap.UriSource = new Uri(openFileDialog.FileName,
      UriKind.Absolute);
        bitmap.EndInit();
        imgSlika.Source = bitmap;
        isImageChanged = true;
    }
}
```

OpenFileDialog omogućava korisniku odabir slike s računala. Korisniku se samo prikazuju slike sa zadanim ekstenzijama unutar Filtera. Ako korisnik uspješno odabere sliku, stvara se novi BitmapImage objekt [21] i inicijalizira se s odabranom slikom. Svojstvo UriSource postavlja se na putanju odabrane slike. Nakon toga, slika se prikazuje u Image kontroli (imgSlika). Varijabla isImageChanged koristi se za praćenje je li slika promijenjena, što može biti korisno pri spremanju slike u bazu podataka. [22]

5.3.2.2. Konverzija slike u oblik za pohranu u bazu podataka

Budući da se slike ne mogu direktno pohraniti u bazu podataka u formatu BitmapImage, potrebno je sliku konvertirati u niz bajtova (byte[]). Za to koristimo metodu ImageToByte.

```
private byte[] ImageToByte(BitmapImage bitmapImage)
{
    byte[] data;
    JpegBitmapEncoder encoder = new JpegBitmapEncoder();
    encoder.Frames.Add(BitmapFrame.Create(bitmapImage));
    using (MemoryStream ms = new MemoryStream())
    {
        encoder.Save(ms);
        data = ms.ToArray();
    }
    return data;
}
```

JpegBitmapEncoder se koristi za enkodiranje BitmapImage objekta u format JPEG. Slika se enkodira i sprema u memorijski tok (MemoryStream), a zatim se niz bajtova (byte[]) ekstrahira iz memorijskog toka. Ovaj niz bajtova može se potom pohraniti u bazu podataka kao varbinary ili sličan tip podataka. [22]

5.3.2.3. Konverzija oblika slike nazad u oblik za prikaz

Kada dohvaćamo sliku iz baze podataka, ona je pohranjena kao niz bajtova (byte[]). Da bismo je prikazivali korisniku, potrebno je ovaj niz bajtova ponovno konvertirati u BitmapImage.

```
private BitmapImage ByteToImage(byte[] imageData)
{
    using (MemoryStream ms = new MemoryStream(imageData))
    {
        BitmapImage bitmap = new BitmapImage();
        bitmap.BeginInit();
        bitmap.StreamSource = ms;
        bitmap.CacheOption = BitmapCacheOption.OnLoad;
        bitmap.EndInit();
        return bitmap;
    }
}
```

MemoryStream se koristi za čitanje niza bajtova iz baze podataka. Stvara se novi BitmapImage objekt, kojem se kao izvor (StreamSource) postavlja memorijski tok koji sadrži podatke o slici. Nakon što se BitmapImage inicijalizira, vraća se i može se koristiti za prikaz slike u korisničkom sučelju. [22]

5.3.3. Implementacija IMemoryCache

IMemoryCache je usluga dostupna u .NET-u koja omogućava pohranu podataka u memoriju na strani poslužitelja, čime se omogućuje brzo dohvaćanje podataka bez potrebe za ponovnim dohvaćanjem istih iz baze podataka ili drugih sporijih izvora. Ovo može značajno ubrzati aplikaciju, posebice u situacijama kada se isti podaci koriste više puta u kratkom vremenskom periodu. [14]

U svome sustavu za upravljanje restoranom, koristio sam IMemoryCache kako bi ubrzao dohvaćanje podataka iz baze podataka, a za primjer ću pokazati kako sam ubrzao dohvaćanje hrane i pića iz baze podataka prilikom pretrage izbora kod kreiranja rezervacija, odnosno narudžbi.

Kada korisnik unese najmanje tri slova u polje za pretragu hrane (FoodFilterTextBox) ili pića (DrinkFilterTextBox), pokreće se sljedeći postupak:

Prvo se provjerava postoji li već filtrirani popis hrane ili pića u memoriji, koristeći ključ koji kombinira naziv filtera s prefiksom (FilteredFoodItems_ za hranu i (FilteredDrinkItems_ za pića). Ako filtrirani popis postoji u cache-u, odmah se koristi ta verzija popisa, čime se izbjegava ponovno dohvaćanje podataka iz baze podataka. Ako filtrirani popis nije pronađen

u cache-u, podaci se dohvaćaju iz baze podataka pomoću odgovarajućih servisnih klasa (_jeloServices za hranu i _piceServices za pića). Nakon dohvaćanja podataka iz baze podataka, oni se pohranjuju u cache s istim ključem kako bi sljedeći put bili brzo dostupni. Na kraju, filtrirani popis prikazuje se u odgovarajućem ListView elementu (FoodListView ili DrinkListView).

```
private async void FoodFilterTextBox_TextChanged(object sender,
TextChangedEventArgs e)
{
    string filter = FoodFilterTextBox.Text.Trim();
    if (filter.Length >= 3)
    {
            if (_cache.TryGetValue("FilteredFoodItems_" + filter, out
            List<Jelo> cachedFilteredFoodItems))
        {
            _filteredFoodItems = cachedFilteredFoodItems;
        }
        else
        {
            _filteredFoodItems = await
            _jeloServices.GetJelaByNameAsync(filter);
            _cache.Set("FilteredFoodItems_" + filter,
            _filteredFoodItems);
        }

        FoodListView.ItemsSource = _filteredFoodItems;
    }
    else
    {
        FoodListView.ItemsSource = null;
        _filteredFoodItems = null;
    }
}
```

Ovakav pristup omogućuje da aplikacija bude brža i responzivnija, osobito kada korisnici često pretražuju slične pojmove.

5.3.3.1. Prednosti korištenja IMemoryCache

Smanjuje broj upita prema bazi podataka, što smanjuje opterećenje na poslužitelju i poboljšava vrijeme odaziva aplikacije. IMemoryCache je jednostavan za implementaciju i korištenje, posebno u scenarijima gdje je potrebno privremeno pohraniti podatke koji se često ponavljaju. Omogućuje jednostavno upravljanje memorijom i pohranom podataka koji su specifični za aplikaciju ili korisničku sesiju. [14]

5.3.4. Implementacija validacije korisničkog unosa

Validacija korisničkog unosa ključan je dio svake aplikacije jer osigurava da podaci uneseni od strane korisnika zadovoljavaju određene kriterije prije nego što budu pohranjeni ili obrađeni. [15] U svojoj aplikaciji, koristio sam metodu „provjeriUnose()“ kako bi osigurao ispravnost podataka prije njihove obrade.

U primjeru koji ću prikazati, metoda „provjeriUnose()“ se koristi za provjeru ispravnosti unosa korisnika prilikom ažuriranja svojih podataka na UserControli profila. Validacija uključuje provjeru korisničkog imena, lozinke, imena, prezimena i email adrese.

Metoda vraća string koji sadrži sve poruke o pogreškama, a ako su svi unosi ispravni, vraća prazan string.

```
private string provjeriUnose()
{
    StringBuilder errorMessage = new StringBuilder();
    string korisnickoIme = txtKorime.Text;
    string lozinka = txtLozinka.Text;
    string ime = txtIme.Text;
    string prezime = txtPrezime.Text;
    string email = txtEmail.Text;

    bool isKorisnickoImeValid = Regex.IsMatch(korisnickoIme,
    @"^[a-zA-Z0-9_]{3,20}$");
    bool isLozinkaValid = Regex.IsMatch(lozinka,
    @"^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*?&]{5,}$");
    bool isImeValid = Regex.IsMatch(ime,
    @"^[a-zA-Z\s-]{1,}$");
    bool isPrezimeValid = Regex.IsMatch(prezime,
    @"^[a-zA-Z\s-]{1,}$");
    bool isEmailValid = Regex.IsMatch(email,
    @"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$");
```

```
            if (!isKorisnickoImeValid)
            {
                errorMessage.AppendLine("Korisničko ime mora imati između 3 i 20 znakova, i može sadržavati slova, brojeve i donje crte.");
            }
            if (!isLozinkaValid)
            {
                errorMessage.AppendLine("Lozinka mora imati najmanje 5 znakova, jedan broj i jedno slovo. Može imati i znakove '@$!%*?&'.");
            }
            if (!isImeValid)
            {
                errorMessage.AppendLine("Ime može sadržavati samo slova, znak '-' i razmake.");
            }
            if (!isPrezimeValid)
            {
                errorMessage.AppendLine("Prezime može sadržavati samo slova, znak '-' i razmake.");
            }
            if (!isEmailValid)
            {
                errorMessage.AppendLine("Email adresa nije u ispravnom formatu.\n     - Primjer maila: ime.prezime5@gmail.com");
            }

            return errorMessage.ToString();
        }
```

Korisničko ime mora imati između 3 i 20 znakova. Dozvoljeni znakovi su slova, brojevi i donje crte. Lozinka mora imati najmanje 5 znakova. Mora sadržavati barem jedno slovo i jedan broj. Dozvoljeni su i posebni znakovi „@$!%*?&“. Ime i prezime smiju sadržavati samo slova, razmake i znak '-'. Ime i prezime moraju imati barem jedan znak. Email adresa mora biti u ispravnom formatu kao tipična email adresa, na primjer „ime.prezime@gmail.com“.

Ako je lozinka unesena i validna, koristi se metoda HashPassword iz servisne klase korisnika za hashiranje lozinke prije nego što se pohrani. Ako lozinka nije unesena, zadržava se prethodno pohranjena lozinka. Implementirao sam da se lozinka nikad ne prikazuje korisniku zbog dodatnog sloja sigurnosti. [16]

5.3.4.1. Prednosti validacije korisničkog unosa

Metoda „provjeriUnose()“ centralizira validaciju, što olakšava održavanje i nadogradnju pravila validacije. Korisnik dobiva jasne i specifične poruke o tome koji unosi nisu ispravni, što poboljšava korisničko iskustvo.

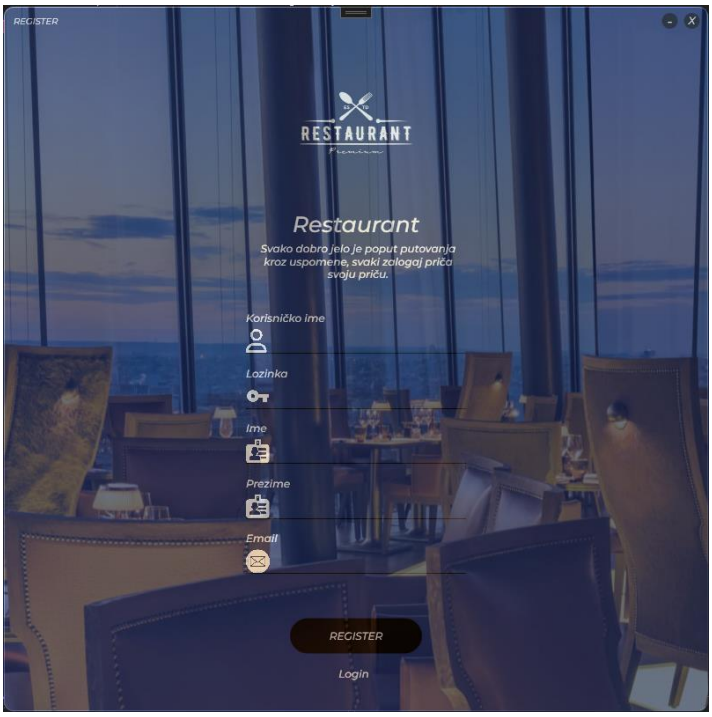
Validacijom na strani poslužitelja smanjuje se mogućnost da nepravilni ili zlonamjerni podaci uđu u sustav. Validacija unosa prije nego što se podaci obrade ili pohrane sprječava potencijalne greške ili iznimke koje bi mogle nastati zbog neispravnih podataka.

Validacija unosa, posebice za lozinke i email adrese, smanjuje rizik od sigurnosnih propusta poput SQL injekcija ili upada u sustav. Pravila validacije mogu se lako proširiti i promijeniti na jednom mjestu, što omogućuje jednostavno prilagođavanje budućim zahtjevima ili standardima.

5.4. Implementacija funkcionalnosti

5.4.1. Login i Registracija

LoginWindow i RegistrationWindow su bitan dio aplikacije koji omogućavaju korisnicima da se prijave ili registriraju za pristup sustavu. Proces autentifikacije osigurava da samo ovlašteni korisnici mogu pristupiti određenim dijelovima aplikacije. Registracija omogućava novim korisnicima stvaranje računa, pri čemu se unose osnovni podaci poput korisničkog imena, lozinke, email adrese i drugih potrebnih informacija. [17]

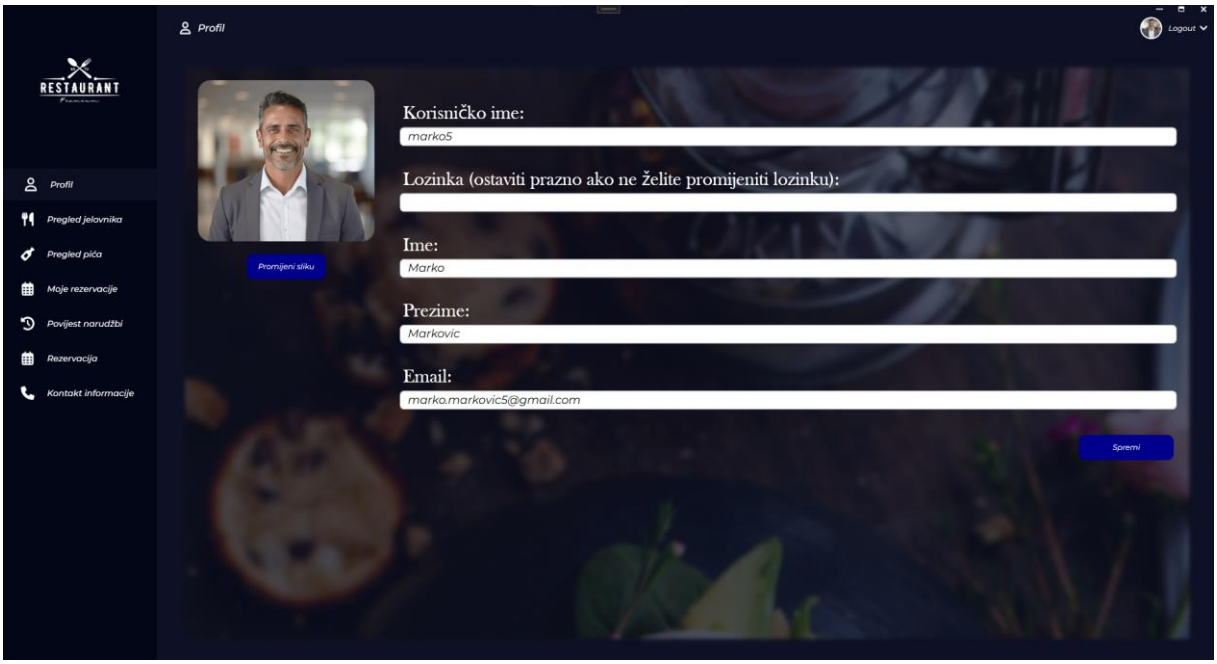


Slika 5. Prozor registracije (Izvor: samostalna izrada 2024.)

Korisnici se prijavljuju svojim korisničkim imenom i lozinkom. Lozinka se hashira kako bi se mogla usporediti sa hashiranom lozinkom u bazi podataka koja je kreirana za vrijeme registracije. Za hashiranje koristi se SHA256 algoritam. Nakon uspješne prijave, korisnicima se prikazuju prozori (Windows) specifični za njihovu ulogu.

5.4.2. Profil korisnika

Korisnički profil omogućuje svakom korisniku da pregleda i ažurira svoje osobne podatke unutar aplikacije. Kroz ovu funkcionalnost, korisnici mogu održavati svoje podatke ažuriranima, kao i mijenjati postavke računa prema vlastitim potrebama.

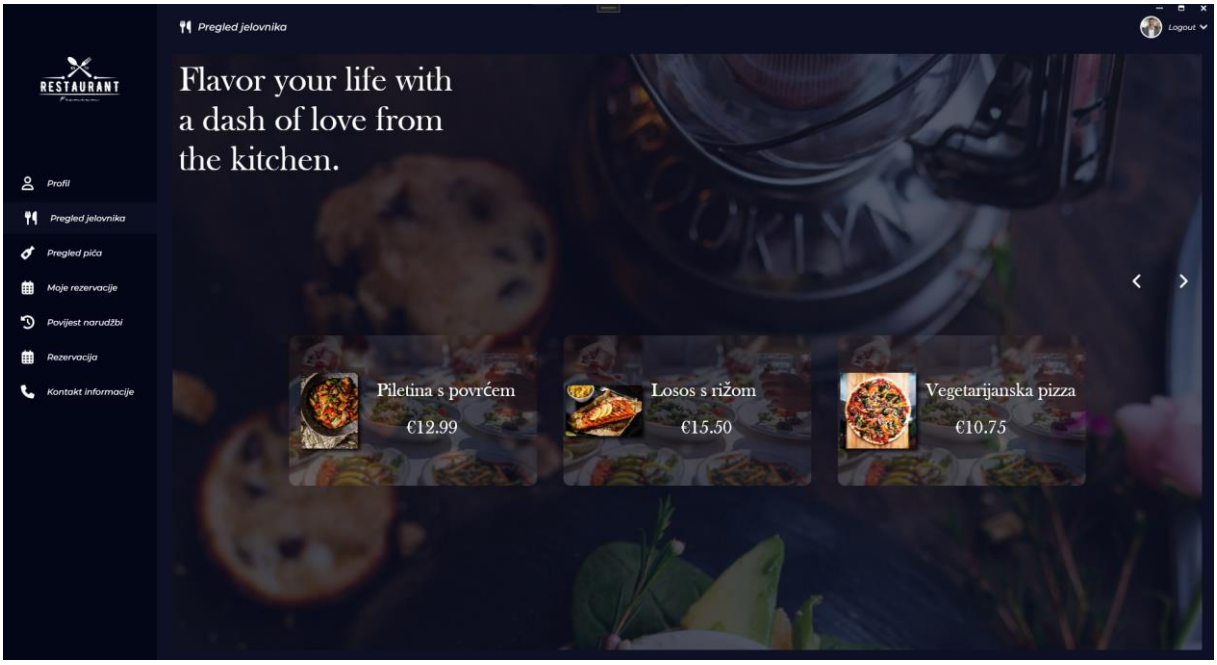


Slika 6. Profil korisnika (Izvor: samostalna izrada 2024.)

Kao što vidite na slici, lozinka se na profilu ne prikazuje. Prilikom izmjene korisničkih podataka, ako se ne želi promijeniti lozinka, ona se samo neće unijeti od strane korisnika. Ako se želi promijeniti, samo se u PasswordBox lozinke upiše nova lozinka, koja se također ne prikazuje kao stvarna vrijednost nego kao točkice. Odlučio sam se za ovakav pristup jer ne želim da se na korisničkom profilu prikazuje hashirana verzija lozinke, niti želim da je lozinku iz baze podataka ikad potrebno vraćati u pravi oblik radi sigurnosti.

5.4.3. Pregled Jelovnika i Pregled Pića

Funkcionalnost pregleda jelovnika i pića omogućuje korisnicima pregled dostupnih jela i pića u restoranu. Kroz ovu opciju, korisnici mogu jednostavno pronaći i odabrati artikle koji ih zanimaju, te na taj način mogu pregledati detalje o svakom pojedinom jelu i piću, kao što su nutritivne informacije, cijena, alergeni i slično.



Slika 7. Pregled jelovnika i pića (Izvor: samostalna izrada 2024.)

Dohvaćena jela i pića postavljaju se u svoje kartice u kojima se prikazuju nekakvi osnovni podaci kao što su naziv, cijena i slika. Za optimizaciju prikaza velikog broja artikala, implementirana je funkcionalnost paginacije. Hard kodirano je da se mogu prikazati najviše tri jela ili pića na jednoj stranici, a do ostalih se može doći putem paginacije.

```
private int currentPage = 0;
private int itemsPerPage = 3;
CurrentPageJela = new ObservableCollection<Jelo>();

private void PrikaziStranicu()
{
    if (isLoading)
    {
        loadingText.Visibility = Visibility.Visible;
```

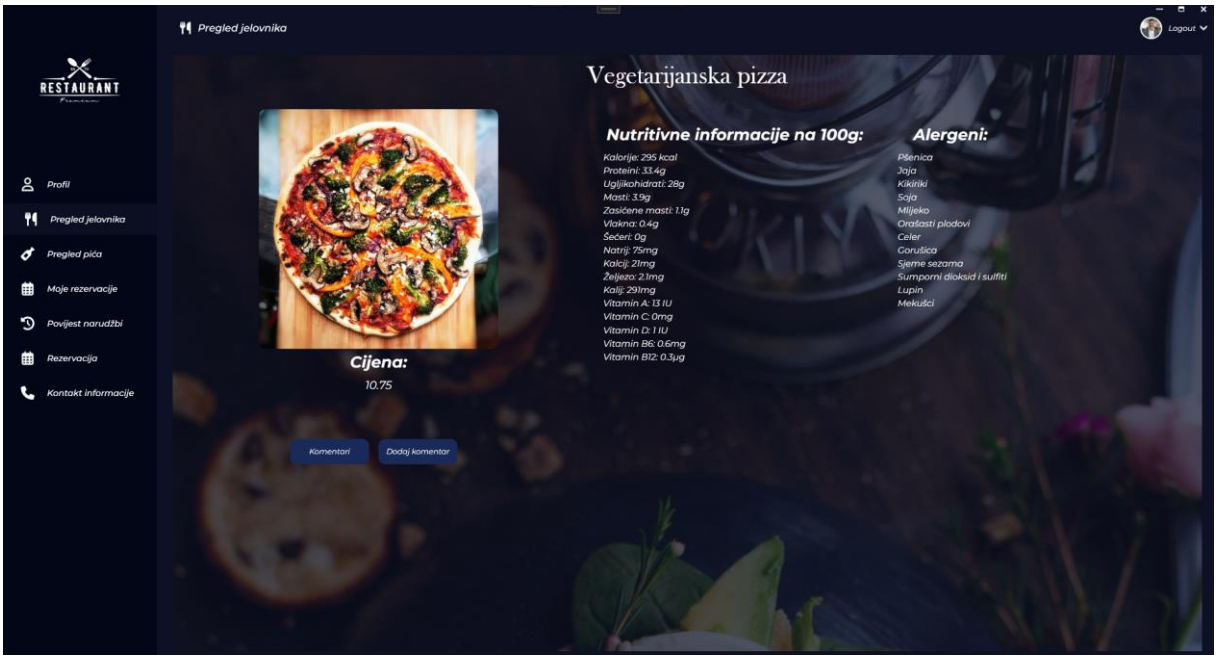
```
            return;
        }

        CurrentPageJela.Clear();

        if (cache.TryGetValue("SvaJelaCache", out List<Jelo>
        SvaJelaCache))
        {
          var items = SvaJelaCache.Skip(currentPage *
          itemsPerPage).Take(itemsPerPage).ToList();

            foreach (var item in items)
            {
                CurrentPageJela.Add(item);
            }

            loadingText.Visibility = Visibility.Collapsed;
        }
        else
        {
            loadingText.Visibility = Visibility.Visible;
            CurrentPageJela.Clear();
        }

        UpdateButtons();
    }
    private void UpdateButtons()
    {
        PrevButton.IsEnabled = currentPage > 0;
        NextButton.IsEnabled = (currentPage + 1) * itemsPerPage <
(cache.Get<List<Jelo>>("SvaJelaCache")?.Count ?? 0);
    }
    private void PrevButton_Click(object sender, RoutedEventArgs e)
    {
        if (currentPage > 0)
        {
            currentPage--;
            PrikaziStranicu();
        }
    }
    private void NextButton_Click(object sender, RoutedEventArgs e)
```

```
{
    if (!isLoading)
    {
        currentPage++;
        PrikaziStranicu();
    }
    else
    {
        CurrentPageJela.Clear();
        loadingText.Visibility = Visibility.Visible;
        currentPage++;
    }
}
```

Klikom na jedan od ponuđenih jela ili pića odvesti će nas, pomoću postavljanja novog glavnog grida, na stranicu za prikaz detalja jela ili detalja pića. [20]

5.4.4. Detalji Jela i Detalji Pića

Stranica za prikaz detalja jela i pića omogućuje korisnicima da pregledaju sve bitne informacije o određenom artiklu, uključujući sliku, cijenu, nutritivne vrijednosti, alergene, te recenzije drugih korisnika. Također, omogućuje dodavanje novih komentara i ocjena za svaki artikl.



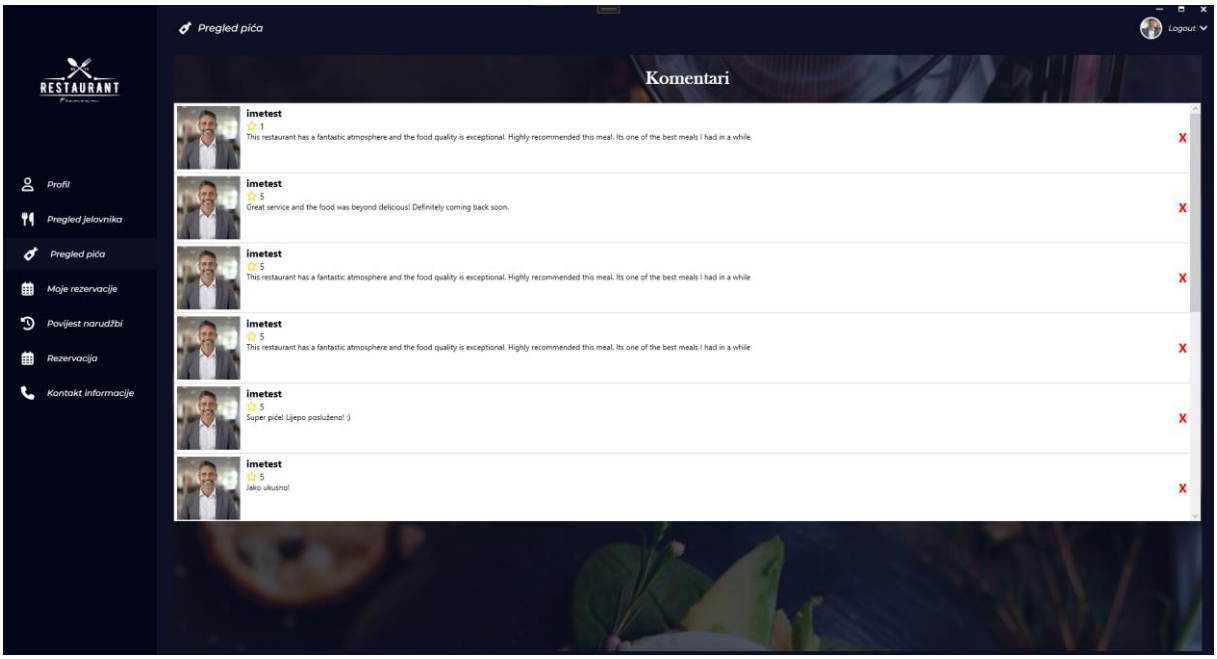
Slika 8. Detalji jela i pića (Izvor: samostalna izrada 2024.)

Slika 9. Dodavanje komentara (Izvor: samostalna izrada 2024.)

Klikom na gumb „Komentari“, aplikacija će nas odvesti na UserControl koji služi za prikaz svih komentara za jelo ili piće na kojemu smo se nalazili.

5.4.5. Komentari Jela i Komentari Pića

Navedene funkcionalnosti omogućuju pregledavanje i interakciju s komentarima ostavljenim na određenom jelu ili piću. Komentari su prikazani u ListView-u, gdje se svaki komentar može vidjeti i predstavlja jedan item u ListView. Na komentaru se mogu vidjeti slika korisnika, korisničko ime, te ocjena i komentar koje je korisnik objavio.



Slika 10. Komentari jela i pića (Izvor: samostalna izrada 2024.)

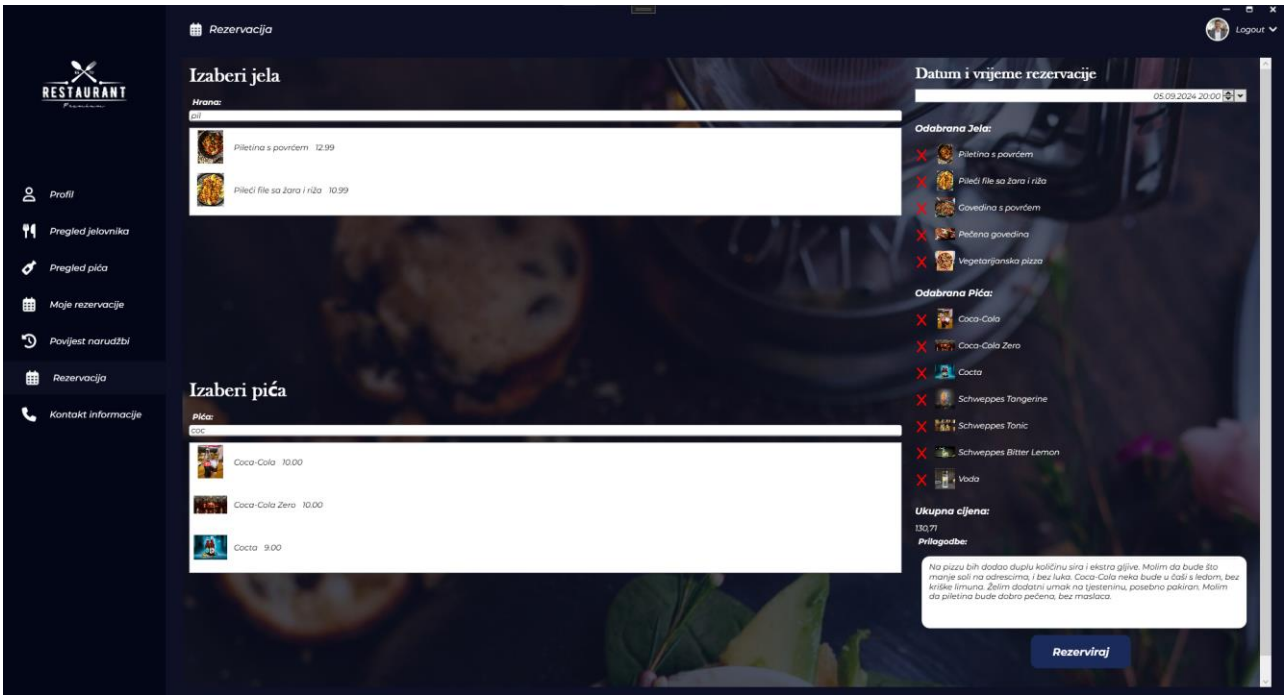
Opcija za brisanje komentara prikazuje se pokraj svakog komentara. Kada korisnik pritisne na opciju za brisanje, provjerava se vlasništvo komentara te uloga korisnika koji je objavio komentar. Komentar je korisnik u mogućnosti obrisati samo ako je njegov komentar, s iznimkom da administrator može brisati i tuđe komentare. Ako korisnik želi pregledati sve komentare, može se koristiti scrollanjem kroz ListView. Ova funkcionalnost osigurava da je pregled komentara intuitivan i jednostavan, bez potrebe za dodatnim navigacijskim alatima.

5.4.6. Narudžba i Rezervacija

Na stranici za rezervaciju i narudžbu nalazi se nekoliko ključnih elemenata koji korisnicima omogućuju jednostavno i intuitivno odabiranje i prilagođavanje svojih narudžbi. Koristi se ListView za prikaz jela i pića, gdje korisnici mogu pregledavati dostupne artikle, pretraživati ih i odabrati one koje žele naručiti. Uz narudžbu moguće je opcionalno popuniti i sekciju za prilagodbe narudžbe, gdje korisnici mogu unijeti specifične zahtjeve ili promjene (npr. Dodatni sastojci, način pripreme).

Nakon odabira jela i pića, korisnici mogu odabrati datum i vrijeme rezervacije putem intuitivnog sučelja za odabir datuma i vremena. Svi odabrani artikli i prilagodbe se prikazuju na zaslonu u sekciji prikaza odabranih jela i pića, gdje je vidljiva i ukupna cijena narudžbe, koja se automatski ažurira prilikom biranja artikala. Kraj svakog artikla također se automatski kreira i gumb za uklanjanje jela ili pića iz košarice. Za finalizaciju narudžbe, korisnici klikom na gumb „Rezerviraj“ potvrđuju rezervaciju, pri čemu se kreira narudžba, rezervacija te se zauzima stol u restoranu za odabrani datum i vrijeme.

Postoji 20 stolova u bazi podataka. Kada korisnik kreira rezervaciju, dodjeljivanje stolova je automatizirano, znači stol će mu se automatski dodijeliti ovisno o zauzetosti stolova. Rezervacije korisnika traju 2 sata. To znači, ako na primjer, jedan korisnik napravi rezervaciju 12. listopada 2024. godine u 16:00 sati, dobit će stol s „Id = 1“. Ako drugi korisnik napravi rezervaciju 12. listopada 2024. godine u 16:30 sati, iz razloga što rezervacije traju 2 sata, dobit će stol s „Id = 2“. Ako treći korisnik napravi rezervaciju 12. listopada 2024. godine u 15:30 sati, dobit će stol s „Id = 3“. Zato što se sve tri narudžbe preklapaju u rasponu od 2 sata, što znači da se ograničava kapacitet restorana na 20 zauzetih stolova u isto vrijeme.



Slika 11. Narudžba i rezervacija (Izvor: samostalna izrada 2024.)

U kodu ću sada objasniti kako funkcionira automatizirano dodjeljivanje stolova prilikom rezervacije, odnosno kreiranja narudžbe.

```
    DateTime endTime = selectedDateTime.Value.AddHours(2);

var sveRezervacije = await _rezervacijaServices.GetAllRezervacijeAsync();
    var overlappingReservations = sveRezervacije
    .Where(r => r.datum_vrijeme < endTime &&
    r.datum_vrijeme.Value.AddHours(2) > selectedDateTime)
    .ToList();

    var stolovi = await _stolServices.GetSlobodneStolove();
    int? tableId = null;
    foreach (var stol in stolovi)
    {
    bool isTableAvailable = !overlappingReservations.Any(r =>
    r.Stol_id_stol == stol.id_stol);
        if (isTableAvailable)
        {
            tableId = stol.id_stol;
            break;
        }
```

```
}

if (tableId == null)
{
    System.Windows.Forms.MessageBox.Show("Nema dostupnog stola za
    odabrani termin.");
    return;
}
Rezervacija novaRezervacija = new Rezervacija
{
    datum_vrijeme = selectedDateTime.Value,
    Korisnik_id_korisnik = korisnik.id_korisnik,
    Stol_id_stol = tableId.Value
};
_rezervacijaServices.AddRezervaciju(novaRezervacija);
```

Najprije se izračunava krajnje vrijeme rezervacije dodavanjem dva sata od odabranog vremena početka rezervacije. To omogućava dovoljno vremena za obrok i osigurava da rezervacija neće trajati predugo. Iz baze podataka se dohvaćaju sve rezervacije koje se preklapaju s odabranim terminom. Rezervacija se smatra preklapajućom ako njezino početno vrijeme pada unutar vremenskog okvira trenutne rezervacije. Sljedeći korak je dohvaćanje svih stolova koji su trenutno slobodni, to jest nisu zauzeti ili rezervirani za odabrano vrijeme. Aplikacija prolazi kroz listu slobodnih stolova i provjerava za svaki od njih postoji li preklapajuća rezervacija. Ako se pronađe stol koji nije rezerviran u preklapajućem terminu, taj stol se automatski dodjeljuje korisniku. Ako je dostupan stol pronađen, aplikacija automatski stvara novu rezervaciju s dodijeljenim stolom, korisnikom koji je napravio rezervaciju i odabranim vremenom. Ako nijedan stol nije dostupan, korisnik dobiva obavijest da nema slobodnih stolova za odabrani termin.

Košarica odabranih artikala za rezervaciju ili narudžbu upravlja se pomoću dvije kolekcije [20], jedna za hranu te jedna za pića. Ove kolekcije omogućuju korisnicima da dodaju ili uklone artikle iz svoje narudžbe, dok se cijena automatski ažurira.

```
private ObservableCollection<Jelo> _selectedFoodItems = new
ObservableCollection<Jelo>();
private ObservableCollection<Pice> _selectedDrinkItems = new
ObservableCollection<Pice>();
```

Metoda AddSelectedJelo omogućuje dodavanje odabranog jela u košaricu. Kada korisnik odabere jelo, ono se dodaje u kolekciju jela. Zatim se osvježava prikaz odabranih jela u korisničkom sučelju. Metoda CalculateTotalPrice se poziva kako bi se izračunala ukupnu cijenu narudžbe na temelju dodatnih artikala.

```
private void AddSelectedJelo(Jelo selectedJelo)
{
    _selectedFoodItems.Add(selectedJelo);
    SelectedFoodItemsControl.ItemsSource = null;
    SelectedFoodItemsControl.ItemsSource = _selectedFoodItems;
    CalculateTotalPrice();

}
```

Metoda RemoveJeloButton_Click omogućuje korisnicima da uklone jelo iz košarice. Ova metoda provjerava koji je artikl povezan s pritisnutim gumbom za uklanjanje putem „Tag“ svojstva. Nakon uklanjanja jela iz kolekcije, sučelje se ponovno osvježava i ukupna cijena narudžbe se automatski ažurira.

```
private void RemoveJeloButton_Click(object sender, RoutedEventArgs e)
{
    if (sender is Button button && button.Tag is Jelo selectedJelo)
    {
        _selectedFoodItems.Remove(selectedJelo);
        SelectedFoodItemsControl.ItemsSource = null;
        SelectedFoodItemsControl.ItemsSource = _selectedFoodItems;
        CalculateTotalPrice();
    }
}
```

Na kraju procesa narudžbe, metoda GenerateRacun koristi StringBuilder kako bi generirala detaljan račun. Na računu se nalaze svi artikli iz košarice zajedno s njihovim pojedinačnim cijenama, te konačna ukupna cijena narudžbe. Ovaj račun je spremljen u bazu podataka unutar narudžbe te se po njemu priprema i naplaćuje narudžba.

5.4.7. Povijest narudžbi i Moje rezervacije

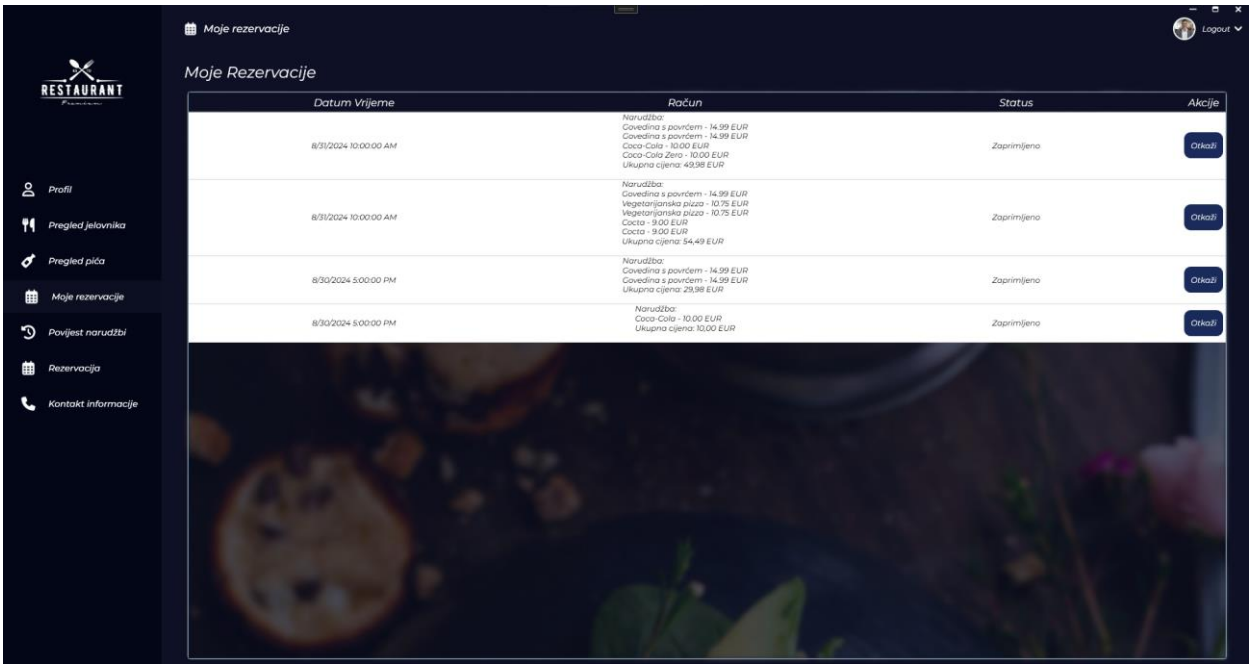
Aplikacija pruža korisnicima mogućnost pregleda njihovih prošlih narudžbi i aktualnih rezervacija putem dvaju korisničkih kontrola, a to su „Povijest narudžbi“ i „Moje recenzije“. Iako su ove funkcionalnosti slične u prikazu informacija, postoje ključne točke koje ih razlikuju.

Na obije korisničke kontrole implementiran je DataGridView kao jednostavan prikaz informacija, te je implementirano scrollanje kroz sučelje zbog jednostavnog i intuitivnog pregleda svih narudžbi i rezervacija.

5.4.7.1. Povijest narudžbi

Korisničko sučelje „Povijest narudžbi“ omogućuje korisnicima pregled svih narudžbi i rezervacija koje su ikada napravili. Ova funkcionalnost je korisna za pregled prethodnih narudžbi, uvid u detalje narudžbi te praćenje povijesti troškova.

Na ovoj korisničkoj kontroli prikazuju se datum i vrijeme narudžbe, odnosno rezervacije, račun koji sadrži popis svih jela i pića naručenih tom prilikom, uz pojedinačne cijene artikala te ukupni iznos narudžbe. Također, na sučelju se nalazi i status narudžbe, koji može biti u stanju „Zaprimljeno“, to je prvo stanje narudžbe što znači da se narudžba tek kreirala i prikazana je osoblju. Zatim, sljedeće stanje koje dolazi po redu je „U tijeku“, što znači da se narudžba trenutno priprema. Narudžba je najčešće u statusu „U tijeku“ na dan rezervacije, ali sve ovisi o trajanju pripreme narudžbe. Zadnje stanje u kojemu narudžba može biti je stanje „Gotovo“ što znači da je narudžba završena i ponuđena, odnosno konzumirana.



Slika 12. Povijest narudžbi i moje rezervacije (Izvor: samostalna izrada 2024.)

5.4.7.2. Moje rezervacije

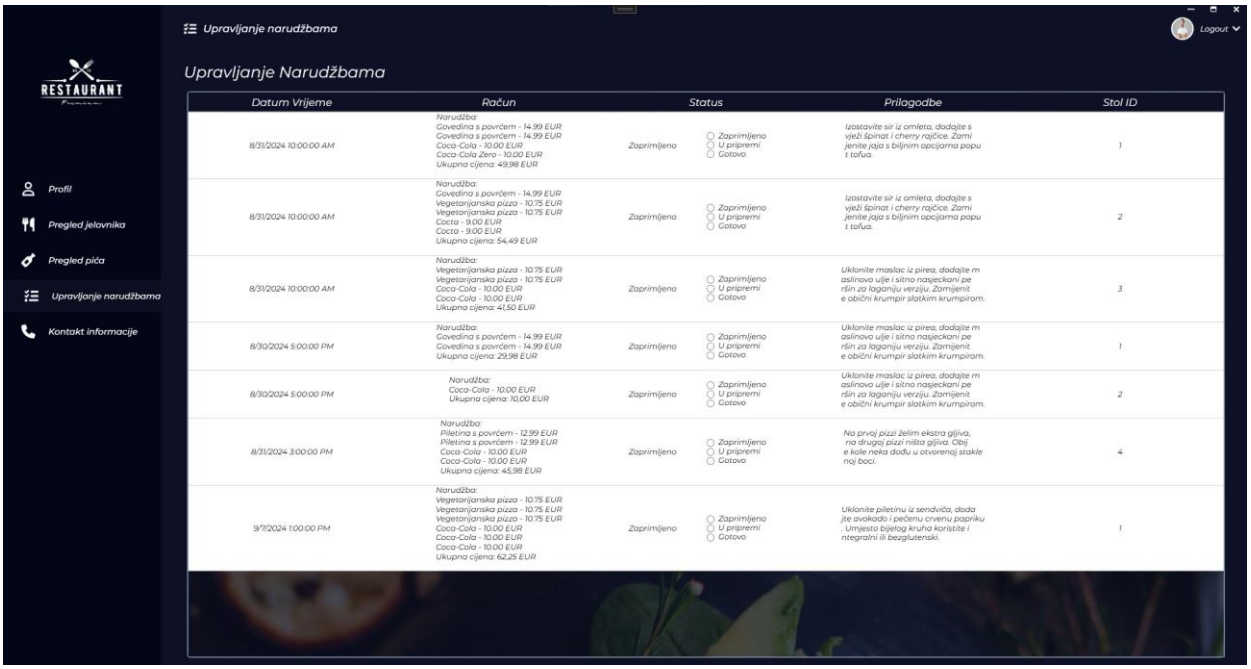
Korisnička kontrola koja služi za pregled aktualnih rezervacija i narudžbi koje korisnik trenutno ima. Prikazuje slične informacije kao i „Povijest narudžbi“, ali se odnosi isključivo na aktivne i nadolazeće rezervacije.

Osim prikaza datuma, vremena, računa i statusa narudžbe, ova kontrola ima dodatnu funkcionalnost, a to je otkazivanje rezervacije. Rezervacije su prikazane u DataGridView-u, a pokraj svake aktivne rezervacije nalazi se gumb „Otkaži“, koji korisniku omogućuje otkazivanje rezervacije. Kada korisnik klikne na ovaj gumb, rezervacija se uklanja s popisa aktualnih rezervacija i oslobađa se stol u restoranu za druge goste.

5.4.8. Upravljanje narudžbama

Upravljanje narudžbama je jedna od funkcionalnosti kojoj može pristupiti samo korisnik s ulogom „Osoblje“. Ova funkcionalnost omogućuje osoblju pregled i upravljanje narudžbama korisnika putem korisničke kontrole nazvane „Upravljanje narudžbama“.

Funkcionalnost je implementirana kao DataGridView, gdje su prikazane sve aktivne narudžbe i rezervacije unutar aplikacije. Ova kontrola centralizira sve bitne informacije na jednom mjestu, omogućujući osoblju restorana jednostavno upravljanje narudžbama i koordinaciju s kuhinjom i posluživanjem.

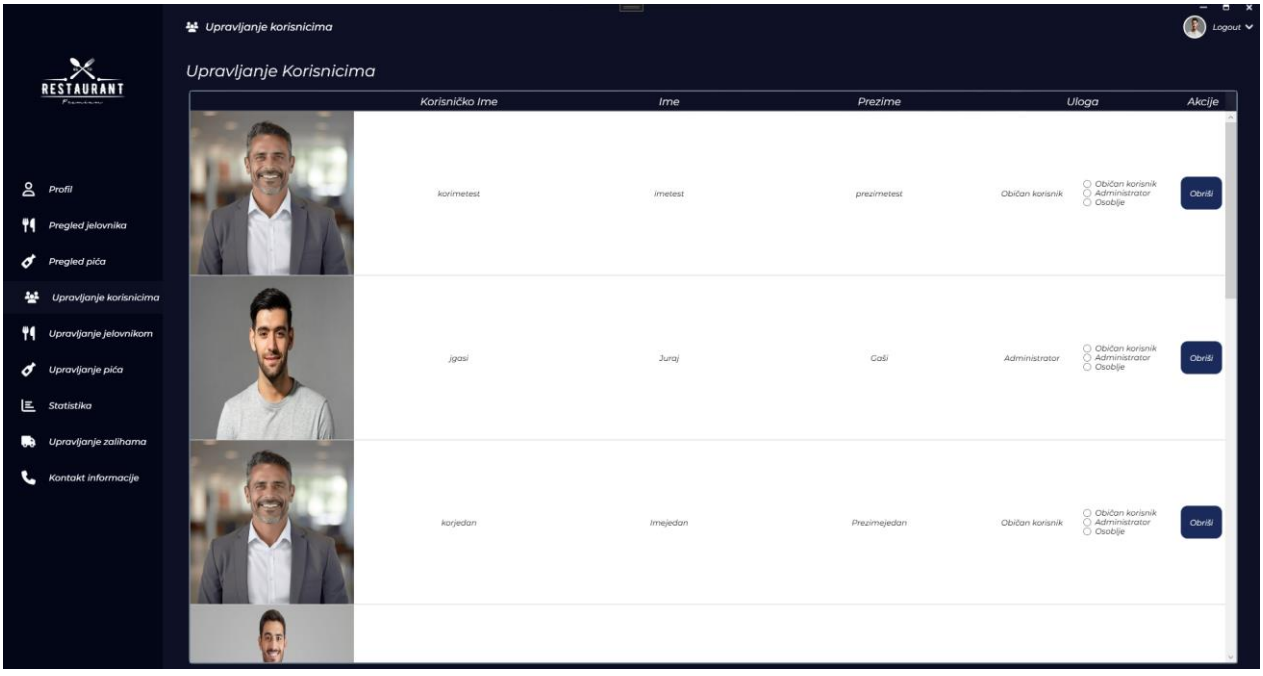


Slika 13. Upravljanje narudžbama (Izvor: samostalna izrada 2024.)

Informacije koje su prikazane za svaku narudžbu uključuju datum i vrijeme rezervacije, što omogućuje bolju organizaciju osoblja i kuhinje kako i kada početi pripremu za svaku rezervaciju. Također, prikazuje račun koji sadrži detaljan popis svih jela i pića uključenih u narudžbu, kao i pojedinačne cijene artikala te ukupnu cijenu narudžbe. Zatim, status narudžbe koji Osoblje može mijenjati u status „Zaprimljeno“ što znači da je narudžba zaprimljena, ali nije još obrađena, „U tijeku“, odnosno da je narudžba trenutno u pripremi te „Gotovo“, što znači da je narudžba završena i spremna za posluživanje/preuzimanje ili je već poslužena/preuzeta. Promjena statusa narudžbe izvodi se pomoću radio buttona koji su smješteni odmah pored prikaza trenutnog statusa. Zatim, prilagodbe, gdje se prikazuju sve specifične želje ili zahtjevi koje je korisnik unio prilikom rezervacije, što omogućuje kuhinji da precizno prilagodi svaku narudžbu. Na kraju, prikazuje se stol koji je rezerviran za korisnika.

5.4.9. Upravljanje korisnicima

Jedna od kontrola kojoj samo administrator ima pristup je „Upravljanje korisnicima“. Ova funkcionalnost omogućava administratoru da upravlja svim korisnicima aplikacije na jednom mjestu, što uključuje pregled njihovih podataka, promjenu uloge i brisanje korisnika iz sustava.



Slika 14. Upravljanje korisnicima (Izvor: samostalna izrada 2024.)

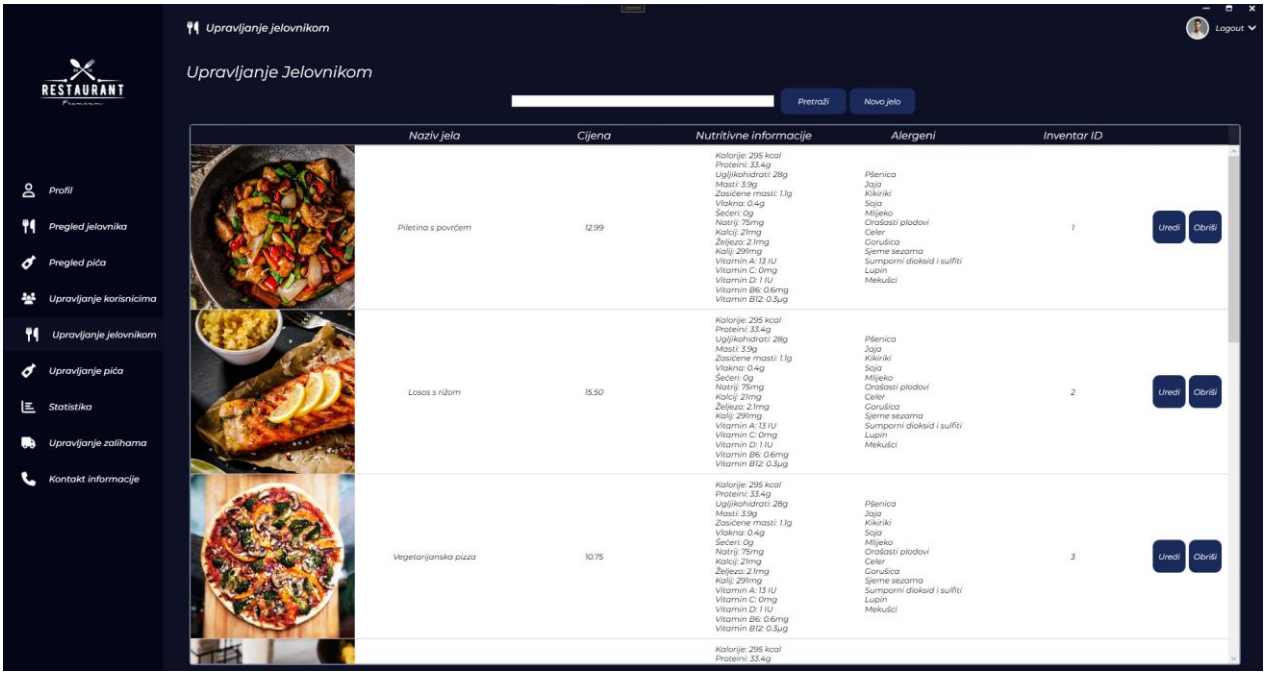
Na ovoj korisničkoj kontroli, svi korisnici aplikacije prikazani su u DataGridView komponenti koja omogućuje pregled i interakciju s podacima u tabelarnom formatu. Svaki red u DataGridView-u sadrži sliku korisnika, korisničko ime, ime i prezime te ulogu korisnika. U redu, također, postoje dva tipa gumba koji omogućuju drukčije akcije.

Jedna od tih akcija je promjena uloge korisnika. Administrator može promijeniti ulogu bilo kojeg korisnika koristeći radio buttons. Svaki korisnik može biti prebačen u jednu od tri dostupne uloge. Uloge koje su dostupne su običan korisnik koji je zapravo početna, odnosno zadana vrijednost, zatim administrator te osoblje. Administrator ima pristup naprednim funkcionalnostima koje se većinom fokusiraju na upravljanje nekih pozadinskih radnji. Osoblje se fokusira na funkcionalnosti vezanim za upravljanje narudžbama i rezervacijama.

Također, jedna od tih akcija je brisanje korisnika. Na kraju svakog reda, nalazi se „Obriši“ gumb. Pritiskom na ovaj gumb, administrator pokreće proces brisanja korisnika iz sustava. Prije nego što se korisnik obriše, prikazuje se dodatna potvrda kako bi se spriječile nenamjerne radnje. Nakon potvrde, korisnik se trajno uklanja iz baze podataka kao i njegove narudžbe i rezervacije vezane za njega te korisnik više nema pristup aplikaciji.

5.4.10. Upravljanje jelovnikom i Upravljanje pića

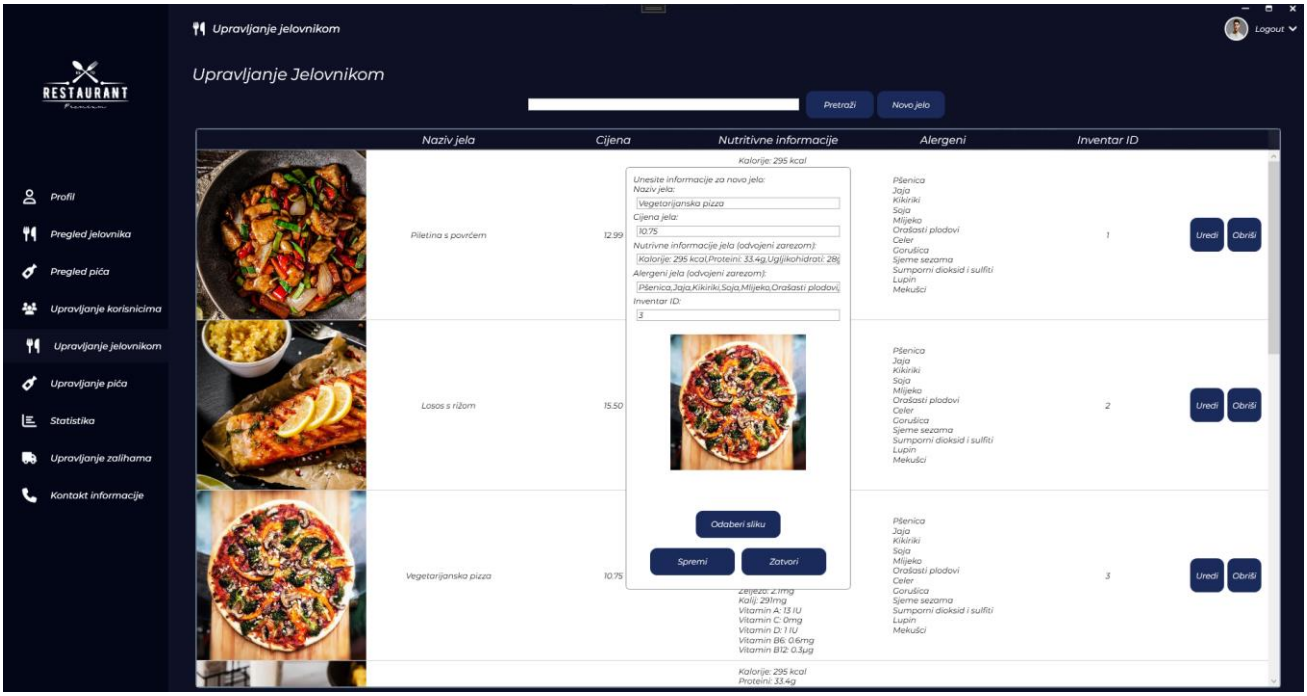
Na stranici za upravljanje jelovnikom i upravljanje pića unutar aplikacije, administrator ima mogućnost pregleda, uređivanja, brisanja te dodavanja novih jela i pića u bazu podataka. Ove dvije kontrole su gotovo identične u funkcionalnosti, ali se odnose na različite tipove podataka, jedna za jela, a druga za pića.



Slika 15. Upravljanje jelovnikom i pićem (Izvor: samostalna izrada 2024.)

Informacije o svakom jelu ili piću prikazuju se u DataGridView-u u tabelarnom formatu. Za svaki red, odnosno za svaki artikl, informacije koje su prikazane su slika jela ili pića, naziv, cijena, nutritivne informacije, alergeni te ID inventara. Također u svakome redu postoje dvije akcije koje se mogu izvršiti na odabranom artiklu, a to su akcija Uredi i Obriši.

Pritiskom na gumb Uredi, otvara se popup prozor [19] s trenutnim podacima o artiklu. Ovdje korisnik može promijeniti informacije, a regex-i osiguravaju da svi unosi budu validni. Nakon izmjene, podaci se ažuriraju u bazi podataka.



Slika 16. Kreiranje novog jela/pića i uređivanje (Izvor: samostalna izrada 2024.)

Pritiskom na gumb Obriši, korisniku se prikazuje poruka „Jeste li sigurni?“. Ako potvrdi, artikl se trajno briše iz baze podataka, uključujući sve veze, kao što su narudžbe i rezervacije vezane uz taj artikl.

Dodatne opcije koje se nalaze iznad DataGridView-a su pretraživanje te kreiranje novog artikla. Pretraživanje nam omogućuje pretragu artikala po imenu, olakšavajući pronalaženje specifičnih artikala u bazi.

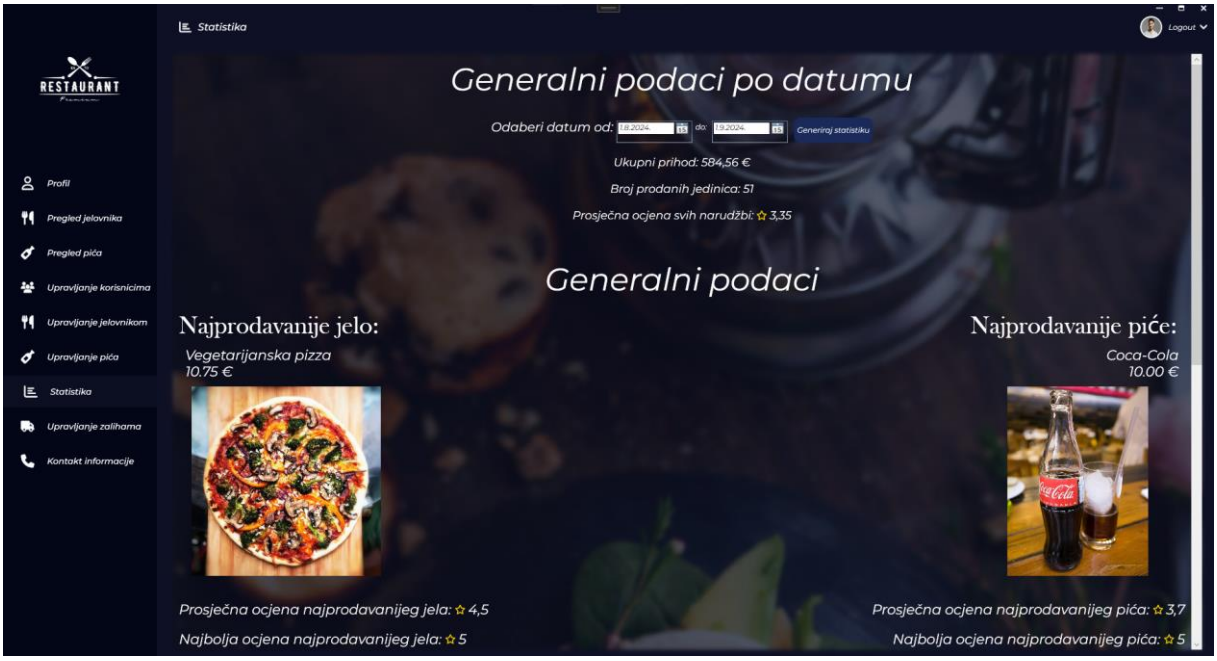
Pritiskom na gumb „Novo jelo“ ili „Novo piće“, otvara se popup prozor [19] s praznim poljima za unos informacija o novom artiklu. Nakon popunjavanja podataka, artikl se dodaje u bazu podataka.

Svako jelo ili piće mora biti povezano s inventarom kroz ID inventara, što predstavlja vezu 1 naprema 1. Kada se kreira novo jelo ili piće, potrebno je prvo kreirati odgovarajući unos

u inventaru, jer artikl ne može postojati bez povezanog inventara. Ovo osigurava da podaci o inventaru i artiklima ostanu dosljedni i sinkronizirani unutar sustava. Također, onemogućava dodavanje artikala prije nego su naručeni odnosno postoji zapis u inventaru te tim putem zabranjuje narudžbu tog artikla prije nego je uopće dostupan.

5.4.11. Statistika

Statistika je administrativna kontrola unutar aplikacije koja omogućuje administratorima pristup ključnim poslovnim podacima i analitici. Ova kontrola pruža pregled poslovnih performansi kroz različite aspekte, kao što su financijski rezultati, popularnost artikala te opća aktivnost korisnika u aplikaciji.

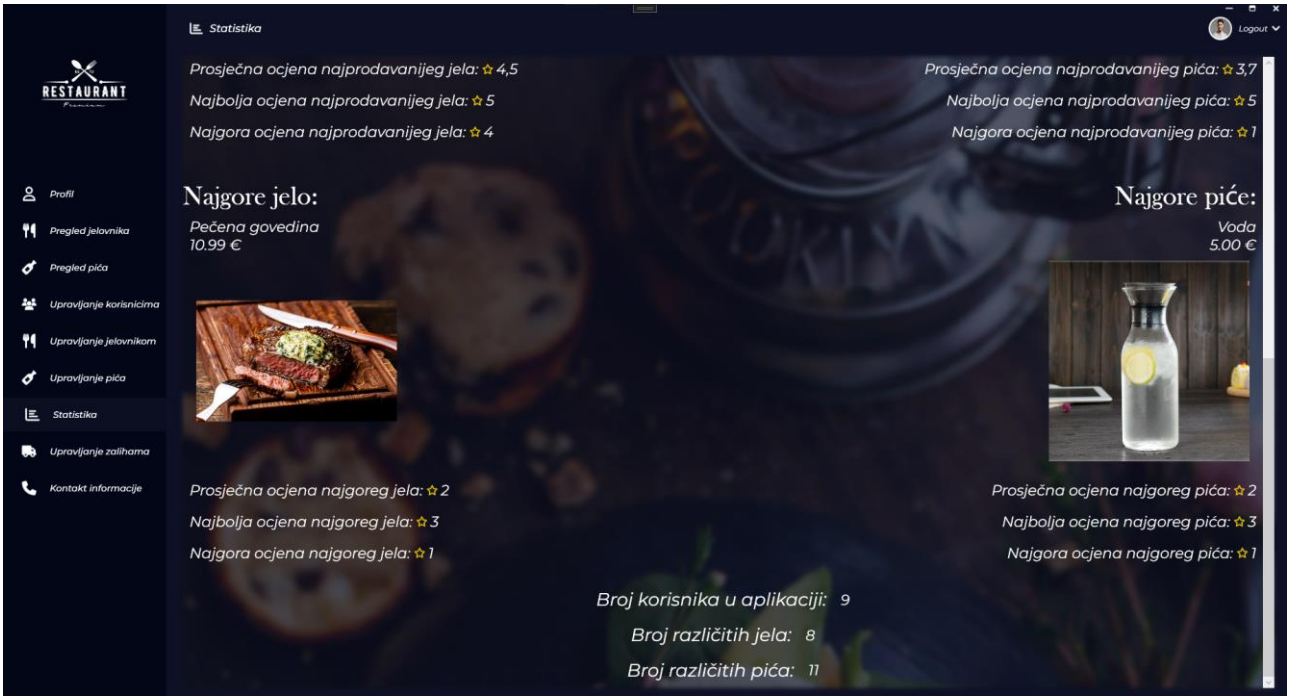


Slika 17. Statistika 1. dio (Izvor: samostalna izrada 2024.)

Postoji više sekcija stranice, a jedna od njih je generalni podaci po datumu. Ova sekcija omogućava administratorima da odaberu specifičan datum ili razdoblje i generiraju statističke podatke za to razdoblje. Nakon odabira datuma i pritiska na gumb „Generiraj statistiku“, prikazuju se informacije kao što su ukupni prihod za to razdoblje, broj prodanih jedinica te prosječna ocjena svih narudžbi.

Još jedna sekcija je „Generalni podaci“. Ova sekcija pruža pregled najvažnijih podataka vezanih uz jela i pića, kao što su najprodavanije jelo, gdje se prikazuju ime, slika i cijena najprodavanijeg jela. Također, prikazana je i prosječna ocjena tog jela, kao i njegova najbolja

i najgora ocjena. Iste takve informacije prikazuju se i za Najprodavanije piće, te najgore jelo i najgore piće. Dodatne informacije koje se prikazuju na dnu kontrole su broj korisnika u aplikaciji, broj različitih jela te broj različitih pića.

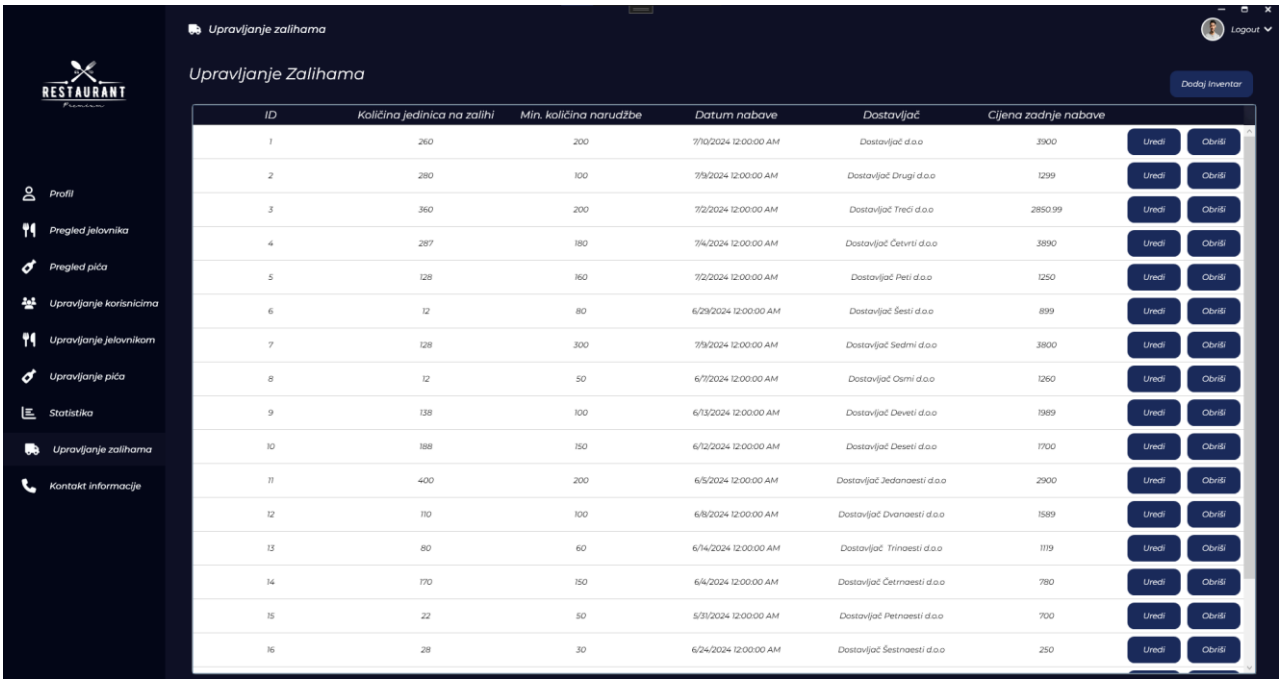


Slika 18. Statistika 2. dio (Izvor: samostalna izrada 2024.)

Kontrola Statistika omogućuje administratorima uvid u ključne poslovne podatke i trendove, olakšavajući donošenje oduka temeljenih na analizi podataka. Ova funkcionalnost pomaže u razumijevanju ponašanja korisnika, popularnosti artikala i financijskog učinka, pružajući tako neophodan alat za optimizaciju poslovanja.

5.4.12. Upravljanje zalihama

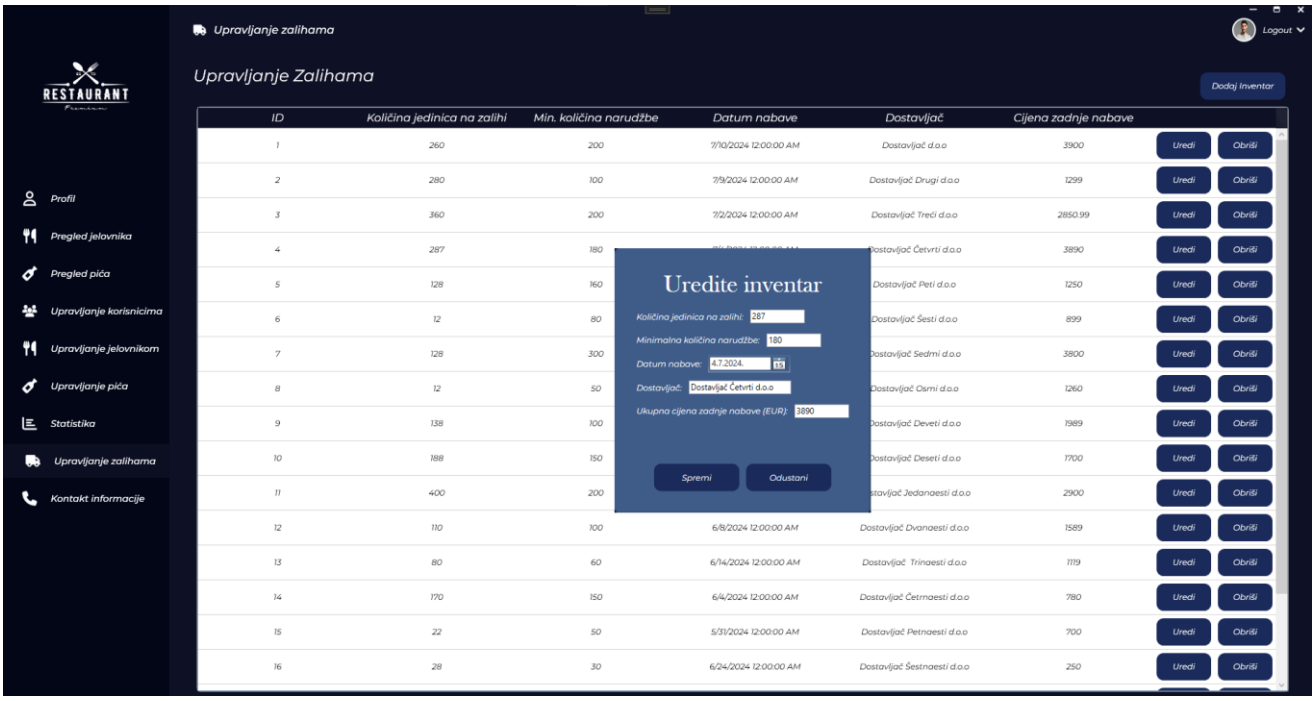
Upravljanje zalihama je ključna administrativna funkcionalnost unutar aplikacije, koja omogućuje pregled, dodavanje, uređivanje i brisanje inventara koji su povezani s određenim jelima ili pićima. Svaki inventar predstavlja specifičan skup zaliha povezan s jednim jelom ili jednim pićem, a njegova funkcionalnost osigurava da zalihe budu ažurne i pravilno održavane.



Slika 19. Upravljanje zalihama (Izvor: samostalna izrada 2024.)

Podaci o inventaru prikazani su u DataGridView-u (DGV), koji omogućuje lako pretraživanje i manipulaciju zalihama. Svaki red u DGV-u prikazuje informacije kao što su ID inventara, količina jedinica na zalihi, minimalna količina narudžbe, datum nabave, dostavljač nabave, odnosno datum kada su zalihe posljednji put nadopunjene te cijena zadnje nabave, odnosno trošak posljednje isporuke za povezano jelo ili piće.

Svaki red u DGV-u nudi dvije akcije, a to su gumb „Uredi“ i „Obriši. Kada se pritisne na gumb „Uredi“, otvara se popup prozor [19] koji prikazuje trenutne podatke o odabranom inventaru. Unutar ovog prozora korisnik može izmijeniti podatke, a izmjene su podložne validaciji putem regexa. Pritiskom na gumb „Obriši“ omogućuje se brisanje inventara iz sustava, ali tek nakon što se korisnika pita je li siguran u tu akciju. Važno je napomenuti da inventar nije moguće obrisati dok je povezan s nekim jelom ili pićem. Prvo je potrebno prebaciti artikl na drugi inventar ili obrisati artikl iz sustava.



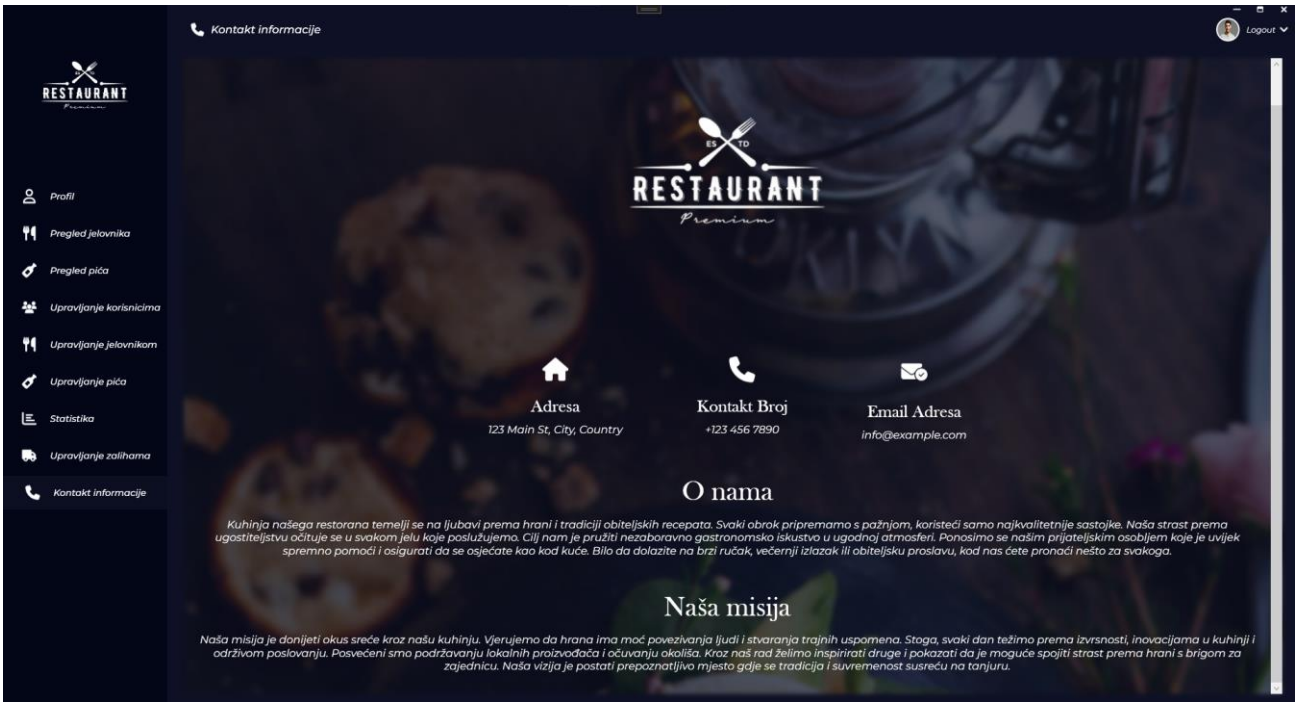
Slika 20. Kreiranje novog inventara i uređivanje (Izvor: samostalna izrada 2024.)

Iznad DGV-a nalazi se gumb „Dodaj inventar“ koji otvara popup prozor [19] za kreiranje novog inventara. U ovom prozoru korisnik unosi sve potrebne podatke kao što su količina jedinica na zalihi, minimalna količina narudžbe, dostavljač, cijena zadnje nabave i datum nabave. Nakon unosa, novi inventar se sprema u bazu podataka i postaje dostupan za povezivanje s jelima ili pićima.

Ova kontrola također, podržava scrollanje, omogućujući jednostavan pregled i upravljanje većim količinama podataka bez gubitka pregleda nad svim stavkama inventara.

5.4.13. Kontakt informacije

Kontakt informacije je jednostavna, ali važna kontrola unutar aplikacije koja služi za prikaz osnovnih podataka o restoranu. Ova kontrola omogućuje korisnicima da brzo i jednostavno pronađu sve relevantne kontakt informacije i saznaju više o samom restoranu.



Slika 21. Kontakt informacije (Izvor: samostalna izrada 2024.)

Informacije koje kontrola sadrži su logo restorana koji vizualno prikazuje brend restorana, adresa fizičke lokacije restorana kako bi korisnici znali kako doći do restorana, kontakt broj gdje je naveden broj telefona na koji korisnici mogu nazvati kako bi dobili više informacija, rezervirali stol ili postavili bilo kakva pitanja vezana uz usluge restorana te email adresa.

Postoji i sekcija „O nama“ koja sadrži kratak tekst o povijesti, vrijednostima ili posebnostima restorana. Cilj ove sekcije je pružiti korisnicima uvid u pozadinu i filozofiju poslovanja restorana, stvarajući povezanost i razumijevanje između restorana i korisnika.

Također, postoji sekcija „Naša misija“ u kojoj se opisuje misija i ciljevi restorana. Ovdje se može pronaći informacija o tome kako restoran pristupa kvaliteti usluge, zadovoljenju korisničkih potreba i svojim planovima za budućnost. To pomaže korisnicima razumjeti viziju restorana i ono što ga izdvaja od konkurencije.

Zaključak

U ovom završnom radu razvijena je desktop aplikacija namijenjena restoranima, koja omogućuje cjelovito upravljanje poslovnim procesima poput korisnika, rezervacija, narudžbi, jelovnika, zaliha i statistika. Aplikacija je strukturirana na temelju troslojne arhitekture koja uključuje prezentacijski sloj, poslovnu logiku i sloj za pristup podacima. Ovaj pristup omogućava veću modularnost i fleksibilnost prilikom razvoja i daljnje nadogradnje aplikacije, čime je omogućeno lako održavanje i proširenje funkcionalnosti.

Cilj aplikacije bio je pojednostaviti i optimizirati svakodnevne operacije u restoranima kroz automatizaciju ključnih zadataka. Automatizirano dodjeljivanje slobodnih stolova tijekom rezervacija, upravljanje narudžbama u stvarnom vremenu, te praćenje zaliha samo su neke od funkcionalnosti koje smanjuju mogućnost ljudskih pogrešaka i povećavaju učinkovitost poslovanja. Osim toga, aplikacija pruža personalizirano iskustvo za tri različite vrste korisnika, a to su administrator, osoblje i običan korisnik. Administrator ima potpunu kontrolu nad sustavom, dok osoblje upravlja narudžbama i korisnici vrše rezervacije, narudžbe i recenzije.

Tijekom razvoja ove aplikacije, potvrđena je pretpostavka da digitalizacija poslovnih procesa može značajno unaprijediti organizaciju rada i poboljšati korisničko iskustvo. Automatizacijom ključnih funkcija smanjene su mogućnosti grešaka, povećana je brzina obrade narudžbi, a korisnici su dobili brži i jednostavniji pristup uslugama.

Korišteni su suvremeni alati i tehnologije koji su omogućili implementaciju svih funkcionalnosti na efikasan i skalabilan način. Posebna pažnja posvećena je sigurnosti podataka, provjeri korisničkih unosa pomoću regularnih izraza te spremanje korisničkih podataka uz povećanu sigurnost.

Razvijena aplikacija predstavlja sveobuhvatan alat za upravljanje restoranom koji ne samo da optimizira radne procese, nego i poboljšava kvalitetu usluge, olakšava upravljanje resursima te pruža mogućnost praćenja i analize poslovnih rezultata u realnom vremenu. Korištenjem ove aplikacije restorani mogu osigurati visoku razinu organizacije, učinkovitosti i zadovoljstva svojih korisnika.

Poveznica na GitHub repozitorij

https://github.com/jgasi/RestaurantManagementSystem

Popis literature

[1] Microsoft (2024). C#. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/dotnet/csharp/ [Pristupljeno: 21-kol-2024].

[2] Microsoft (2024). Entity Framework. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/ef/core/ [Pristupljeno: 21-kol-2024].

[3] Microsoft (2024). WPF. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0 [Pristupljeno: 21-kol-2024].

[4] Microsoft (2024). XAML. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/xaml/?view=netdesktop-8.0 [Pristupljeno: 21-kol-2024].

[5] TechTarget (2024). SQL Server. [Na internetu]. Dostupno na: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server [Pristupljeno: 21-kol-2024].

[6] ZDNet (2020). Microsoft Azure. [Na internetu]. Dostupno na: https://www.zdnet.com/article/microsoft-azure-everything-you-need-to-know/ [Pristupljeno: 21-kol-2024].

[7] MySQL (2024). MySQL Workbench. [Na internetu]. Dostupno na: https://dev.mysql.com/downloads/workbench/ [Pristupljeno: 21-kol-2024].

[8] Microsoft (2024). Visual Studio. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/visualstudio/ide/?view=vs-2022 [Pristupljeno: 21-kol-2024].

[9] GitHub (2024). GitHub Documentation. [Na internetu]. Dostupno na: https://docs.github.com/en [Pristupljeno: 21-kol-2024].

[10] Microsoft (2024). Microsoft Word. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/office/client-developer/word/word-home [Pristupljeno: 21-kol-2024].

[11] GeeksforGeeks (2024). Arhitektura sustava/aplikacije (troslojna arhitektura). [Na internetu]. Dostupno na: https://www.geeksforgeeks.org/three-tier-client-server-architecture-in-distributed-system/ [Pristupljeno: 21-kol-2024].

[12] GeeksforGeeks (2024). ERA model. [Na internetu]. Dostupno na: https://www.geeksforgeeks.org/introduction-of-er-model/ [Pristupljeno: 21-kol-2024].

[13] Mijač, M. (2023). Implementacija slojevite arhitekture u .NET-u. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=igd9-pOOJ1o [Pristupljeno: 25-tra-2024].

[14] Microsoft (2024). In memory cache ASP.NET. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/aspnet/core/performance/caching/memory?view=aspnetcore-8.0 [Pristupljeno: 10-svi-2024].

[15] RexEgg. Regex guide. [Na internetu]. Dostupno na: https://www.rexegg.com/regex-quickstart.php [Pristupljeno: 05-lip-2024].

[16] ByteHide (2023). Regex u C#. [Na internetu]. Dostupno na: https://www.bytehide.com/blog/regex-csharp [Pristupljeno: 05-lip-2024].

[17] YouTube (2022). WPF & MVVM/Modern Main UI Design Part 1. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=kxhvwGEqvcs [Pristupljeno: 28-tra-2024].

[18] YouTube (2022). WPF & MVVM/Modern Main UI Design Part 2. [Na internetu]. Dostupno na: https://www.youtube.com/watch?v=76JLBZJR5gE [Pristupljeno: 28-tra-2024].

[19] Microsoft (2024). Popup. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/controls/popup-overview?view=netframeworkdesktop-4.8 [Pristupljeno: 02-lip-2024].

[20] Microsoft (2024). Observable Collection. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/dotnet/api/system.collections.objectmodel.observablecollection-1?view=net-8.0 [Pristupljeno: 12-svi-2024].

[21] Microsoft (2024). BitmapImage Class. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/dotnet/api/system.windows.media.imaging.bitmapimage?view=windowsdesktop-8.0 [Pristupljeno: 20-svi-2024].

[22] Microsoft (2024). Convert Image in Byte Array with the Same Pixel Format. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/answers/questions/1164941/convert-image-in-byte-array-with-the-same-pixelfor [Pristupljeno: 20-svi-2024].

Popis slika