Implementacija baze podataka za potrebe ljekarne

Josipović, Dominik

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Josipović

IMPLEMENTACIJA BAZE PODATAKA ZA POTREBE LJEKARNE

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Josipović

Matični broj: 49301

Studij: Informacijski i poslovni sustavi

IMPLEMENTACIJA BAZE PODATAKA ZA POTREBE LJEKARNE

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Kornelije Rabuzin

Varaždin, rujan 2024.

Dominik Josipović

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

U radu se teorijski obrađuje pojam relacijskih baza podataka i njihovih najvažnijih koncepata pokazanih na primjerima. Spomenute su i ostale vrste baza podataka kao što su baze podataka dokumenata, grafovske baze podataka i XML baze podataka. Teorijski su obrađeni načini spremanja podataka zapisanih u NoSQL bazama u relacijske baze podataka. Praktični dio rada obuhvaća modeliranje baze podataka korištenjem VATEK metode, izrade dijagrama entiteti-veze, ERA modela, rječnika podataka i implementaciju aplikacije za upravljanje bazom u troslojnoj arhitekturi gdje je objašnjen svaki sloj aplikacije i kako se koristi grafičko sučelje aplikacije. Prikazano je postavljanje SQL servera, modeliranje baze podataka u SQL Server Management Studiu i izrada aplikacije u .NET okviru pomoću biblioteke Windows Forms i Entity Frameworka.

Ključne riječi: relacijske baze podataka, modeliranje, SQL Server, Windows Forms, .NET, Entity Framework, troslojna arhitektura

Sadržaj

1. Uvod 1
2. Metode i tehnike rada 2
3. Relacijske baze podataka 4
   3.1. Usporedba s grafovskim bazama podataka 7
   3.2. Usporedba s bazama podataka dokumenata 9
   3.3. Usporedba s XML bazama podataka 11
   3.4. Prednosti i nedostaci relacijskih baza podataka 12
4. Modeliranje baze podataka ljekarne 14
   4.1. Opis poslovnih pravila ljekarne 15
   4.2. Dijagram entiteti-veze i relacijska shema 15
5. ERA model baze podataka 18
6. SQL Server i SQL Server Management Studio 19
   6.1. Postavljanje SQL Servera 19
   6.2. Implementacija baze podataka 23
      6.2.1. Kreiranje tablica i postavljanje primarnih ključeva 24
      6.2.2. Postavljanje ograničenja i UNIQUE indeksa 25
      6.2.3. Definiranje veza i vanjskih ključeva 27
7. Aplikacija 29
   7.1. Arhitektura aplikacije 29
   7.2. .NET platforma i Windows Forms 30
   7.3. Entity Framework i spajanje na bazu podataka 31
   7.4. Implementacija Repository klasa u sloju pristupa podacima 36
      7.4.1. FarmaceutRepository 37
      7.4.2. JedinicaMjereRepository 38
      7.4.3. ArtiklRepository 38
      7.4.4. RacunRepository 39
      7.4.5. NarudzbaRepository 40
   7.5. Implementacija Service klasa u sloju poslovne logike 41
      7.5.1. Iznimke 42
      7.5.2. FarmaceutServices 42
      7.5.3. JedinicaMjereServices 43
      7.5.4. ArtiklServices 44
      7.5.5. RacunServices 45
      7.5.6. NarudzbaServices 46
   7.6. Prezentacijski sloj i grafičko sučelje 47
      7.6.1. Login forma 47
      7.6.2. Korisnički račun 49

7.6.3. Artikli 52
7.6.4. Računi 54
8. Zaključak 58
Popis literature 59
Popis slika 60
Popis tablica 62
Prilozi 63
1. Opis relacija 63
1.1. Relacija Jedinica mjere 63
1.2. Relacija Artikl 63
1.3. Relacija Pacijent 64
1.4. Relacija Ustanova 64
1.5. Relacija Liječnik 65
1.6. Relacija Farmaceut 65
1.7. Relacija Recept 66
1.8. Relacija Dobavljač 66
1.9. Relacija Status narudžbe 66
1.10. Relacija Narudžba 67
1.11. Relacija Primka 67
1.12. Relacija Račun 68
1.13. Relacija Stavke 68
2. Opis veza 69
2.1. Veza Jedinica mjere – Artikl 69
2.2. Veza Pacijent – Recept 69
2.3. Veza Ustanova – Liječnik 70
2.4. Veza Recept – Liječnik 70
2.5. Veza Status narudžbe – Narudžba 70
2.6. Veza Narudžba – Dobavljač 70
2.7. Veza Narudžba – Farmaceut 71
2.8. Veza Narudžba – Primka 71
2.9. Veza Dobavljač – Primka 71
2.10. Veza Primka – Farmaceut 72
2.11. Veza Farmaceut – Račun 72
2.12. Veza Artikl – Stavke računa 72
2.13. Veza Stavke računa – Račun 72
3. SQL skripta baze podataka 73
4. Slika zaslona aplikacije recepata 83
5. Slika zaslona aplikacije artikala 83
6. Slika zaslona aplikacije dobavljač 84

7. Slika zaslona aplikacije primke 85
8. Slika zaslona aplikacije narudžbe 85

1. Uvod

Sve većim korakom u napretku tehnologija većina stvari je digitalizirana i virtualna. Različite domene poslovanja zahtijevaju jedinstveni pristup svakom pokušaju automatizaciji podataka koji se pohranjuju u velikim količinama. Potreba za prilagodbom svakoj od domena nastale su različite vrste baza podataka među kojima se nalaze i relacijske baze podataka. Kako bi se jasnije predočila razlika teorijski je obrađena razlika između relacijskih i ostalih vrsta baza podataka iz koje se mogu vidjeti prednosti i nedostaci korištenja svake od vrste baza.

Svaka baza podataka da bi bila od koristi mora biti pravilno implementirana. Pravilno implementirati bazu moguće je detaljnim proučavanjem poslovne domene i poznavanjem metoda koje olakšavaju prepoznavanje poslovnog okruženja i podataka koji kruže u poslovanju. U ovom radu obrađeno je implementiranje baze podataka za poslovanje ljekarne uz pomoć korištenjem metoda koje olakšava prepoznavanje poslovnih objekata i povezivanje poslovanja u cjelinu. Cilj izrade baze podataka je pružiti pomoć korisnicima i radnicima ljekarne na način da se poslovni procesi odvijaju brzo i efikasno. Kako bi se omogućilo korisnicima korištenje i upravljanje bazom podataka potrebno je pružiti mogućnost svakom korisniku razumijevanje kako upravljati podacima, odnosno napraviti grafičko sučelje koje će svaki korisnik moći razumjeti koristiti i biti u mogućnosti brzo shvatiti. Motivacija za izradu ovog rada je prikazati tehnike i metode izrade baze podataka za domenu ljekarne proučavanjem poslovnih dokumenata i pravila te izradi korisničkog sučelja koji će omogućiti svim korisnicima lako upravljanje podacima.

2. Metode i tehnike rada

Za potrebe teorijskog dijela ovog završnog rada korišteni su vanjski izvori koji obuhvaćaju stručne knjige i članke s interneta. U praktičnom dijelu rada korištene su metode modeliranja podataka prema teoriji na domeni ljekarne. Za detaljniju spoznaju poslovanja ljekarne poslovna pravila kao i dokumenti analizirali su se u pravoj ljekarni što je uključivalo pronalazak stručne osobe i dogovoreni posjet ljekarni. Podaci i dokumenti o ljekarni prikupljeni su od strane zaposlenika ljekarne.

Modeliranje baze podataka obuhvaćalo je istraživanje tehnika i načina dokumentiranja poslovnih dokumenata te kako prikazati veze i entitete na grafičkom prikazu. Za izradu ERA modela i dijagrama entiteti-veze korišten je online alat Diagrams.net. Implementacija same baze podataka zahtijevala je instalaciju i konfiguraciju lokalnog SQL servera i programa SQL Server Management Studio preko kojega je napravljena baza podataka i korisnik u bazi preko kojeg se pristupalo bazi iz vanjskih aplikacija.

Izrada aplikacije odvijala se u alatu Visual Studio 2022 u .NET Frameworku. Aplikacija je pisana u C# programskom jeziku, a grafičko sučelje aplikacije napravljen je u Windows Formsu. Za spajanje na lokalni SQL Server potrebno je bilo instalirati Entity Framework koji je služio za generiranje klasa entiteta baze podataka kao i cijeli kontekst baze podataka preko kojeg se moglo povezati na bazu.

Na početku rada teorijski je obrađen pojam relacijskih baza podataka. Spomenuto je što čine relacijske baze podataka, od kojih elemenata se sastoje gdje je bio obuhvaćen model podataka, struktura relacijskih baza podataka, primarni i vanjski ključevi, veze, relacijska shema i usporedba relacijskih baza podataka s grafovskim bazama podataka, XML bazama podataka i bazama podataka dokumenata. Na kraju poglavlja navedene su ukratko prednosti i nedostaci relacijskih baza podataka.

U poglavlju modeliranja baza podataka spomenuta je VATEK metoda koja služi za prepoznavanje entiteta i njihovih atributa te povezivanje podataka u cjelinu. Napravljena je relacijska shema baze podataka, dijagram entiteti-veze i rječnik podataka.

Poglavlje ERA model baze podataka obuhvaća sliku ERA modela napravljenu prema relacijskoj shemi. Opisane su relacije i atributi, tipovi vrijednosti kao i veze između relacija koje su popraćene kratkim objašnjenjima.

U sljedećem poglavlju opisan je SQL Server i alat SQL Server Management Studio gdje je objašnjeno kako se provelo modeliranje baze podataka i kako definirati veze i vanjske ključeve u alatu.

U zadnjem poglavlju opisana je aplikacija za upravljanje bazom gdje je objašnjena troslojna arhitektura koju aplikacija koristi, .NET i Windows Forms. Objašnjen je Entity Framework i prikazano kako se pomoću njega može spojiti na bazu podataka u aplikaciji. Posebno je obrađen svaki sloj aplikacije gdje su prikazani programski kodovi na svakom sloju, rukovanje iznimkama i prikaz grafičkog sučelja aplikacije.

3. Relacijske baze podataka

„Baza podataka je kolekcija podataka, ograničenja i operacija koja reprezentira neke aspekte realnog svijeta.“ [1], [2] Relacijske baze podataka ostvaruju se primjernom relacijskog modela podataka koji je osnovni element sustava za upravljanje bazom podataka.

Kako je u samom nazivu spomenuto relacijske baze podataka sastoje se od skupa relacija koje predstavljaju pravokutne tablice koje opisuju određene entitete. Svaka relacija u bazi podataka ima svoj naziv koji je razlikuje od ostalih te sadrži skup atributa pri čemu svaki atribut predstavlja stupac tablice i određen je nazivom po čemu se razlikuje od ostalih atributa na istoj relaciji. Za svaki atribut definira se tip podataka koji je pohranjen u stupcu tablice čime je određeno koji podaci mogu biti zapisani pod svakim atributom. Ime tablice i popis njenih atributa čine relacijsku shemu. [2], [3]

Tablica 1: Relacija osoba

OSOBA

OIB	IME	PREZIME	GRAD
23925830841	Marko	Ivić	Varaždin
95392854034	Ivo	Jurić	Zagreb
40649293407	Pero	Markić	Zagreb
84856372138	Jura	Perić	Koprivnica

Relacija „osoba“ (Tablica 1.) sastoji se od atributa OIB, ime, prezime, grad i svaki redak u tablici prikazuje zapis (slog) jednog entiteta koji u ovom slučaju predstavlja osobu. Redak tablice naziva se n-torka koja se sastoji od svih atributa tablice što znači da se svaki zapis u tablici mora sastojati od svih atributa, tako na primjer imamo prvi slog tablice u kojoj se nalazi osoba Marko Ivić i možemo iščitati sve vrijednosti tog entiteta. Poredak slogova i atributa u tablici je nebitan te ako bi u bazi postojala druga tablica koja se sastoji od istih atributa kao što je ovdje tablica „osoba“ mogla bi se smatrati istom tablicom. [3] Isto bi vrijedilo i za poredak slogova u tablici. Zapis relacijske sheme tablice 1 izgledao bi ovako: OSOBA (OIB, IME, PREZIME, GRAD) gdje atribut OIB čini primarni ključ relacije. Svaki slog tablice mora biti jednoznačno određen primarnim ključem budući da u tablici ne smiju postojati dvije jednake n-torke. Manger navodi definiciju ključa relacije na sljedeći način:

„*Ključ K relacije R je podskup skupa atributa od R sa sljedećim svojstvima:*

1. *Vrijednost atributa iz K jednoznačno određuju n-torku u R. Dakle, ne mogu u R postojati dvije n-torke s istim vrijednostima atributa iz K.*
2. *Ako izbacimo iz K bilo koji atribut, tada se narušava svojstvo 1.*“[3]

Za tablicu osobe primarni ključ je OIB budući da ne mogu postojati dvije osobe s istim OIB-om. Primarni ključ tablice mogu činiti i dva atributa, recimo ime i prezime osobe, no to ne bi mogao biti ključ ove relacije budući da bi se vjerojatno pojavile osobe s istim imenom i prezimenom.

Više relacija u bazi podataka može biti povezano. Relacije se povezuju pomoću primarnih ključeva gdje primarni ključ relacije u nekoj drugoj relaciji predstavlja vanjski ključ.

Na primjeru sljedeće relacijske sheme imamo tri relacije:

NASTAVNIK(ŠIFRA-NAS, IME, PREZIME)

PREDMET(ŠIFRA-PRED, NAZIV, ECTS)

PREDAJE(ŠIFRA-NAS, ŠIFRA-PRED)

Slika 1: Konceptualna shema nastavnik-predmet

Temeljem relacijske sheme može se vidjeti kako nastavnik predaje predmete. Za ovakvu vezu potrebne su tri tablice budući da nastavnik može predavati više predmeta, a predmet može biti predavan od više nastavnika (slika 1). Pojavom veze više na više moramo uvesti treću tablicu pomoću koje ćemo moći spojiti relacije „nastavnik“ i „predmet“. Tablica „predaje“ služi da spojimo n-torke iz relacija „nastavnik“ i „predmet“, a budući da se svaka n-torka u tablici može jednoznačno odrediti pomoći pripadnog primarnog ključa, u tablicu je potrebno zapisati parove primarnih ključeva nastavnika i predmeta (tablica 4). U ovom primjeru možemo vidjeti kako je primarni ključ tablice „predaje“ sastavljen od dva atributa koja su šifra nastavnika i šifra predmeta koji su pak vanjski ključevi tablica „nastavnik“ i „predmet“.

Tablica 2: Relacija nastavnik

NASTAVNIK

ŠIFRA-NAS	IME	PREZIME
1	Marko	Ivić
2	Ivo	Jurić
3	Pero	Markić
4	Jura	Perić

Tablica 3: Relacija predmet

PREDMET

ŠIFRA-PRED	NAZIV	ECTS
1	Baze podataka	5
2	Strukture podataka	6
3	Poslovno odlučivanje	4
4	Organizacija	5

Tablica 4: Relacija predaje

PREDAJE

ŠIFRA-NAS	ŠIFRA-PRED
3	4
2	2
3	2
1	3
4	1

U relaciji „nastavnik“ (Tablica 2) možemo vidjeti kako se nastavnik Pero Markić pod šifrom nastavnika 3, nalazi u relaciji „predaje“ prema njegovo šifri, odnosno primarnom ključu. U relaciji „predaje“ (Tablica 4) nalaze se dva zapisa koja za šifru nastavnika imaju Peru Markića što znači kako Pero Markić predaje dva predmeta, Prema zapisima iz relacija „predaje“ mogu se iščitati vanjski ključevi predmeta koje Pero Markić predaje a to su predmeti koji za primarni ključ imaju vrijednosti 4 i 2. Prema povezanim vanjskim ključevima nastavnika i predmeta možemo točno doći do informacije koji predmet koji nastavnik predaje, tako možemo točno vidjeti da Pero Markić predaje predmete Strukture podataka i Organizaciju čije su šifre 2 i 4.

U relaciji „predaje“ ne bi se smjeli moći nalaziti predmeti ili nastavnici koji ne postoje u relacijama „predmet“ ili „nastavnik“ jer bi inače došlo do pitanja ispravnosti podataka u bazi. Ispravnost i istinitost podataka u bazi spada pod integritet baze podataka koji sprječava unos slučajne pogrješke u relaciju [4]. U prethodnom primjeru do pitanja integriteta dolazi kada bi se u relaciju „predaje“ unio podatak kako nastavnik pod šifrom 7 predaje predmet pod šifrom 3, budući da u relaciji „nastavnik“ ne postoji zapis koji za šifru nastavnika ima vrijednost 7. Konkretan primjer koji je dan u prethodnoj rečenici spada u referencijalni integritet koji govori kako bi se atribut u zapisu u relaciji koji je vanjski ključ, trebao nalaziti u povezanoj relaciji kao primarni [4], [5]. Tkalac [4] definira referencijalni integritet na sljedeći način:

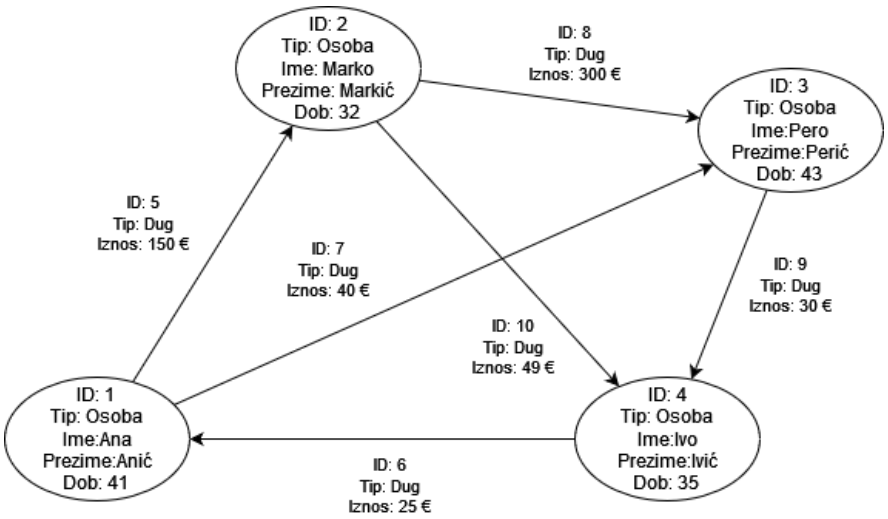
> *„Neka u relaciji r(R) postoji vanjski ključ koji odgovara primarnom ključu u relaciji s(S). Svaka vrijednost vanjskog ključa u r mora biti:*
>
> - *Ili jednaka vrijednosti primarnog ključa u nekoj od n-torki relacije s*
> - *Ili jednaka nul-vrijednosti (…)*
>
> *Relacije r(R) i s (S) ne moraju biti različite relacije.“*

Referencijalni integritet trebao bi biti uvijek osiguran u bazi podataka te se mora paziti prilikom upisa podataka u povezanu relaciju kada se upisuje vrijednost vanjskog ključa u relaciji, ista vrijednost mora postojati u relaciji gdje se nalazi primarni ključ, ista je stvar prilikom ažuriranja podataka. Prilikom ažuriranja podataka u relaciji s primarnim ključem, moraju se kaskadno ažurirati vrijednosti u povezanoj relaciji na vanjskom ključu. Moguća je situacija gdje se želi obrisati zapis u relaciji koji predstavlja vanjski ključ u nekoj drugoj te se prilikom toga moraju definirati akcije koje se trebaju dogoditi. Prilikom brisanja zapisa, ukoliko neka tablica ima vrijednost vanjskog ključa na taj zapis, može se napraviti kaskadno brisanje ili zabraniti brisanje ukoliko postoje povezane relacije, još postoji mogućnost da se vrijednosti vanjskih ključeva postave na NULL ukoliko je to dozvoljeno. [5]

3.1. Usporedba s grafovskim bazama podataka

Osim što se podaci mogu spremati u obliku relacija (tablica), postoje još načina pohrane podataka, a jedan od njih je pohrana u obliku grafova. Grafička struktura podataka sastoji se od čvorova, u koje se spremaju objekti, i veza između čvorova u koje se također mogu spremati podaci. Grafovske baze podataka najčešće se koriste u domenama gdje je bitna veza između objekata. Lena Wiese [5] navodi primjer korištenja grafovske baze podataka na temelju gradova gdje svaki čvor predstavlja grad i informacije o njemu, a veze između gradova predstavljaju koliko je grad udaljen od drugog.

U grafičkoj strukturi podataka mora vrijediti pravilo da između svakog čvora mora biti jedna vrsta veze što bi na konkretnom primjeru značilo da ne smiju postojati dvije veze duga između dvije iste osobe, a isto tako mogu postojati ograničenja da veza koja je tipa „Dug“ može postojati samo između čvorova koji su tipa „Osoba“ [5]. Kada se uvedu sva svojstva čvorova i veza podaci u grafičkom prikazu izgledao bi kao na slici 2.



Slika 2: Prikaz dugova osoba u obliku grafa sa svojstvima (Prema [5])

Prilikom pohrane grafovske baze podataka u relacijske mogu se koristiti dva načina pohrane. Na prvi način kreiraju se relacije za čvorove i veze te posebne relacije u koje se pohranjuju podaci za svaku vrstu čvora i veze. Ovim načinom spremanja grafova u relacijske baze podataka nastaje problem iz razloga što za svaku vrstu čvora ili veze mora postojati posebna tablica koja bi pohranjivala njihova svojstva. Postoji drugi način kako spremiti svojstva čvorova i veza uz uvjet da postoje relacije koje sadrže podatke o vezama i čvorovima. Rješenje je napraviti jednu relaciju „Svojstva“ koja sadrži sve atribute nevezano radi li se o čvoru ili vezi (tablica 5). Relacija „Svojstva“ sadrži atribut „ID_OBJEKTA“ koji je povezan s relacijom „Čvorovi“ ili „Veze“, atribut „Svojstvo“ koje sadrži naziv svojstva čvora ili veze te atributa „Vrijednost“ koji sadrži vrijednost navedenog svojstva. Ovim postupkom dobije se manje relacija, ali se sve informacije o svojstvima spremaju u jednu jedinu relaciju koja može s vremenom imati puno zapisa. Drugi problem je što vrijednost svakog svojstva može biti različitog tipa, a u ovu relaciju sve vrijednosti moraju biti istog tipa kako bi se mogle pohraniti čime ne možemo znati pripada li vrijednost atributa njegovoj domeni [5].

Tablica 5: Relacija Svojstva (Prema [5])

SVOJSTVA

ID_OBJEKTA	SVOJSTVO	VRIJEDNOST
1	Ime	Ana
1	Prezime	Anić
1	Dob	41
2	Ime	Marko
2	Prezime	Markić
2	Dob	32
3	Ime	Pero
3	Prezime	Perić
3	Dob	35
4	Ime	Ivo
4	Prezime	Ivić
4	Dob	43
5	Iznos	150 €
6	Iznos	25 €
7	Iznos	40 €
8	Iznos	300 €
9	Iznos	30 €
10	Iznos	49 €

3.2. Usporedba s bazama podataka dokumenata

Prilikom definiranja svojstva u grafovskim bazama podataka na prethodnom primjeru (slika 2) korišteni su parovi vrijednosti o obliku ključ-vrijednost gdje je na primjer ključ „Dob“, a vrijednost „43“ pri čemu je vrijednost uvijek tekstualnog tipa. Ključ označava identifikator vrijednosti koju pohranjuje i svaki ključ treba biti jedinstven. Operacije koje su podržane su dodavanje, čitanje i brisanje zapisa prema ključu. Prilikom dodavanja zapisa definira se ključ i vrijednost, a prilikom čitanja i brisanja potrebno je znati samo ključ. Pohranjena vrijednost ne mora biti jednostavna, za vrijednost se može staviti JSON objekt, XML zapis ili bilo koji sličan strukturirani zapis. [5]

Baze podataka dokumenata koriste strukturirane zapise koji su formata čitljivog za ljude, a koriste se za pohranu na računalu. Jedan od strukturiranih i standardiziranih zapisa koje koriste baze podataka dokumenata je JSON (eng. JavaScript Object Notation) format. JSON dokument sastoji se od ugniježđenih parova ključa i vrijednosti, započinje vitičastim zagradama ({, }). Ključevi su tekstualni zapisi navedeni pod navodnicima („, “), a vrijednosti ne moraju biti, ovisno koje tipove podataka vrijednost sadržava. Prema [5] za vrijednost se mogu spremiti sljedeći tipovi podataka: broj (decimalni ili cijeli), string (tekstualni tip podataka), boolean (tip podataka koji označava istinitost ili laž – true ili false), polje, JSON objekt ili null. Ključ i pripadajuća vrijednost odvojeni su dvotočkom (:) dok su cijeli zapisi odvojeni zarezom (,) [5].

U nastavku je priložen JSON objekt koji se koristi kao odgovor na dohvaćanje podataka o seriji „Game of Thrones“ servisa TMDB. Zapis JSON objekta je skraćen kako bi se mogli vidjeti primjeri podataka koji su pohranjeni u njemu i preuzet je sa stranice službene dokumentacije TMDB-a [6]:


```
{
  "adult": false,
  "backdrop_path": "/6LWy0jvMpmjoS9fojNgHIKoWL05.jpg",
  "created_by": [
    {
      "id": 9813,
      "credit_id": "5256c8c219c2956ff604858a",
      "name": "David Benioff",
      "gender": 2,
      "profile_path": "/xvNN5huL0X8yJ7h3IZfGG4O2zBD.jpg"
    }
  ],
  "id": 1399,
  "last_episode_to_air": {
    "id": 1551830,
    "name": "The Iron Throne",
    "overview": "In the aftermath of the devastating attack on King's Landing, Daenerys
must face the survivors.",
    "vote_average": 4.809,
    "vote_count": 241,
    "air_date": "2019-05-19",
    "episode_number": 6,
```

```
            "production_code": "806",
            "runtime": 80,
            "season_number": 8,
            "show_id": 1399,
            "still_path": "/zBi2O5EJfgTS6Ae0HdAYLm9o2nf.jpg"
          },
          "name": "Game of Thrones",
          "next_episode_to_air": null,
          "original_language": "en",
          "original_name": "Game of Thrones",
}
```


U bazama podataka dokumenata JSON zapisi imaju svoj ID po kojem se označuju te ne postoje vanjski ključevi koji povezuju ostale entitete, nego su svi entiteti uključeni u objekt. Na primjeru ovog JSON objekta možemo vidjeti da ima svoj ID te njezine kreatore pod ključem „created-by“ koji sprema druge JSON objekte u nizu. Pod ključem „adult“ može se vidjeti primjer boolean vrijednosti koja označuje je li serija namijenjena odrasloj publici. Primjer pohrane JSON objekta unutar ključa je primjer „last_episode_to_air“ koji sadrži objekt zadnje emitirane epizode te ključ „next_episode_to_air“ koji sadrži null vrijednost.

JSON zapisi objekata mogu imati definiranu shemu koja određuje koja svojstva će objekt imati, njegovu strukturu i dozvoljene vrijednosti koje ključevi mogu sadržavati. Lena Wiese [5] daje primjer definiranja JSON sheme:

```
{
        "type": "object",
        "properties": {
                "street": { "type": "string" },
                "number": { "type": "number" },
                "city": { "type": "string" },
                "zip": { "type": "number" }
        },
        "additionalProperties": false,
        "required": ["street", "number", "city"]
}
```

Iz primjera JSON sheme može se vidjeti da definira JSON objekt sa svojstvima street, number, city i zip koji su definirani pod ključem „properties“. Za svako svojstvo unutar vitičastih zagrada definirani su tipovi podataka koji će svojstva sadržavati pod ključem „type“, tako će ključevi „street“, i „city“ podržavati tip podataka string, a ključevi „number“ i „zip“ brojčane tipove podataka. Ključevi „additionalProperties“ i „required“ označuju kako objekt neće moći imati dodatna svojstva i da će obavezno morati biti ispunjeni ključevi „street“, „number“ i „city“.

Primjer baze podataka je MongoDB koji koristi BSON format kao način spremanja podataka. BSON format je binarni JSON koji omogućuje brži način rada s JSON bazama podataka, omogućuje preskakanje ugniježđenih objekata bez da se pročita njihov sadržaj budući da svaki objekt ima definiranu duljinu. Upravljanje podacima odvija se preko metoda kao što su „insert“ za dodavanje zapisa i „find“ za dohvaćanje svih zapisa, a moguće je parametrima namjestiti pretraživanje prema određenim kriterijima. [5]

3.3. Usporedba s XML bazama podataka

XML dokument sastoji se od oznaka koje označavaju dijelove dokumenta. Svaka oznaka smješta se unutar izlomljenih zagrada („<“ i „>“) i ona označava element koji se sastoji od početne oznake i zatvarajuće oznake. Zatvarajuća oznaka sadrži kosu crtu („/“) unutar izlomljenih zagrada. Vrijednosti elemenata se pišu unutar početne i zatvarajuće oznake te se elementi mogu ugnježđivati. Element može sadržavati atribute koji se nalaze u početnoj oznaci na način da se navede naziv atributa i njegova vrijednost koja je pod navodnicima pridružena nazivu znakom jednakosti („=“). [5]

U nastavku je priložen kratki izmišljeni primjer XML dokumenta za biblioteku gdje će biti jasnije objašnjeni koncepti.

```
<knjizara>
    <knjiga id="1" dostupnost="da">
        <naslov>1984</naslov>
        <autor>George Orwell</autor>
        <godina>1949</godina>
        <zanr>znanstvena fantastika</zanr>
        <cijena valuta="EUR">15</cijena>
    </knjiga>
    <knjiga id="2" dostupnost="ne">
        <naslov>Ubiti pticu rugalicu</naslov>
        <autor>Harper Lee</autor>
        <godina>1960</godina>
        <zanr>fikcija</zanr>
        <cijena valuta="USD">12.99</cijena>
    </knjiga>
</knjizara>
```

Na primjeru možemo vidjeti kako je glavni element dokumenta „knjizara“ koji se sastoji od elemenata „knjiga“, nazivi elemenata se mogu ponavljati. Na elementima „knjizara“ i „knjiga“ može se vidjeti primjer ugnježđivanja gdje se u elementu „knjiga“ nalaze elementi koji opisuju knjigu. Atributi su vidljivi na elementima „knjiga“ koji su „id“ i „dostupnost“. Cijena knjige predstavlja broj koliko novčanih jedinica knjiga košta te ne bi mogli znati stvarnu vrijednost knjige bez atributa „valuta“ koja govori o kojoj je točno novčanoj jedinici riječ.

XML može imati definiranu shemu u kojoj je moguće postaviti strukturu XML dokumenta, tipove podataka, broj ponavljanja elemenata, minimalne i maksimalne vrijednosti, napraviti korisničke tipove podataka. Sama XML shema je XML dokument.[5] U nastavku je dan primjer XML sheme.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="knjizara">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="knjiga" maxOccurs="unbounded" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="naslov" type="xs:string" />
              <xs:element name="autor" type="xs:string" />
              <xs:element name="cijena" type="xs:float" />
            </xs:sequence>
```

```
            <xs:attribute name="zanr" type="xs:string" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="lokacija" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Na početku sheme specificiran je URL koji je potreban kako bi se mogli koristiti podaci i atributi koji definiraju samo shemu. Shema definira početni element „knjizara“ koji se sastoji od više knjiga. Elementi koji u sebi sadrže elemente su tipa „complexType“, a ti se elementi navode u „sequence“ elementu. U ovom slučaju nalaze se više „knjiga“ elemenata koji su opet tipa „complexType“ koji sadrži nekoliko elemenata. Svojstvo „name“ definira kako će se element zvati, „type“ koji tip podataka će se nalaziti u elementu. Svojstva „minOccurs“ i „maxOccurs“ označuju koliko puta će se element pojavljivati, trenutno je postavljeno tako da element „knjiga“ ne mora pojavljivati niti jednom, a može i neograničen broj puta. Na kraju se navode svojstva koje će element „knjizara“ imati. Za primjer je postavljeno svojstvo „lokacija“ koji je tekstualni tip podataka.

XML dokumenti se prema Lena Wiese [5] mogu spremiti u relacijske baze podataka na više načina. Jedan od načina je da se koristi SQL/XML u kojemu se može koristiti tip podataka XML koji podržava spremanje cijelog dokumenta u atributu relacije te se za dohvaćanje podataka mogu koristiti ugrađene funkcije koje pretvaraju XML dokument u relaciju. Drugi način je da se prema XML shemi naprave relacije u koju će se spremati podaci gdje će svaka relacija predstavljati pojedinu vrstu XML elementa. Do problema u ovakvom načinu može doći kada se postavlja pitanje redoslijeda pojavljivanja ugniježđenih elemenata i do koje razine je provedena ugniježđenost. Treći način pohrane je da se ne korist XML shema, nego da se napravi jedna relacija u koju će se spremati svi elementi, atributi i vrijednosti elemenata. Za svaki XML dokument napravila bi se tablica koja bi za svaki od navedenih elemenata sadržavala poseban ID, kojeg je tipa zapis, naziv i vrijednost koja je pohranjena gdje će neki zapisi, kao što je na primjer vrijednost za tekst sadržan u elementu, za naziv imati praznu vrijednost. Svi zapisi imati će vanjski ključ na zapis kojem pripadaju.

3.4. Prednosti i nedostaci relacijskih baza podataka

Sustav za upravljanje relacijskim bazama podataka svoju prednost ima što koristi standardan jezik (SQL) kojim se mogu izvoditi kompleksne operacije i općenito sve radnje u bazi. Relacijske baze podataka imaju mogućnost provjeravanja ograničenja postavljenih u bazi podataka te u svakom trenutku osiguravaju da unos, promjena ili brisanje podataka budu u

skladu s njima te se tako i održava integritet podataka. U bazu podataka mogu se dodati različiti korisnici i grupe korisnika kojima su dodijeljena prava koja mogu biti definirana na razini cijele baze podataka kao i na razini pojedine radnje za određenu relaciju čime je baza osigurana od čitanja ili izmjene podataka neautoriziranim korisnicima. Podaci u relacijskim bazama podataka se ne bi trebali duplicirati budući da moraju biti dobro organizirani te nad svakom radnjom u relacijskim bazama podataka mogu se definirati procedure ili okidači koji omogućavaju izvršavanje definiranih radnji prilikom manipuliranja podacima. Različiti korisnici mogu istovremeno pristupati bazi i čitati ili izmjenjivati podatke [5], [7]

Kada se podaci pohranjuju u relacijsku bazu podataka moraju biti spremljeni u relaciji u obliku tablica, a za određene vrste podataka pohrana u relacijsku bazu podataka može predstavljati problem. Podaci se mogu pojaviti u različitim formatima kao što su JSON i XML te bi prilagođavanje podataka predstavljenim u takvim oblicima za pohranu u relacije moglo biti kompliciranо. Prilikom dohvaćanja ovakvog tipa podataka pohranjenih u relacije moglo bi doći do kompliciranosti dostupnosti jer su podaci raštrkani po više relacija te bi se za dohvaćanje jednog entiteta trebalo spojiti više tablica. Između entiteta može postojati više različitih veza koje je potrebno realizirati što može zahtijevati da se kreira više tablica koje bi implementirale veze. Model baze podataka je vrlo striktno definiran te bi neke promjene morale značiti kompletno reorganiziranje podataka kao što je promjena tipova podataka, dodavanje novih stupaca u relaciju itd. [5]

4. Modeliranje baze podataka ljekarne

Kako bi se napravio ispravan model baze podataka za potrebe ljekarne potrebno je koristiti određene tehnike i poslužiti se dostupnim resursima informacija o poslovanju ljekarne. „Model podataka je dio modela koji prikazuju cjelokupni informacijski sustav.“ [8] Prije same izrade sheme baze podataka, potrebno je provjeriti jesu li svi potrebni entiteti prepoznati i njihovi atributi pokriveni zato je lakše napraviti dijagram entiteti-veze (DEV, EV) prema kojem se radi relacijska shema i ERA model.

Izrada modela podataka može se temeljiti na puno različitih izvora koji mogu biti postojeći informacijski sustav, dokumenti korišteni u poslovanju, poslovna pravila, postojeća baza podataka itd. Postoje razne definirane strategija modeliranja baza podataka od kojih se može koristiti više tehnika istodobno za modeliranje, za potrebe izrade modela podataka ovog završnog rada koristit će se informacijski sustav ljekarne, dokumenti te će sve biti popraćeno poslovnim pravilima ljekarne. Za metodu podataka odabrana je metoda VATEK (Vrijednost, Atribut, Tip Entiteta, Ključ) koju spominje i objašnjava Pavlić [8] gdje spominje korake modeliranje podataka po VATEK-u:

1. *Uočiti podatak*
2. *Odrediti tip vrijednosti*
3. *Imenovati atribut čija je to vrijednost*
4. *Uočiti entitet koji ima taj atribut*
5. *Klasificirati entitet koji ima taj atribut*
6. *Klasificirati entitete u tip entiteta*
7. *Pronaći ključ tipa entiteta*
8. *Odrediti brojnost tipa veze*
9. *Odrediti brojnost tipa atributa*
10. *Dodati izvedene atribute*
11. *Dekomponirati složene tipove veza*
12. *Specijalizirati podtipove tipa entiteta*
13. *Definirati specijalne tipove veza i slabe tipove entiteta*
14. *Definirati povratne tipove veza*

Za svaki podatak iz dokumenta provode se navedene aktivnosti, kada se za podatak ustanove svi kriteriji, uzima se sljedeći podatak sve dok se ne iscrpe svi podaci. Za potrebe ovog rada vjerojatno se neće provoditi svi navedeni koraci u slučaju da model neće imati složene tipove entiteta ili se neće pojavljivati povratne veze.

4.1. Opis poslovnih pravila ljekarne

Ljekarna dobiva lijekove tako da kreira narudžbe za dobavljače. Svaka narudžba ima svoj identifikacijski broj, datum, status (zaključena ili u izradi), dobavljača od kojeg se lijekovi naručuju, napomenu, naručene artikle te vrijednost narudžbe.

Svaki dobavljač ima svoju šifru, naziv, OIB, IBAN, email adresu i adresu.

Kada naručeni lijekovi stignu, radnik u ljekarni dobije račun-otpremnicu dobavljača na kojoj se nalazi popis naručene robe prema kojoj se radi primka u sustavu. Primka ima svoj broj, datum knjiženja, dobavljača, stavke te opcionalno narudžbu na koju se veže.

Artikl ima svoj naziv, jedinicu mjere, stanje zaliha i cijenu.

Lijekovi se mogu izdavati i preko recepta kojeg izdaje liječnik, podaci koji se nalaze na receptu su matični broj pacijenta, šifra liječnika, šifra ustanove te lijekovi koji se moraju podići.

Podaci za pacijenta su matični broj, ime, prezime, adresa, datum rođenja.

Podaci za liječnika su šifra, ime, prezime, adresa, telefon.

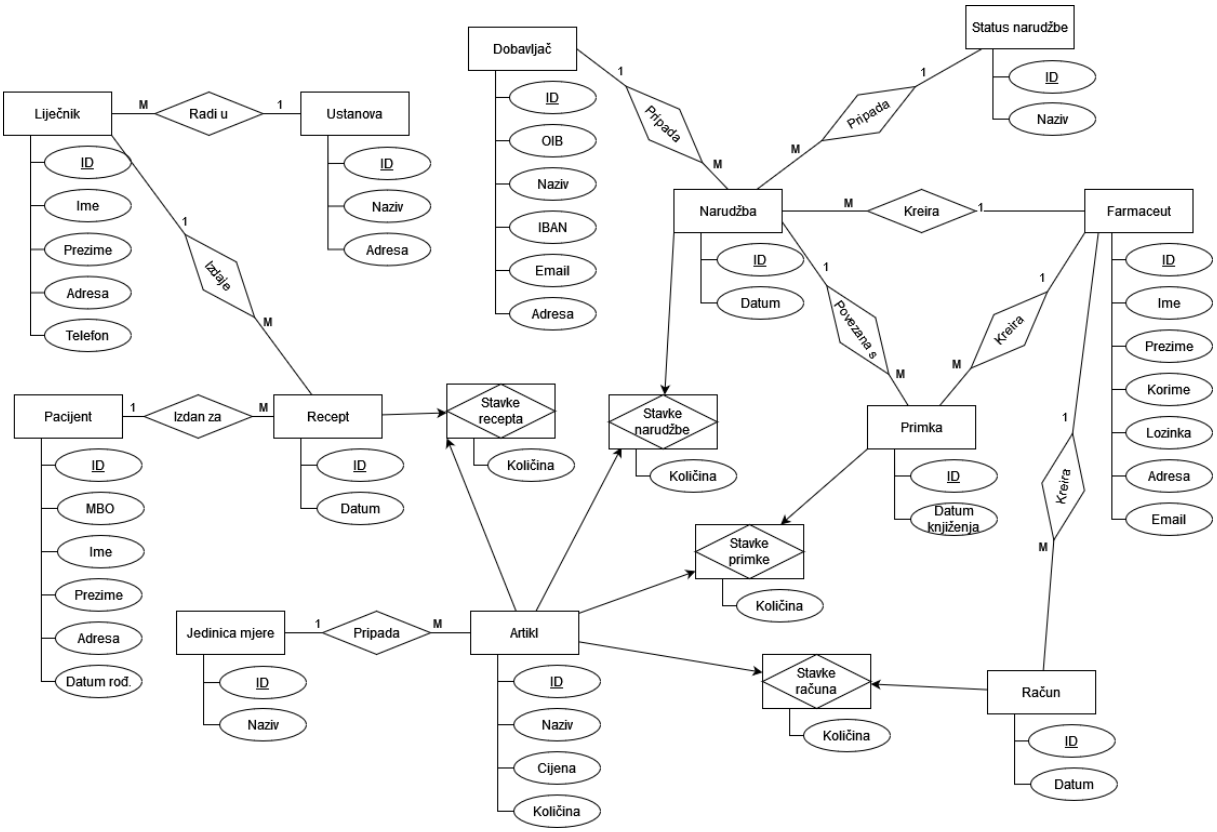
Podaci ustanove su šifra, naziv i adresa.

Radnik u ljekarni izdaje račune za svu robu koja izlazi iz ljekarne. Na računu se nalaze informacije o prodanim artiklima i količinama, informacije o ljekarni (naziv, adresa, OIB, telefon), ukupna cijena, broj računa, datum i vrijeme izdavanja i šifra operatera.

Za farmaceuta koji je u ljekarni zaposlen osim osobnih podataka potrebni su i podaci s kojima se prijavljuje u sustav tako da podaci koji se spremaju su ID, ime, prezime, adresa, lozinka, korisničko ime i email.

4.2. Dijagram entiteti-veze i relacijska shema

Kako bi bilo jednostavnije modeliranje baze podataka za ljekarnu, korištena je metoda entiteti-veze koja olakšava pregled podataka grafičkim prikazom. Metoda grafički prikazuje entitete i njihovu povezanost unutar odabranog sustava. Sam dijagram koji prikazuje metodu entiteti-veze naziva se dijagram entiteti-veze (Slika 3). Na slici entiteti su označeni kao pravokutnici koji su međusobno povezani vezama prikazanih u obliku romba. Svaki entitet ima svoje atribute koji su na slici označeni kao elipse gdje su podcrtani atributi primarni ključevi. Mogu se primijetiti veze koje imaju atribute, to je agregirani tip veze koji se vodi kao novi entitet te prevođenjem u relacijsku shemu postaje relacija. [8]



Slika 3: Dijagram entiteti-veze ljekarne

Prema dijagramu entiteti-veze moguće je napraviti relacijsku shemu prema kojoj se radi ERA model. U relacijskoj shemi potrebno je navesti sve entitete uključujući i agregirane veze budući da one postaju relacije. Atributi su navedeni u zagradama, a primarni ključevi su podvučeni jednostrukom crtom, a vanjski dvostrukom. Relacijska shema baze podataka nalazi se u nastavku teksta.

USTANOVA (ID, NAZIV, ADRESA)
JEDINICA MJERE (ID, NAZIV)
STATUS NARUDŽBE (ID, NAZIV)
LIJEČNIK (ID, IME, PREZIME, ADRESA, TELEFON, USTANOVAID)
PACIJENT (ID, MBO, IME, PREZIME, ADRESA, DATUM ROĐENJA)
ARTIKL (ID, NAZIV, CIJENA, KOLIČINA, JEDINICAMJEREID)
FARMACEUT (ID, IME, PREZIME, KORIME, LOZINKA, ADRESA, EMAIL)
DOBAVLJAČ (ID, OIB, NAZIV, IBAN, EMAIL, ADRESA)
RECEPT (ID, DATUM, LIJECNIKID, PACIJENTID)
STAVKE RECEPTA (RECEPTID, ARTIKLID, KOLIČINA)
RAČUN (ID, DATUM, FARMACEUTID)
STAVKE RAČUNA (ARTIKLID, RAČUNID, KOLIČINA)

PRIMKA (ID, DATUMKNJIZENJA, DOBAVLJACID, NARUDZBAID, FARMACEUTID)
STAVKE PRIMKE (PRIMKAID, ARTIKLID, KOLIČINA)
NARUDŽBA (ID, DATUM, DOBAVLJACID, STATUSID, FARMACEUTID)
STAVKE NARUDŽBE (ARTIKLID, NARUDZBAID, KOLIČINA)

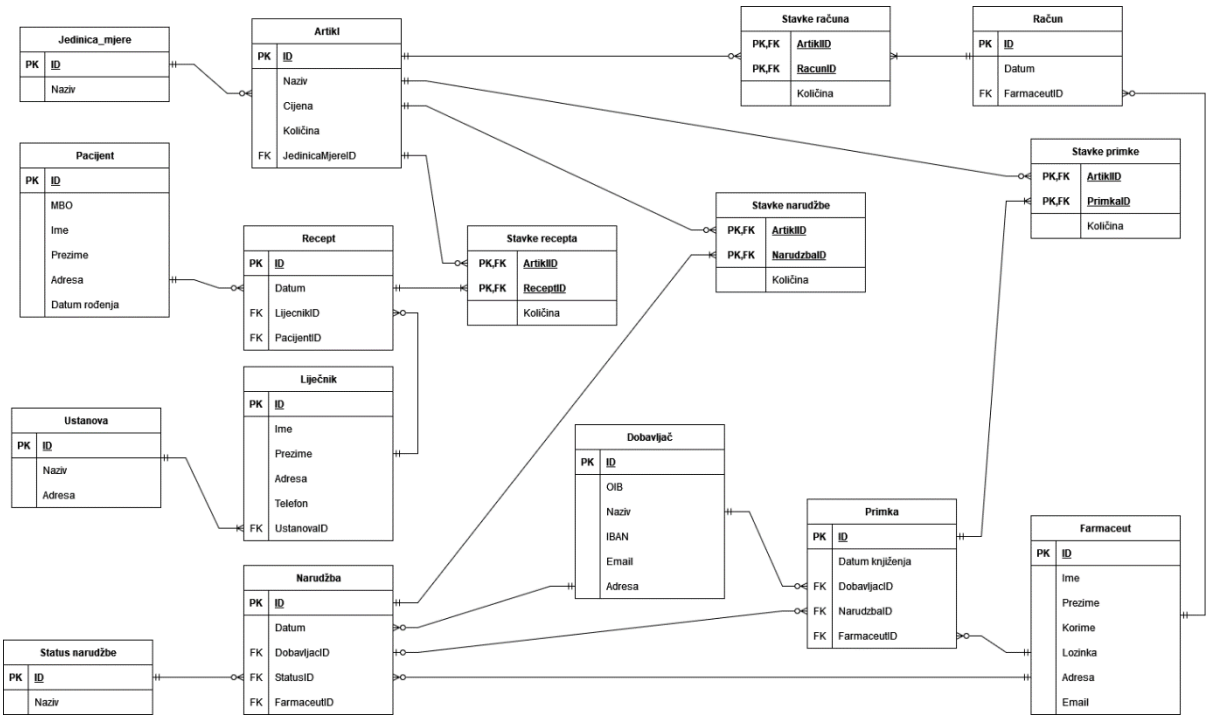
Iz same relacijske sheme možemo vidjeti entitete i njihove atribute, no ne možemo vidjeti koji tipovi vrijednosti se spremaju u atribute. Kako bismo mogli napraviti ERA model potrebno je napraviti rječnik podataka (tablica 6) koji nadopunjuje relacijsku shemu. Rječnik podataka sadrži ime atributa, tip i opis gdje ograničenja podataka opisujemo onako kako je u stvarnosti bez obzira na to kako će biti implementirana u bazi podataka. [3]

Tablica 6: Rječnik podataka za bazu podataka ljekarne

IME ATRIBUTA	TIP	OPIS
ID	Cijeli broj	Generirani cijeli broj
NAZIV	Niz znakova	Naziv ustanove, jedinice mjere, statusa narudžbe itd.
ADRESA	Niz znakova	Adresa osobe ili ustanove
TELEFON	Niz znakova	Broj telefona osobe
MBO	Niz znakova od 9 znamenaka	MBO osobe
DATUM ROĐENJA	Datum	Datum kada se osoba rodila
CIJENA	Decimalni broj	Cijena proizvoda
KOLIČINA	Cijeli broj	Prikazuje stanje artikla ili broj stavki na dokumentu
KORIME	Niz znakova od minimalno 5 znamenaka	Korisničko ime s kojim se korisnik prijavljuje u aplikaciju
LOZINKA	Niz od minimalno 6 znakova	Lozinka korisnika
EMAIL	Niz znakova	Email korisnika ili osobe
OIB	Niz znakova od točno 11 znamenaka	OIB osobe ili tvrtke
IBAN	Niz znakova od točno 21 znamenke	IBAN tvrtke
DATUM	Datum	Datum i vrijeme kada se nešto desilo

5. ERA model baze podataka

Prema relacijskoj shemi napravljen je ERA model baze podataka koji se sastoji od ukupno 16 relacija (Slika 4). U nastavku će biti svaka tablica opisana kao i njezini atributi te će biti opisane i objašnjene veze između svake relacije.



Slika 4: ERA model

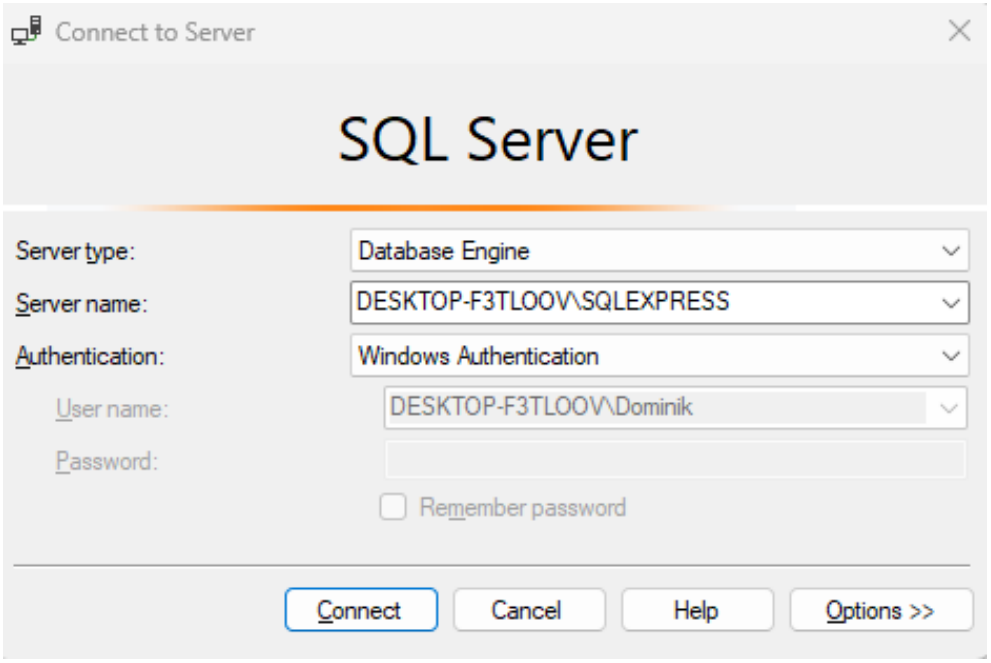
6. SQL Server i SQL Server Management Studio

SQL Server je sustav za upravljanje relacijskim bazama podataka koji je u vlasništvu tvrtke Microsoft. Koristi razne mehanizme kako bi zaštitio podatke prilikom pojavljivanja grešaka ili fizičkih kvarova. Microsoft koristi SQL Server za poslovne aplikacije. [9] Ima ukupno 4 vrste SQL Servera, a za potrebe ovog završnog rada koristit će se „Developer“ verzija.

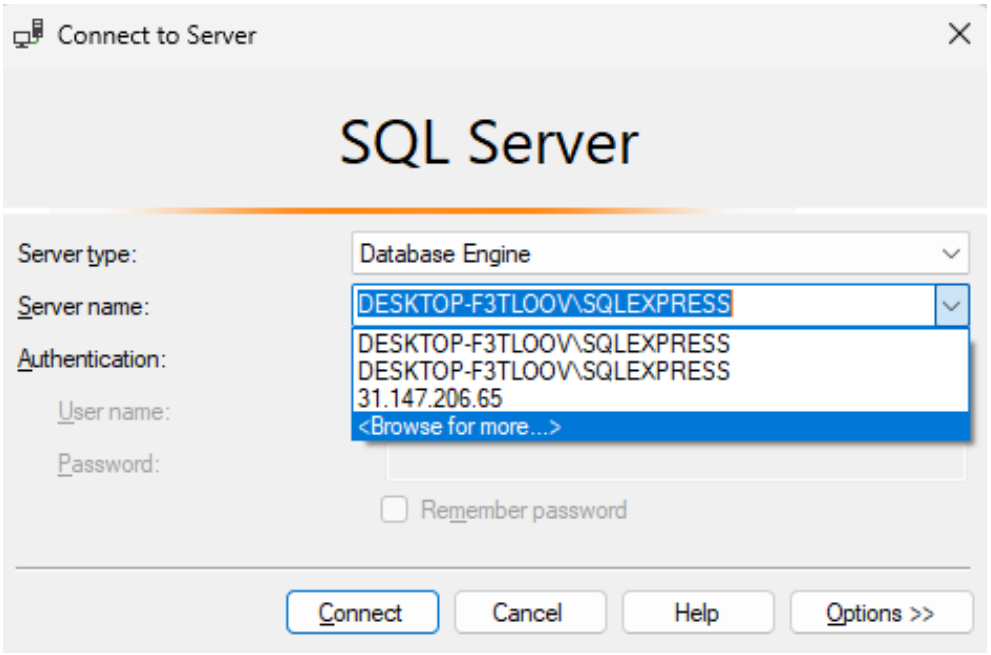
SQL Server Management Studio (SSMS) je okruženje za manipuliranje SQL infrastrukture, a koristit će se za spajanje na SQL Server za kreiranje baze podataka. SSMS služi za administraciju i konfiguriranje svih komponenata SQL Servera i Azure SQL baza podataka. Trenutno se SSMS može koristiti samo na Windows operacijskim sustavima. [10]

6.1. Postavljanje SQL Servera

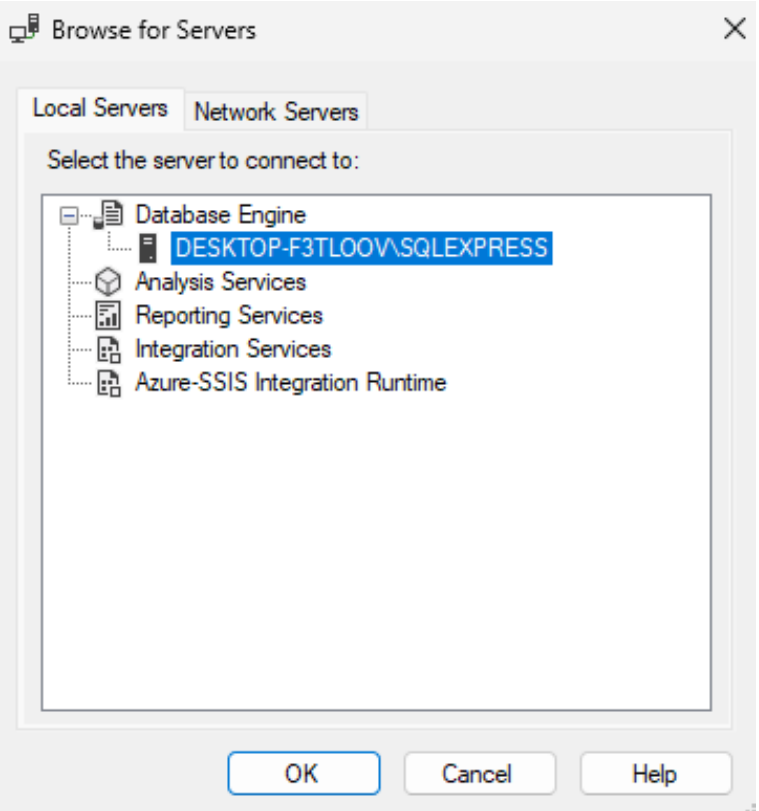
Za početak je potrebno instalirati SQL server i SQL Server Management Studio. Koraci instalacije oba programa vrlo su jednostavni tako da se ovdje neće obrađivati. Dovoljno je samo reći kako je potrebno preuzeti besplatnu verziju SQL Servera za developere. Instalacije programa vrlo su jednostavne te je potrebno samo slijediti upute instalacije nakon pokretanja. Kada su oba programa instalirana potrebno je pokrenuti SQL Server Management Studio te se spojiti na lokalni SQL Server. Prilikom pokretanja program će sam pokrenuti formu za spajanje na kojoj je potrebno odabrati lokalni server i za vrstu autentifikacije Windows Authentication (Slika 5). Ukoliko se na listi servera ne nalazi naš server potrebno je ručno odabrati na način da kliknemo padajući izbornik i odaberemo opciju „Browse for more“ (Slika 6). U prozoru koji se otvorio potrebno je odabrati karticu „Local Servers“ i proširiti kategoriju „Database Engine“ u kojoj bi trebali vidjeti lokalni server (Slika 7). Nakon odabira servera potrebno je spojiti se.



Slika 5: Spajanje na SQL Server preko Windows Authentication

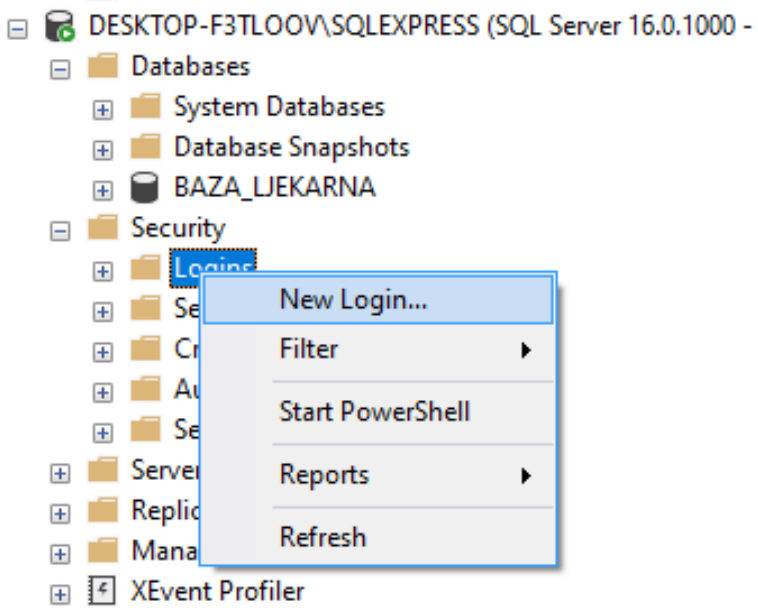


Slika 6: SQL Server Browse for more

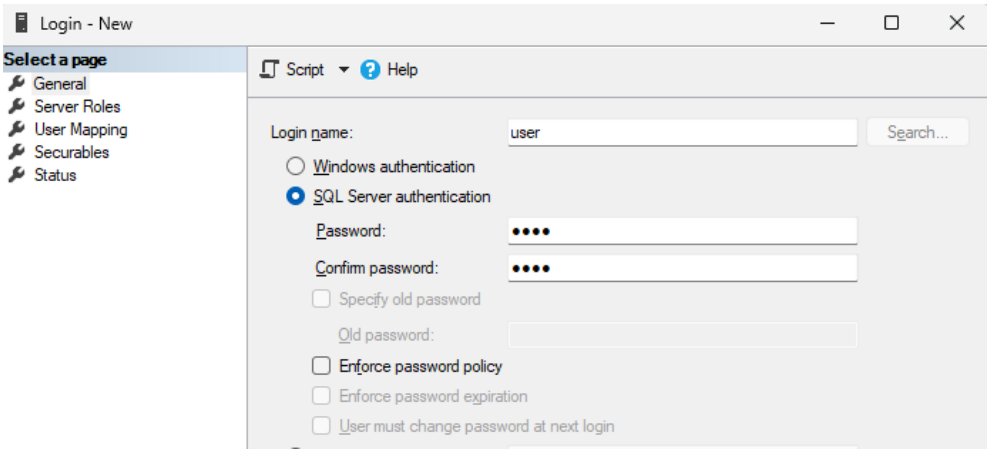


Slika 7: Prikaz lokalnih SQL servera

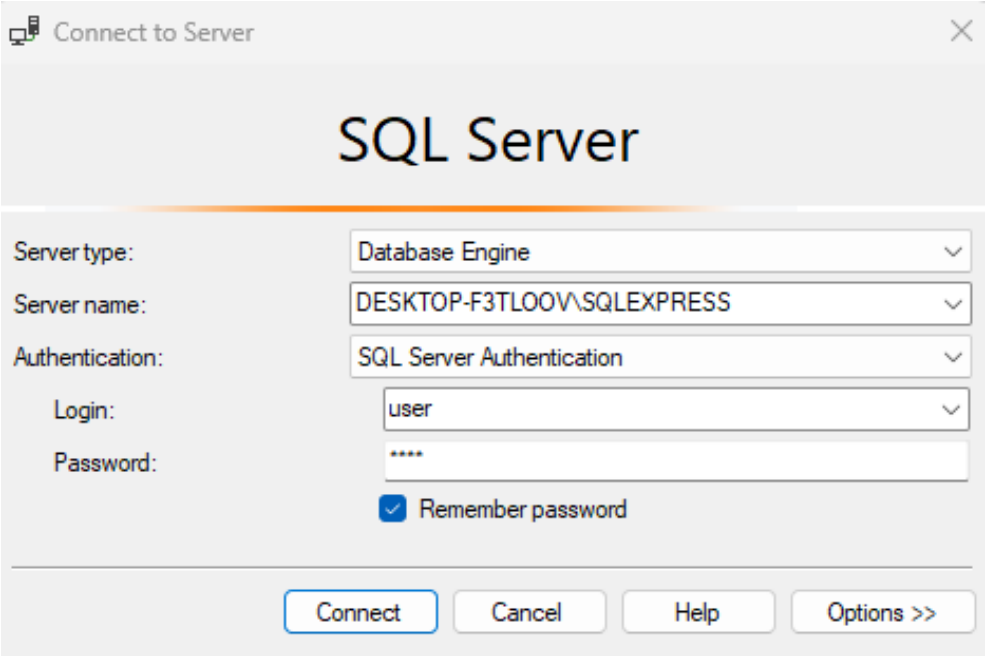
Kada smo spojeni na server potrebno je dodati korisnika preko kojeg će se aplikacija spojiti na server. Proširujemo mapu „Security“ i na mapu „Logins“ kliknemo desni klik i odaberemo opciju „New Login“ (Slika 8). Otvara se prozor u kojem je potrebno upisati naziv korisnika i odabire se opcija „SQL Server authentication“ te se upisuje lozinka za korisnika. Nakon upisa lozinke potrebno je isključiti opciju „Enforce password policy“ i dodati korisnika (Slika 9). Za spajanje na bazu korisnika opet se spajamo ali ovog puta odabire se opcija „SQL Server Authentication“ i upisuju se podaci dodanog korisnika (Slika 10). Kako bi se vidjelo da smo na server spojeni preko korisnika u prikazu veze trebalo bi pisati ime korisnika kojeg smo dodali (Slika 11).



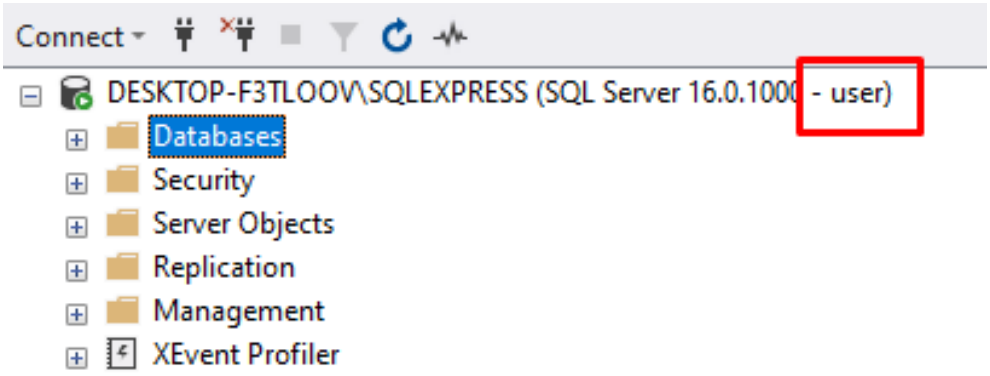
Slika 8: Dodavanje novog korisnika na server



Slika 9: Kreiranje korisnika na serveru



Slika 10: Spajanje na server s dodanim korisnikom



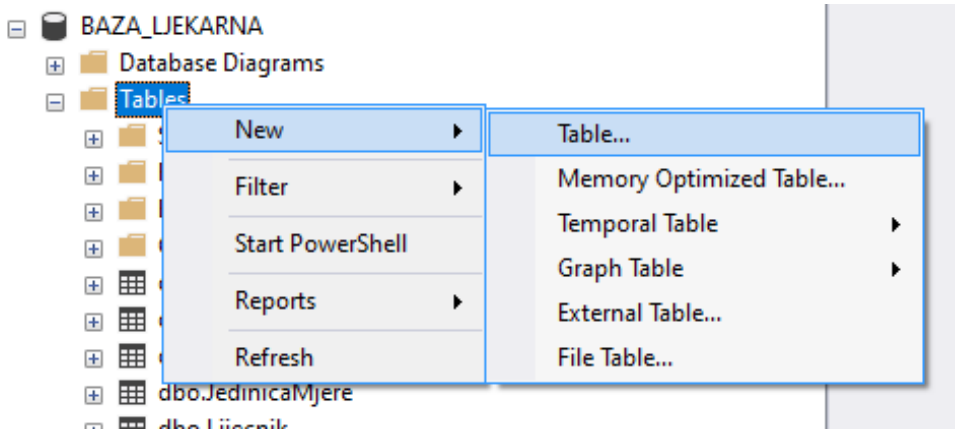
Slika 11: Veza sa serverom preko dodanog korisnika

6.2. Implementacija baze podataka

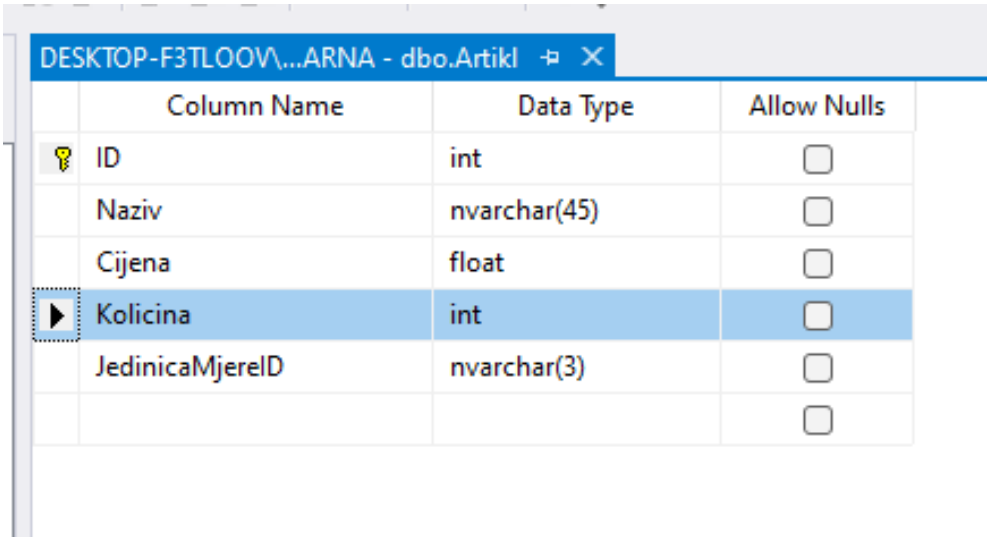
Implementacija baze podataka je napravljena preko alata SSMS u kojem su se izrađivale tablice, dodavali atributi, postavljali primarni ključevi i implementirale veze između relacija. Potrebno je bilo dodati UNIQUE indekse i ograničenja na neke atribute. Koraci implementacije baze u SSMS biti će pokazani u nastavku.

6.2.1. Kreiranje tablica i postavljanje primarnih ključeva

Prije nego što se mogu napraviti tablice podrazumijeva se da je kreirana prazna baza podataka. U slučaju da baza ne postoji, jednostavno se može napraviti izvršavanjem upita „CREATE DATABASE“. Kreiranje tablica preko grafičkog sučelja radi se tako da se na mapu „Tables“ napravi desni klik, odabere opcija „New“ i zatim „Table“ (Slika 12). Kreirati će se tablica gdje se mogu dodavati atributi i njihovi tipovi podataka (Slika 13).



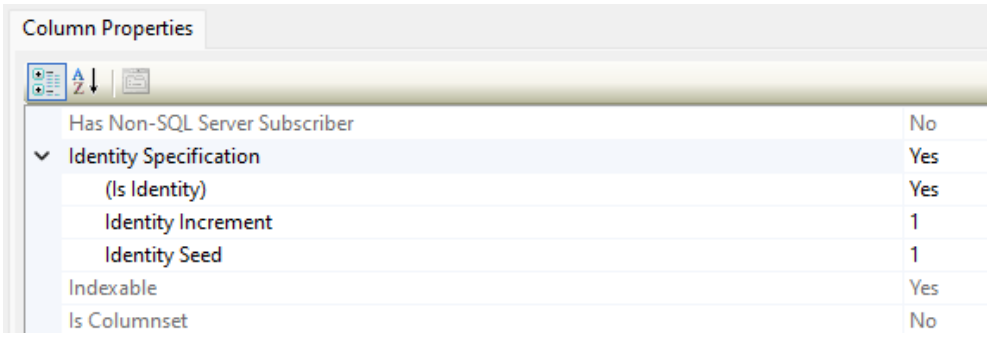
Slika 12: Dodavanje nove tablice



Slika 13: Prikaz tablice u SSMS

Primarni ključ postavlja se na način da se napravi desni klik na atribut za koji želimo da bude primarni ključ i odaberemo opciju „Set Primary Key“. Ako želimo da se vrijednost ključa automatski generira i povećava moramo odabrati atribut koji je primarni ključ i u odjeljku „Column Properties“ pronaći kategoriju „Identity Specification“ (Slika 14). Vrijednost „Is

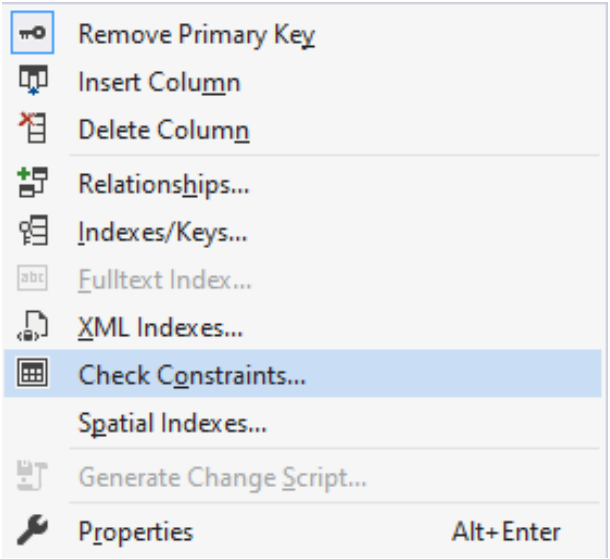
Identity“ mora se postaviti na „Yes“, „Identity Increment“ predstavlja broj za koliko će se vrijednost povećavati, a „Identity Seed“ početnu vrijednost.



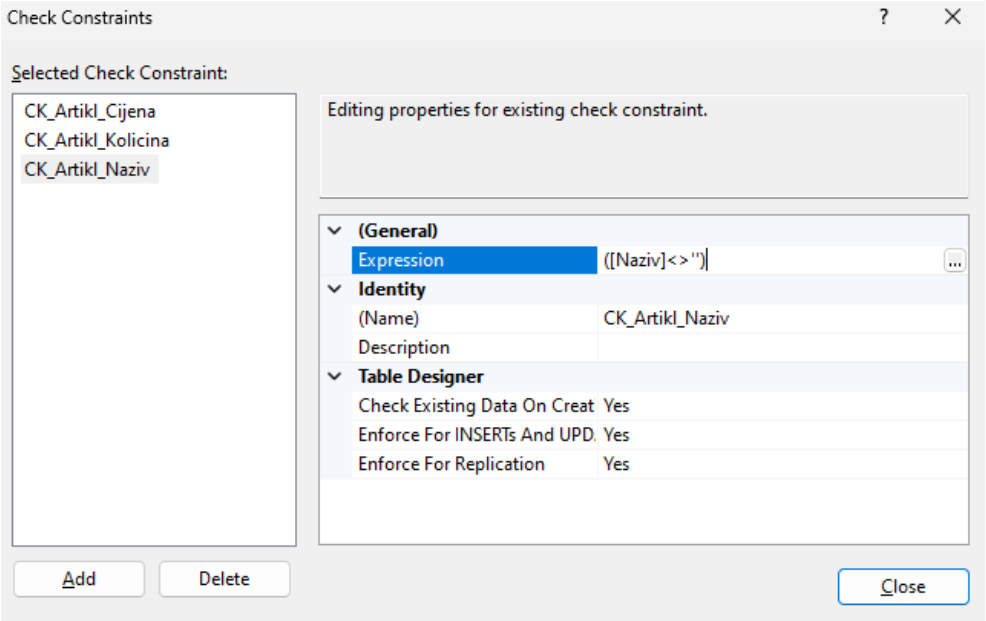
Slika 14: Automatsko generiranje vrijednosti primarnog ključa

6.2.2. Postavljanje ograničenja i UNIQUE indeksa

Tablica Artikl ima atribut naziv koji mora biti jedinstven i ne smije biti prazan niz. Kako bismo dodali ograničenje u SSMS prilikom dizajniranja tablice napravimo desni klik i odaberemo „Check Constraints“ (Slika 15) te nam se otvara prozor u kojem postavljamo ograničenja. Da bismo dodali ograničenje, kliknemo na gumb „Add“ i jednostavno upišemo naziv ograničenja i izraz (Slika 16). U ovom slučaju je to zadavanje da naziv artikla ne bude prazan niz.

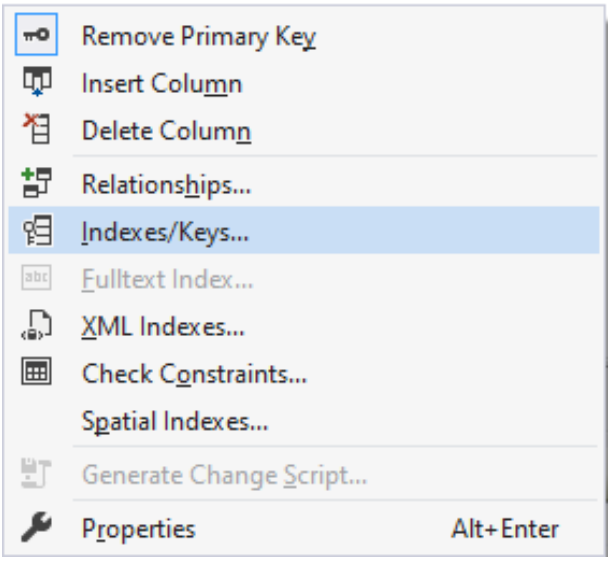


Slika 15: Otvaranje forme za postavljanje ograničenja

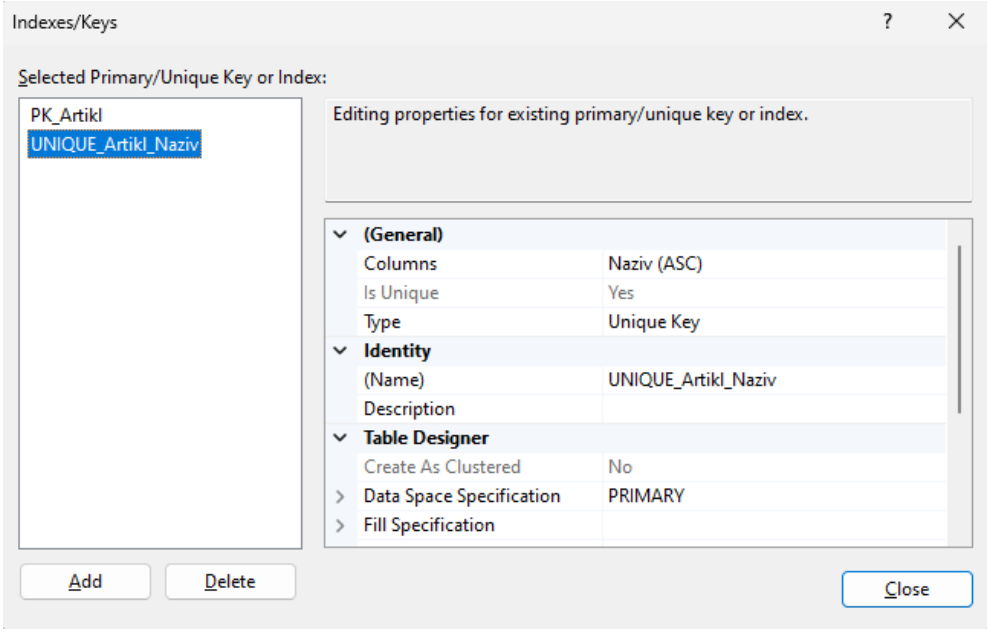


Slika 16: Forma za dodavanje ograničenja

Za postavljanje UNIQUE indeksa slično kao i za ograničenje, kliknemo desni klik i odaberemo opciju „Indeyes/Keys“ (Slika 17) i otvara se nova forma u kojoj odaberemo atribut na koji želimo postaviti indeks (Slika 18). Pod „Type“ se postavi opcija „Unique Key“ i unese se naziv indeksa.



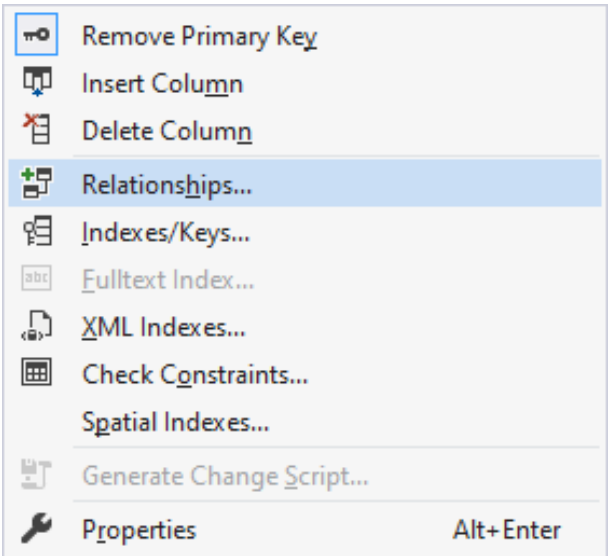
Slika 17: Otvaranje forme za postavljanje indeksa



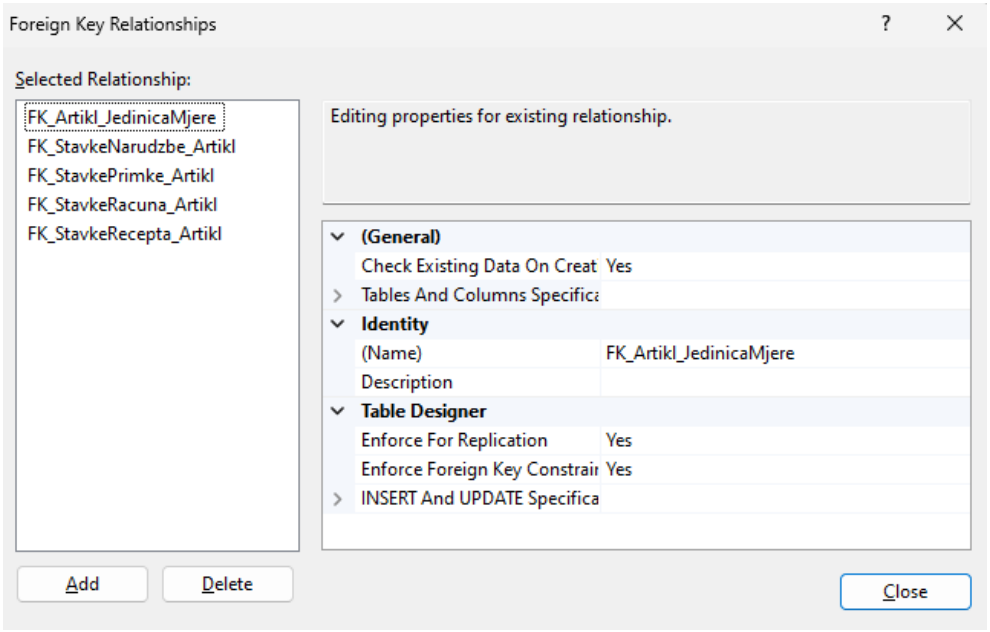
Slika 18: Forma za dodavanje indeksa

6.2.3. Definiranje veza i vanjskih ključeva

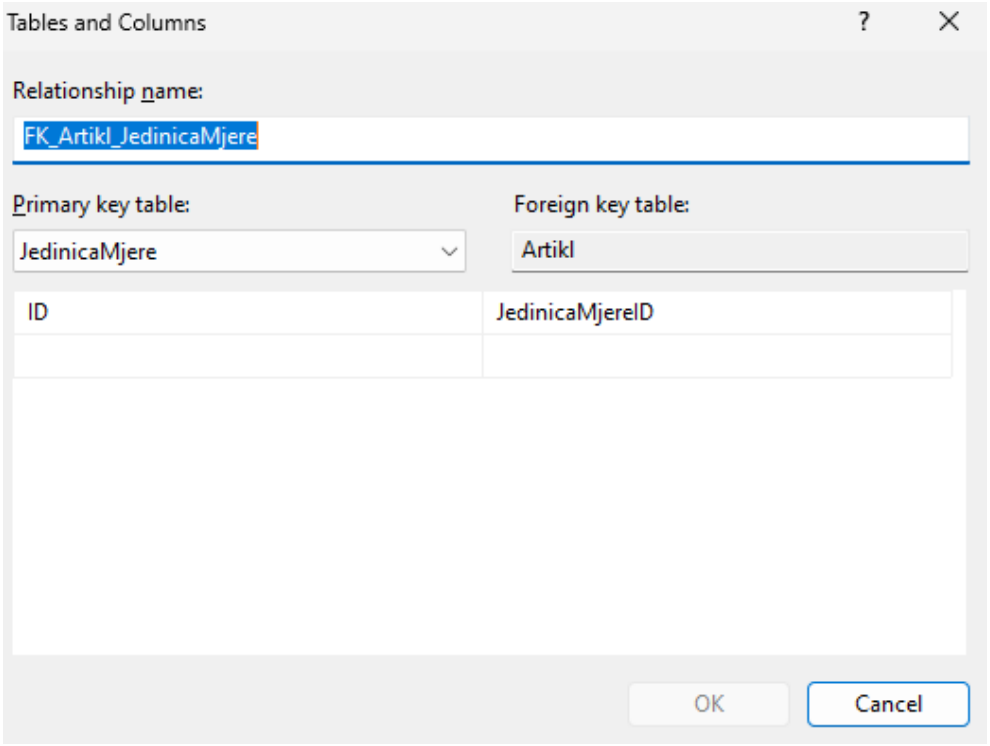
Kako bi se dodao vanjski ključ, u tablici mora postojati atribut koji će biti vanjski ključ. Da bismo atribut povezali s tablicom, potrebno je napraviti desni klik i odabrati opciju „Relationships“ (Slika 19) gdje se otvara forma (Slika 20) u joj je potrebno kliknuti „Add“. Pod opcijom „Tables And Columns Specifications“ potrebno je kliknuti na tri točkice gdje se otvara nova forma u kojoj definiramo vanjski ključ. S lijeve strane odabire se tablica na koju želimo da je vanjski ključ povezan i na koji atribut. S desne strane odabiremo koji atribut u trenutnoj tablici želimo da bude vanjski ključ, a pri vrhu upišemo naziv vanjskog ključa (Slika 21).



Slika 19: Otvaranje forme za kreiranje vanjskih ključeva



Slika 20: Forma za kreiranje vanjskih ključeva



Slika 21: Specifikacija vanjskog ključa

7. Aplikacija

Aplikacija je rađena kao grafičko sučelje za upravljanje bazom podataka. Tehnologije koje se koriste su .NET platforma i Windows Forms koji služi za izradu grafičkog sučelja. Zamišljeno je da se aplikacija pokreće na Windows operacijskim sustavima. Cijeli programski kod aplikacije dostupan je na GitHub repozitoriju koji se nalazi na sljedećoj poveznici: https://github.com/dojosipovic/zavrsni-rad-ljekarna. U nastavku poglavlja opisana će biti arhitektura aplikacije, tehnologije pomoću kojih se izrađivala te sam programski kod i grafičko sučelje.

7.1. Arhitektura aplikacije

Aplikacija je napravljena u troslojnoj arhitekturi gdje su posebno odvojeni grafičko sučelje, logika i pristup podacima iz baze, odnosno u slojeve prezentacijski, sloj poslovne logike i sloj pristupa podacima (Slika 22).

Slika 22: Arhitektura aplikacije (Prema [11])

U troslojnoj arhitekturi interakcija s podacima kreće se kroz slojeve gdje svaki sloj zna samo za onaj sloj ispod sebe npr. prezentacijski sloj ne zna da postoji sloj pristupa podacima i obratno, ali zna da postoji sloj poslovne logike. Interakcija korisnika odvija se preko prezentacijskog sloja i svaka radnja odvija se na način da se prolazi kroz sve slojeve i na kraju dolazi do baze podataka gdje se zapravo odvijaju operacije nad podacima. Prezentacijski sloj

služi za prezentiranje podataka, komponente sučelja i preko njega se odvija komunikacija između korisnika i programa budući da se na prezentacijskom sloju nalazi korisničko sučelje. Programska logika trebala bi biti minimalna jer se na prezentacijskom sloju pozivaju „servisi“ sa sloja poslovne logike na kojoj bi se trebala nalaziti složenija logika koja služi za obradu podataka proslijeđenih od prezentacijskog sloja. Na sloju pristupa podacima odvija se komunikacija prema bazi podataka na kojem ose odvija promjena, dodavanje ili brisanje podataka. Prednosti korištenja ove arhitekture su što je programski kod organiziran u slojeve gdje je posebno odvojeno grafičko sučelje od sloja koji se zadužen samo za logiku i sloj pristupa podacima koji ima ulogu komunikacije s bazom podataka. Svaki sloj je cjelina za sebe i može se mijenjati, nadograđivati bez da se opterećuju ostali slojevi također sloj se može zamijeniti i ako je arhitektura dobro implementirana zamjena sloja može proći bez da se moraju ostali slojevi prilagođavati. [11]

U aplikaciji napravljenoj za ovu bazu podataka dodan je još sloj entiteta kojeg mogu prepoznati sva tri sloja. Sloj entiteta napravljen je zato što su za komunikaciju s bazom podataka kreirane entitetske klase koje predstavljaju relacije u bazi, a entiteti se moraju koristiti na svakom od sloja kako bi komunikacija između slojeva ispravno funkcionirala.

7.2. .NET platforma i Windows Forms

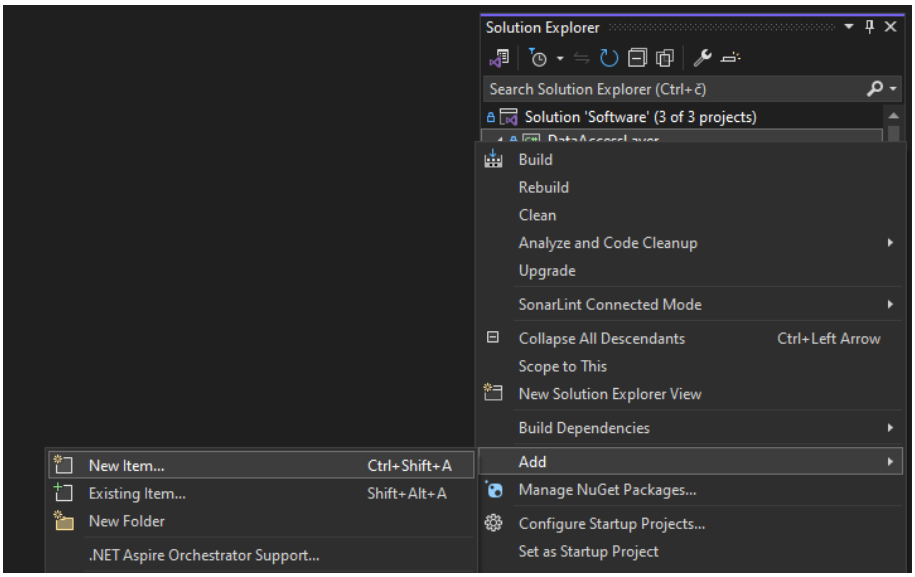
U izradi aplikacije koristio se .NET Framework, a aplikacija je pisana u C# programskom jeziku. U .NET-u je omogućeno korištenje raznih alata i biblioteka koji omogućuju razvijanje modernog softvera, a platforma omogućuje izradu web aplikacija, mobilnih i desktop aplikacija. Prednost .NET platforme je što je potpuno besplatna i otvorenog je koda kojeg održava zajednica programera. Osim C# programskog jezika podržava F# i Visual Basic. Programski kod napisan u C# programskom jeziku može se izvršavati na operacijskom sustavu Windows, Linux i MacOS ni s kakvim ili minimalnim promjenama zahvaljujući .NET runtime koji može kompilirati programski kod na različitim operacijskim sustavima. [12]

Jedna od mnogih biblioteka koja je sadržana u .NET je Windows Forms koja omogućuje izradu GUI aplikacija za Windows operacijske sustave. Uz pomoć Windows Formsa moguće je napraviti grafičko sučelje u kratkom vremenu na vrlo jednostavan način jer podržava gotove komponente koje programer može prilagoditi svojim potrebama. Neke od komponenata su polja za unos teksta, datuma, padajući izbornik, element za učitavanje slike i još mnogi drugi. Elementi se mogu pretplatiti na događaje koje korisnik može napraviti na primjer kada pritisne gumb na formi ili neku određenu tipku na tipkovnici. Omogućeno je pisanje programske logike za svaku formu grafičkog sučelja. [13]

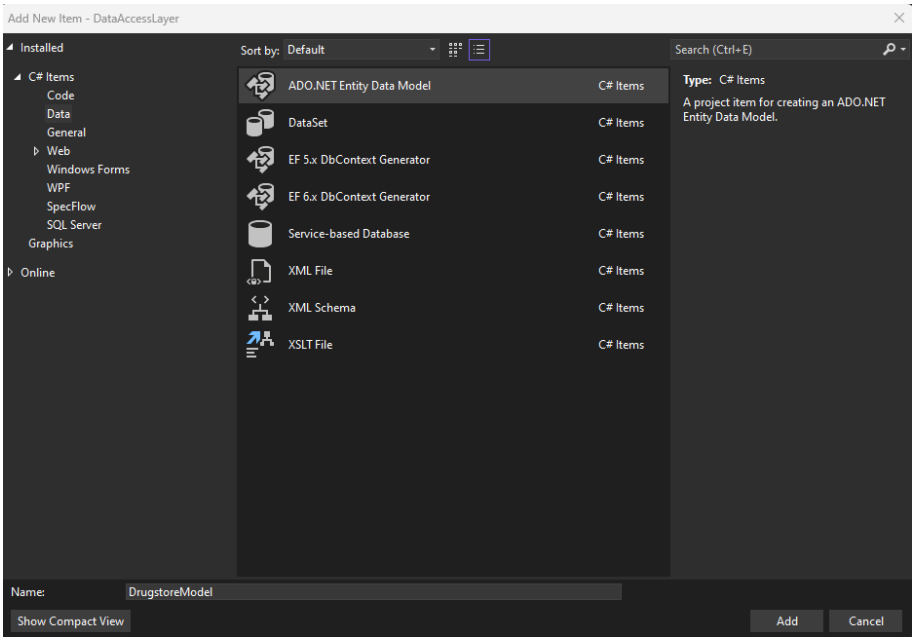
7.3. Entity Framework i spajanje na bazu podataka

Kako bi aplikacija mogla koristiti bazu podataka potrebno je povezati je s bazom. Jedan od načina povezivanja je korištenjem Entity Frameworka (EF) koji omogućuje povezivanje s bazom i automatsko generiranje entitetskih klasa prema postojećoj bazi podataka. Korištenjem EF omogućeno je lakše manipuliranje bazom podataka jer sve operacije koje se trebaju izvršiti prema bazi podataka, izvršavaju se nad klasama. Na ovaj način omogućeno je preko programa i korištenjem klasa manipulirati podacima koji se nalaze u bazi podataka tako da programer uopće nema dojam da koristi bazu. [14]

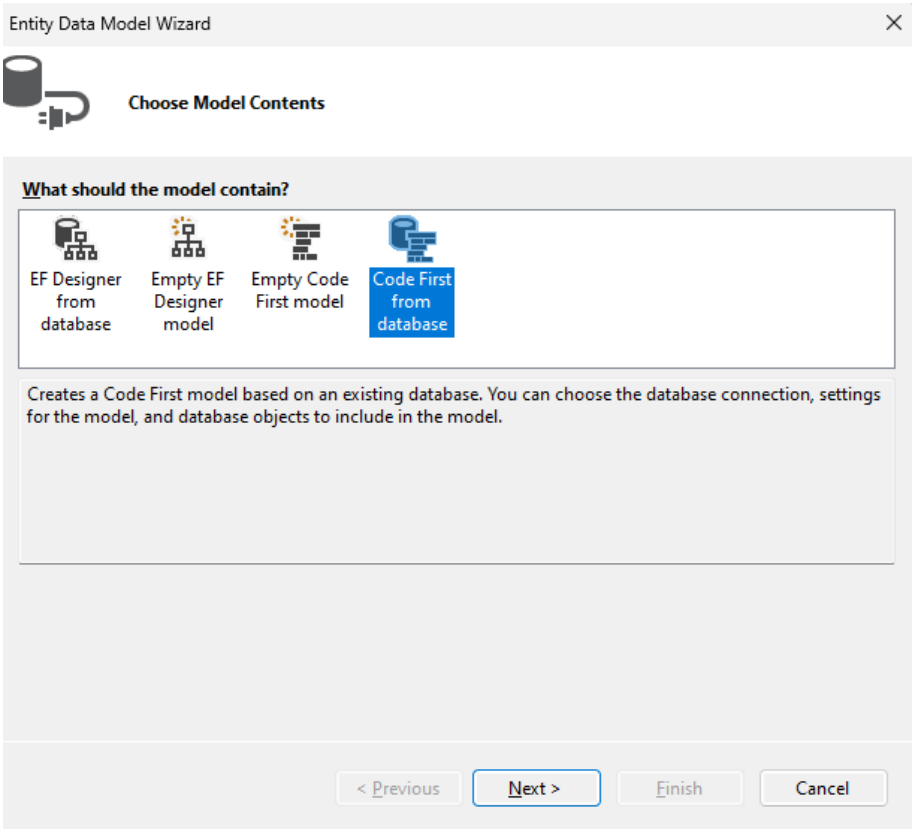
Za spajanje na bazu podataka podrazumijeva se da je EF instaliran u projektu, ako nije potrebno je napraviti desni klik na projekt i odabrati opciju „Manage NuGet packages“ te pronaći EntityFramework i instalirati ga. U ovoj aplikaciji za svaki sloj je napravljen poseban projekt i da bi se napravilo spajanje na bazu na projekt „DataAccessLayer“ napravi se desni klik, opcija „Add“, opcija „New item“ nakon čega se otvara novi prozor (Slika 23). U novootvorenom prozoru odabiremo „ADO.NET Entity Data Model“ i dajemo naziv modelu, u ovom slučaju stavio sam naziv „DrugstoreModel“ budući da se spajam na bazu podataka za ljekarnu (Slika 24). U sljedećem prozoru ponuđene su opcije na koji način se modeli mogu dodati u projekt. Budući da je baza podataka već kreirana i želimo dodati entitetske klase, odabire se opcija „Code First from database“ (Slika 25). Na sljedećem prozoru moramo postaviti vezu prema bazi podataka gdje prvo odabiremo opciju „New Connection“ (Slika 26). U novom prozoru upisujemo IP adresu servera, a budući da je server lokalno pokrenut upisujemo naziv koji se koristio za spajanje SSMS-a (Slika 27). Pod polje „Authentication“ postavljamo opciju kao i ranije, na „SQL Server Authentication“ te upisujemo podatke korisnika kojeg smo dodali u bazu preko SSMS-a. Moramo označiti opciju „Trust Server Certificate“ i odabrati bazu podataka na koju se želimo spojiti te nakon toga pritisnemo gumb „OK“. Na prozoru bi sada trebala biti vidljiva veza koju smo napravili te je još potrebno označiti opciju „Yes“ i opciju za App.config datoteku kako bi se „connection string“ spremio u nju jer će kasnije biti potrebno kopirati vrijednost u App.config projekta za prezentacijski sloj (Slika 28). Na kraju se otvara odabir koje sve objekte iz baze želimo uključiti u projekt. Odabiremo sve tablice i isključujemo opciju imenovanja objekta u množini ili jednini (Slika 29) radi toga što ova opcija funkcionira nad objektima s engleskim nazivima.



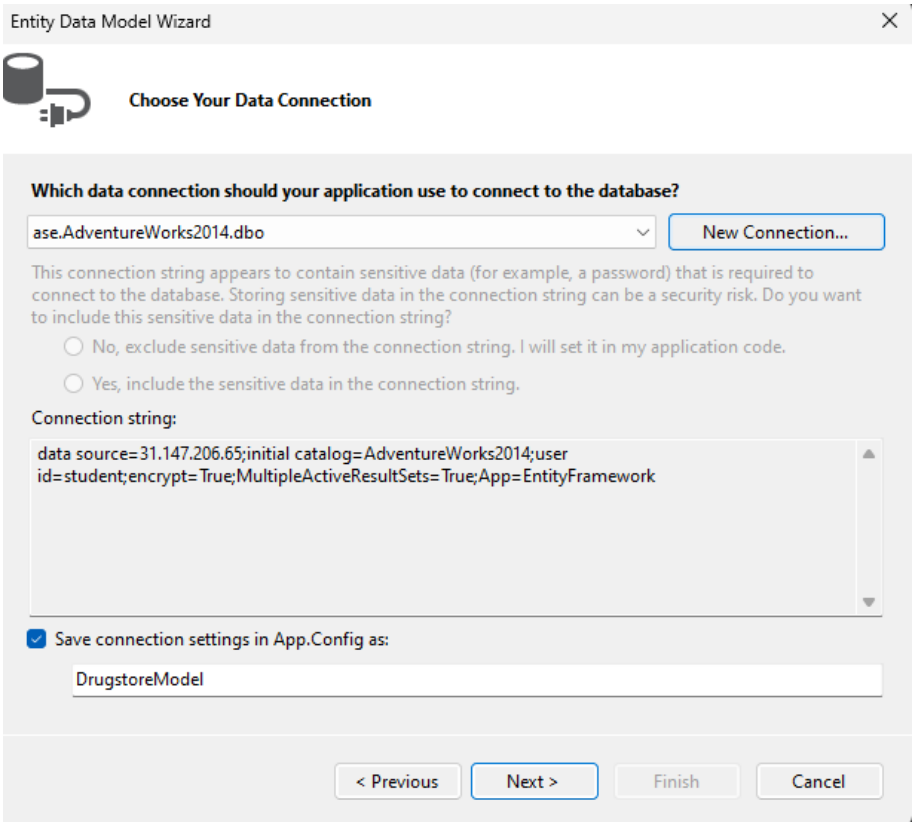
Slika 23: Dodavanje nove stavke u projekt



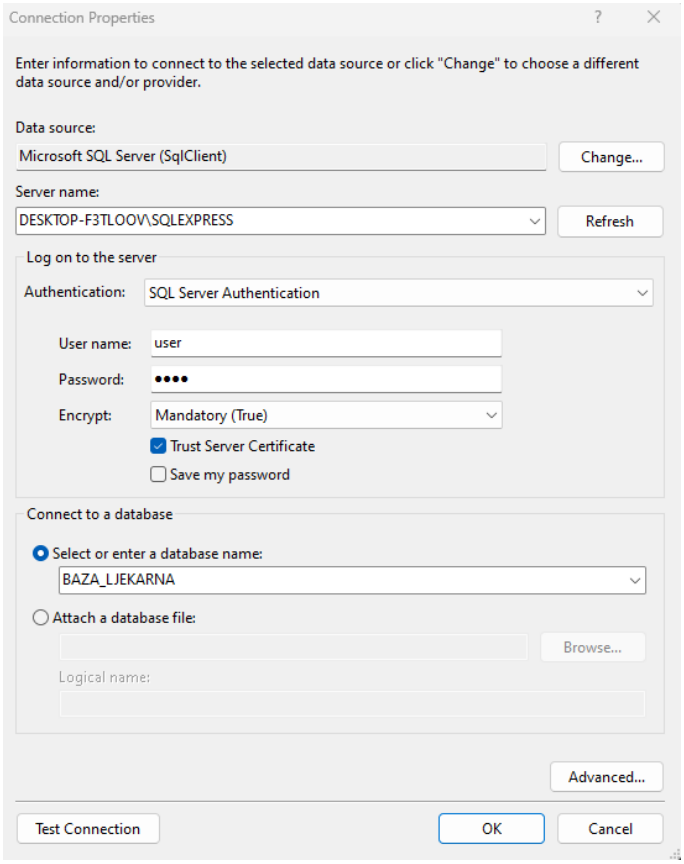
Slika 24: Dodavanje modela baze podataka u projekt



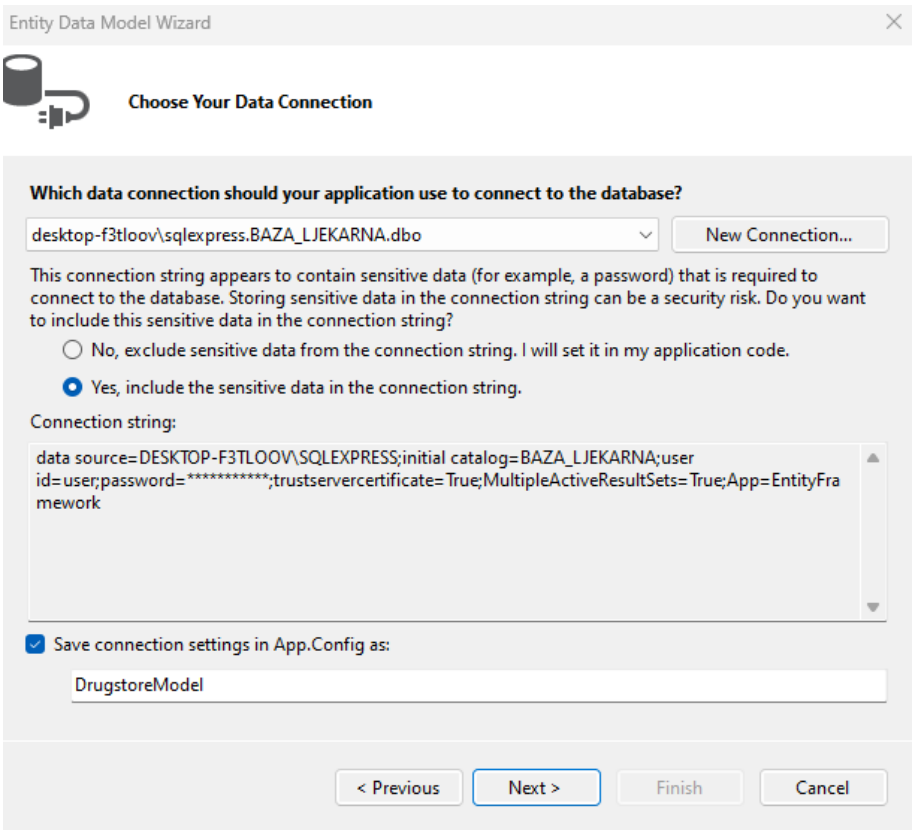
Slika 25: Odabir opcije kreiranja modela



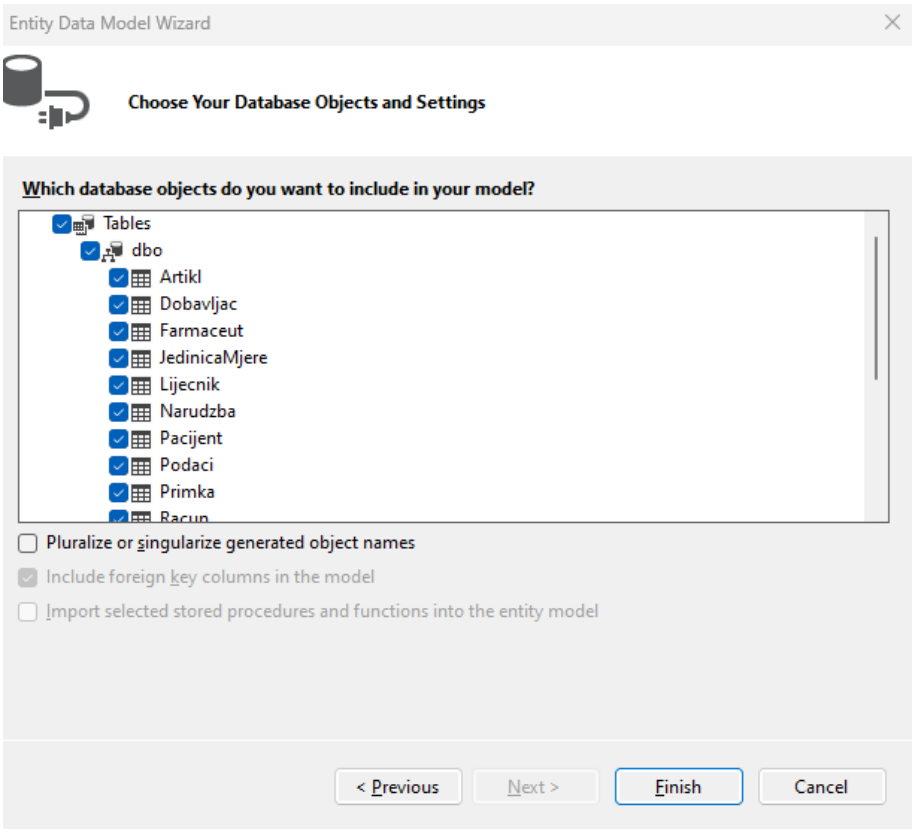
Slika 26: Dodavanje nove veze prema bazi



Slika 27: Upis podataka za spajanje s bazom podataka



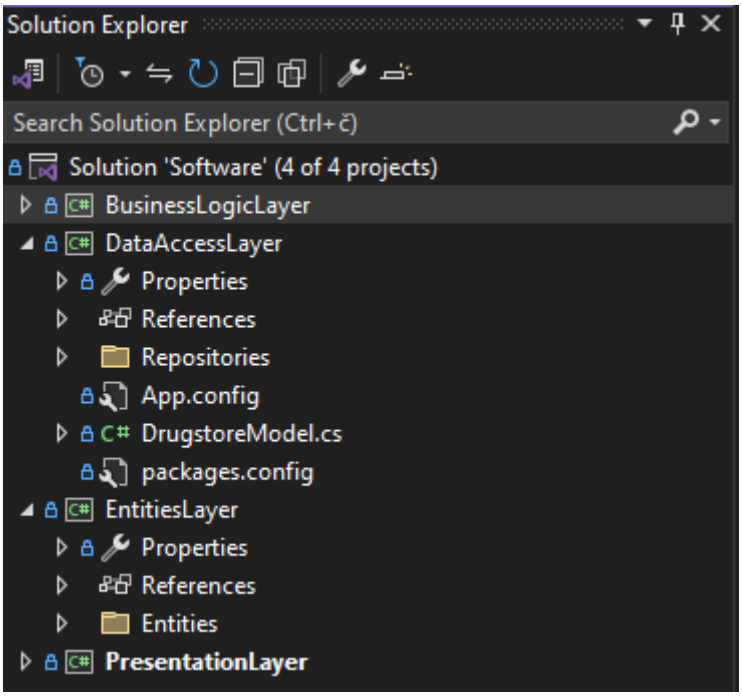
Slika 28: Odabir veze prema bazi



Slika 29: Odabir objekata baze uključenih u projekt

U prethodnom poglavlju je spomenuto kako aplikacija ima još sloj entiteta tako da sve entitetske klase koje su generirane kopiramo u projekt „EntitiesLayer“ u mapu „Entities“ tako da na kraju projekt ima strukturu prikazanu na slici 30. Kako bi aplikacija ispravno radila, moramo iz datoteke App.config projekta „DataAccessLayer“ kopirati „connection string“ u istoimenu datoteku „PresentationLayer“ projekta. Za konkretno ovaj lokalni server na računalu na kojem je rađen ovaj rad moramo kopirati sljedeće iz App.config:

```
<connectionStrings>
  <add       name="DrugstoreModel"       connectionString="data       source=DESKTOP-
   F3TLOOV\SQLEXPRESS;initial                              catalog=BAZA_LJEKARNA;user
   id=user;password=1234;trustservercertificate=True;MultipleActiveResultSets=True;App=
   EntityFramework" providerName="System.Data.SqlClient" />
</connectionStrings>
```



Slika 30: Struktura projekta

7.4. Implementacija Repository klasa u sloju pristupa podacima

U sloju pristupa podacima klase koje komuniciraju s bazom podataka zovu se Repository klase. U programskom rješenju napravljena je apstraktna klasa „Repository“ koju nasljeđuju ostale klase. Svaka klasa zadužena je za manipulaciju podacima nad svojom klasom. Kreirano je 12 klasa na sloju pristupe podacima, no neke ovdje neće biti obrađene budući da je njihova struktura dosta jednostavna i shvatljiva nakon objašnjavanja kompliciranijih. Prvo će biti objašnjena apstraktna klasa „Repository“ koju sve ostale nasljeđuju, a njezin kod nalazi se u nastavku:

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

    public Repository(DrugstoreModel 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 virtual int Add(T entity, bool saveChanges = true)
        {
            Entities.Add(entity);
            return saveChanges ? SaveChanges() : 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);

            return saveChanges ? SaveChanges() : 0;
        }

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

Klasa je generička jer ju nasljeđuju ostale klase koje koriste različite entitetske klase i implementira IDisposable sučelje kako bi se mogla zatvoriti veza prema bazi podataka dok se ne treba koristiti. Član klase Context predstavlja kontekst cijele baze podataka koristi se kako bi komunikacija s bazom bila moguća. Entities predstavlja skup entiteta s kojima klasa radi. Metode SaveChanges() i Dispose() koriste kako bi se mogle spremiti napravljene promjene nad entitetima i zatvorila veza prema bazi. U klasi su implementirane osnovne metode koje bi mogle biti potrebne za rad s podacima. Metoda GetAll() vraća sve zapise entiteta u bazi podataka, a za to koristi LINQ upit. Metoda Add() dodaje u bazu novi zapis entiteta i koristi argument saveChanges preko kojeg se može kontrolirati hoće li se odmah spremiti zapisi ili ne. Metoda Remove() briše entitet iz baze podataka na način da ga poveže u kontekst baze kako bi bio prepoznat i zatim obriše također je dodana mogućnost hoće li se odmah spremiti promjene ili ne. Metoda Update() nije implementirana budući da se svaki entitet može ažurirati na drugačiji način i ona je obavezna za implementirati u onim klasama koju će ovu klasu naslijediti.

7.4.1. FarmaceutRepository

```
public class FarmaceutRepository : Repository<Farmaceut>
{
    public FarmaceutRepository() : base(new DrugstoreModel())
    {
    }

    public override int Update(Farmaceut entity, bool saveChanges = true)
    {
        var pharmacist = Entities.First(x => x.ID == entity.ID);
        pharmacist.Ime = entity.Ime;
        pharmacist.Prezime = entity.Prezime;
        pharmacist.Email = entity.Email;
        pharmacist.Adresa = entity.Adresa;
        pharmacist.Lozinka = entity.Lozinka;
```

```
            return saveChanges ? SaveChanges() : 0;
        }
    }
```

Klasa FarmaceutRepository služi za dohvaćanje i ažuriranje podataka o farmaceutu. Može se primijetiti kako metode za dohvaćanje podataka nema iz razloga što je ona implementirana u klase Repository koja je naslijeđena te se ne treba ponovno pisati. Metoda Update() prima argument farmaceuta te ga prema ID-u vadi iz entiteta, mijenja mu svojstva te sprema promjene ako je saveChanges postavljen na true.

7.4.2. JedinicaMjereRepository

```
public class JedinicaMjereRepository : Repository<JedinicaMjere>
{
    public JedinicaMjereRepository() : base(new DrugstoreModel())
    {
    }

    public JedinicaMjere GetJedinicaMjereByID(string id)
    {
        var query = from e in Entities where e.ID == id select e;
        return query.FirstOrDefault();
    }

    public override int Update(JedinicaMjere entity, bool saveChanges = true)
    {
        var measureUnit = Entities.SingleOrDefault(m => m.ID == entity.ID);
        measureUnit.Naziv = entity.Naziv;

        return saveChanges ? SaveChanges() : 0;
    }
}
```

U klasi JedinicaMjereRepository dodana je metoda GetJedinicaMjereByID() koja dohvaća jedinicu mjere prema proslijeđenom ID-u. Metoda Update() slična je kao i u prethodnoj klasi samo što je ovdje implementirana na način da mijenja naziv jedinice mjere.

7.4.3. ArtiklRepository

```
public class ArtiklRepository : Repository<Artikl>
{
    public ArtiklRepository() : base(new DrugstoreModel())
    {
    }

    public override IQueryable<Artikl> GetAll()
    {
        return base.GetAll().Include("JedinicaMjere");
    }

    public override int Update(Artikl entity, bool saveChanges = true)
    {
        var item = Entities.SingleOrDefault(e => e.ID == entity.ID);
        item.Naziv = entity.Naziv;
        item.Kolicina = entity.Kolicina;
        item.Cijena = entity.Cijena;
        item.JedinicaMjereID = entity.JedinicaMjereID;
```

```
            return saveChanges ? SaveChanges() : 0;
        }

        public override int Remove(Artikl entity, bool saveChanges = true)
        {
            var item = Entities.Find(entity.ID);
            Entities.Remove(item);
            return saveChanges ? SaveChanges() : 0;
        }
    }
```

Klasa ArtiklRepository ima nadjačanu metodu GetAll() iz razloga što povezane objekte, koji su u ovom slučaju to jedinice mjere, uključuje u upit kako bi se iz svakog artikla moglo pristupiti njegovoj jedinici mjere i izvući podatke o istoj. Metoda Update() dohvaća artikl prema ID-u i mijenja naziv, količinu, cijenu i jedinicu mjere.

7.4.4.RacunRepository

```
    public class RacunRepository : Repository<Racun>
    {
        public RacunRepository() : base(new DrugstoreModel())
        {
        }
        public override IQueryable<Racun> GetAll()
        {
            return base.GetAll().Include(r => r.Farmaceut).Include(r => r.StavkeRacuna);
        }
        public override int Update(Racun entity, bool saveChanges = true)
        {
            throw new NotImplementedException();
        }

        public override int Add(Racun entity, bool saveChanges = true)
        {
            using (var model = new DrugstoreModel())
            {
                foreach (var item in entity.StavkeRacuna)
                {
                    model.Artikl.Attach(item.Artikl);
                    item.Artikl.Kolicina -= item.Kolicina;
                }
                model.Racun.Add(entity);

                return saveChanges ? model.SaveChanges() : 0;
            }
        }

        public ICollection<StavkeRacuna> GetInvoiceItems(int invoiceId)
        {
            using (var model = new DrugstoreModel())
            {
                var invoice = model.Racun.Include(n => n.StavkeRacuna).First(o => o.ID ==
invoiceId);
                foreach (var item in invoice.StavkeRacuna)
                {
                    model.Entry(item).Reference(i => i.Artikl).Load();
                }
                return invoice.StavkeRacuna;
            }
        }
    }
```

Klasa RacunRepository također ima nadjačanu metodu GetAll() i uključuje u upitu povezane entitete koji su farmaceut i stavke računa ovaj puta zapisano na drugačiji način. Metoda Add() je nadjačana i radi tako da koristi cijeli kontekst baze iz razloga što u istom bloku koda mora koristiti dva tipa entiteta, a to su račun i artikl. Funkcionira tako da iz svake stavke računa vadi

artikl te mu oduzima količinu na računu s količine na stanju i na kraju sprema račun u bazu. Metoda GetInvoiceItems() vraća stavke računa prema proslijeđenom ID-u te za potrebe grafičkog sučelja mora učitati i artikle koji su povezani na stavke, a to radi pomoću naredbe u foreach petlji. Slično je implementirana klasa PrimkaRepository, jedina razlika je ta što se kod primke povećava stanje na skladištu.

7.4.5.NarudzbaRepository

```
        public class NarudzbaRepository : Repository<Narudzba>
        {
            public NarudzbaRepository() : base(new DrugstoreModel())
            {
            }

            public override IQueryable<Narudzba> GetAll()
            {
                return
base.GetAll().Include("Dobavljac").Include("Farmaceut").Include("StatusNarudzbe").Include("Sta
vkeNarudzbe");
            }

            public override int Add(Narudzba entity, bool saveChanges = true)
            {
                using (var model = new DrugstoreModel())
                {
                    foreach (var item in entity.StavkeNarudzbe)
                    {
                        model.Artikl.Attach(item.Artikl);
                    }
                    model.Narudzba.Add(entity);

                    return saveChanges ? model.SaveChanges() : 0;
                }
            }

            public ICollection<StavkeNarudzbe> GetOrderItems(int orderId)
            {
                using (var model = new DrugstoreModel())
                {
                    var order = model.Narudzba.Include(n => n.StavkeNarudzbe).First(o => o.ID ==
orderId);
                    foreach (var item in order.StavkeNarudzbe)
                    {
                        model.Entry(item).Reference(i => i.Artikl).Load();
                    }
                    return order.StavkeNarudzbe;
                }
            }

            public override int Update(Narudzba entity, bool saveChanges = true)
            {
                using (var model = new DrugstoreModel())
                {
                    var        existingOrder        =        model.Narudzba.Include(o        =>
o.StavkeNarudzbe).FirstOrDefault(o => o.ID == entity.ID);

                    if (existingOrder == null) throw new Exception("Narudžba nije pronađena!");

                    model.Entry(existingOrder).CurrentValues.SetValues(entity);

                    var itemsToRemove = existingOrder.StavkeNarudzbe
                        .Where(existingItem     =>     !entity.StavkeNarudzbe.Any(newItem     =>
newItem.ArtiklID == existingItem.ArtiklID && newItem.NarudzbaID == existingItem.NarudzbaID))
                        .ToList();

                    foreach(var item in itemsToRemove) model.StavkeNarudzbe.Remove(item);

                    foreach (var orderItem in entity.StavkeNarudzbe)
```

```
                    {
                        var existingItem = existingOrder.StavkeNarudzbe
                            .SingleOrDefault(item => item.ArtiklID == orderItem.ArtiklID &&
item.NarudzbaID == orderItem.NarudzbaID);

                        if (existingItem != null)
model.Entry(existingItem).CurrentValues.SetValues(orderItem);
                        else
                        {
                            existingOrder.StavkeNarudzbe.Add(new StavkeNarudzbe
                            {
                                ArtiklID = orderItem.ArtiklID,
                                NarudzbaID = entity.ID,
                                Kolicina = orderItem.Kolicina,
                            });
                        }
                    }

                    return saveChanges ? model.SaveChanges() : 0;
                }
            }

            public override int Remove(Narudzba entity, bool saveChanges = true)
            {
                var order = Entities.Include(o => o.StavkeNarudzbe).FirstOrDefault(o => o.ID ==
entity.ID);
                Entities.Remove(order);

                return saveChanges ? SaveChanges() : 0;
            }
        }
```

Klasa NarudzbaRepository ima većinom sličan kod kao i RacunRepository, ali je ovdje uzeta kao primjer iz razloga što se narudžbi mogu uređivati stavke te je onda Update() metoda složenija. Prvo se dohvaća narudžba iz entiteta te se u listu itemsToRemove spremaju one stavke koje nisu sadržane u promijenjenoj narudžbi. Foreach petljom se prolazi po stavkama koje se trebaju ukloniti. U sljedećoj foreach petlji prolazi se po stavkama promijenjene narudžbe te se provjerava postoje li takve stavke u entitetima. Ukoliko takva stavka postoji, ona se postavlja na vrijednosti iste stavke promijenjene narudžbe, a ukoliko ne postoji dodaje se među ostale stavke postojeće narudžbe. Nakon ažuriranja postojeće narudžbe, spremaju se promjene.

7.5. Implementacija Service klasa u sloju poslovne logike

Na sloju poslovne logike nalaze se klase servisi koje provjeravaju ispravnost podataka i pozivaju repozitorije za obavljanje radnji nad podacima u bazi podataka. Sam programski kod u klasama sloja poslovne logike nije opširan budući da samo poziva metode određenog repozitorija kojeg koristi. Svaki servis pri provjeri podataka može baciti grešku ukoliko se utvrdi da su podaci neispravni. Greška se baca kako bi se krajnjem korisniku koji koristi aplikaciju ispisala greška čiji je tekst sadržan u iznimki. U nastavku su opisane implementacije iznimki kao i kodovi klasa servisa koji koriste prethodno obrađene repozitorije.

7.5.1.Iznimke

Kako bi se korisniku mogla prikazati poruka o pogrešci koja se dogodila na sloju poslovne logike na njemu se treba baciti iznimka. Ovisno o kojem servisu je riječ, odnosno za koji entitet je servis zadužen baca iznimku za taj entitet. Cijela stvar je jasnija kada se pogleda implementacija iznimki.

Potrebno je kreirati općenitu iznimku koju će sve ostale naslijediti. Budući da je riječ o ljekarni, općenitu iznimku imenovao sam „DrugstoreException“ koja nasljeđuje klasu ApplicationException, a služi da postavi poruku iznimke koju kasnije treba prikazati na prezentacijskom sloju. Kod općenito iznimke dan je u nastavku.

```
public class DrugstoreException : ApplicationException
{
    public string Message { get; set; }

    public DrugstoreException(string message)
    {
        Message = message;
    }
}
```

Sve ostale iznimke koje se bacaju u servisima nasljeđuju DrugstoreException klasu. Recimo ako FarmaceutServices treba baciti iznimku, treba postojati FarmaceutException klasa koja nema nikakvu dodatnu implementaciju nego samo nasljeđuje DrugstoreException. Kod FarmaceutException nalazi se u nastavku.

```
public class FarmaceutException : DrugstoreException
{
    public FarmaceutException(string message) : base(message)
    {
    }
}
```

Iznimka FarmaceutException koristi se u kodu na isti način kako je implementirana DrugstoreException jer njezin konstruktor poziva konstruktor DrugstoreException. Način na koji se obrađuju iznimke prikazan je u poglavlju koje opisuje prezentacijski sloj.

7.5.2.FarmaceutServices

```
public class FarmaceusServices
{
    public Farmaceut GetFarmaceutByKorime(string korime)
    {
        using(var repo = new FarmaceutRepository())
        {
            return repo.GetAll().ToList().Find(x => x.Korime == korime);
        }
    }

    public async Task<List<Farmaceut>> GetAll()
    {
```

```
            using(var repo = new FarmaceutRepository())
            {
                return await Task.Run(() => repo.GetAll().ToList());
            }
        }

        public async Task<bool> Update(Farmaceut farmaceut)
        {
            if (farmaceut.Ime.Length > 20 || farmaceut.Ime.Length == 0)
                throw new FarmaceutException("Ime ne smije biti prazno niti dulje od 20
znakova");
            if (farmaceut.Prezime.Length > 30 || farmaceut.Prezime.Length == 0)
                throw new FarmaceutException("Prezime ne smije biti prazno niti dulje od 30
znakova");
            if (farmaceut.Adresa.Length > 200 || farmaceut.Adresa.Length == 0)
                throw new FarmaceutException("Adresa ne smije biti prazna niti dulja od 200
znakova");
            if (!Regex.IsMatch(farmaceut.Email, @"^(?=.{1,50}$)[a-zA-Z0-9._]+@[a-zA-Z0-9.-
]+\.[a-zA-Z]{2,}$"))
                throw new FarmaceutException("Email nije ispravan!");
            if (farmaceut.Lozinka.Length < 6 || farmaceut.Lozinka.Length > 50)
                throw new FarmaceutException("Lozinka ne smije biti kraća od 6 ni dolja od
50 znakova");
            if (!Regex.IsMatch(farmaceut.Lozinka, @"^[a-zA-Z0-9!@#$%^&?_*]{6,50}$"))
                throw new FarmaceutException("Lozinka smije sadržavati samo slova, brojeve
i !@#$%^&?_*");

            using (var repo = new FarmaceutRepository())
            {
                int affectedRows = await Task.Run(() => repo.Update(farmaceut));
                return affectedRows > 0;
            }
        }
    }
```

Prve dvije metode klase FarmaceutServices dohvaćaju farmaceuta preko korisničkom imena i općenito sve farmaceute. Obje metode koriste repozitorij i samo pozivaju potrebnu metodu. Metoda GetAll() je asinkrona metoda koja se izvršava u zasebnoj dretvi kako korisničko sučelje može raditi bez prekida. Isto tako metoda Update() je asinkrona, no ona ima provjere na početku kako bi se ispitalo ispravnost podataka prilikom ažuriranja. Ukoliko dođe do pogrešno upisanog podatka, baca se iznimka.

7.5.3. JedinicaMjereServices

```
    public class JedinicaMjereServices
    {
        public async Task<JedinicaMjere> GetJedinicaMjereByID(string id)
        {
            using(var repo = new JedinicaMjereRepository())
            {
                return await Task.Run(() => repo.GetJedinicaMjereByID(id));
            }
        }

        public async Task<List<JedinicaMjere>> GetAll()
        {
            using(var repo = new JedinicaMjereRepository())
            {
                return await Task.Run(() =>  repo.GetAll().ToList());
            }
        }

        public async Task<bool> Add(JedinicaMjere measureUnit)
        {
            if (measureUnit.ID.Length == 0 || measureUnit.ID.Length > 3)
```

```
                throw new JedinicaMjereException("Duljina ID-a ne smije biti prazna ni veća
od 3 znaka!");
            if (measureUnit.Naziv.Length == 0 || measureUnit.Naziv.Length > 10)
                throw new JedinicaMjereException("Naziv ne smije biti prazan ni veći od 10
znakova!");
            using (var repo = new JedinicaMjereRepository())
            {
                int affectedRows = await Task.Run(() => repo.Add(measureUnit));
                return affectedRows > 0;
            }
        }

        public async Task<bool> Update(JedinicaMjere measureUnit)
        {
            if (measureUnit.Naziv.Length == 0 || measureUnit.Naziv.Length > 10)
                throw new JedinicaMjereException("Naziv ne smije biti prazan ni veći od 10
znakova!");
            using (var repo = new JedinicaMjereRepository())
            {
                int affectedRows = await Task.Run(() => repo.Update(measureUnit));
                return affectedRows > 0;
            }
        }

        public async Task<bool> Remove(JedinicaMjere measureUnit)
        {
            using (var repo = new JedinicaMjereRepository())
            {
                int affectedRows = await Task.Run(() => repo.Remove(measureUnit));
                return affectedRows > 0;
            }
        }
    }
```

Klasa JedinicaMjereServices ima metode GetJedinicaMjereByID(), GetAll() i Remove() koje nemaju nikakvu provjeru, nego samo pozivaju repozitorij s istoimenim metodama. Metoda Add() provjerava duljinu naziva jedinice mjere i duljinu ID-a. Ukoliko naziv ili ID ne zadovoljavaju kriterije, baca se JedinicaMjereException. Metoda Update() koristi se samo za promjenu naziva jedinice mjere koji se također provjerava.

7.5.4. ArtiklServices

```
    public class ArtiklServices
    {
        public async Task<List<Artikl>> GetAll()
        {
            using(var repo = new ArtiklRepository())
            {
                return await Task.Run(() =>  repo.GetAll().ToList());
            }
        }

        public async Task<bool> Add(Artikl artikl)
        {
            ValidateItem(artikl);

            using (var repo = new ArtiklRepository())
            {
                int affectedRows = await Task.Run(() => repo.Add(artikl));
                return affectedRows > 0;
            }
        }

        private static void ValidateItem(Artikl artikl)
        {
            if (artikl.Naziv.Length == 0 || artikl.Naziv.Length > 45)
```

```
                throw new ArtiklException("Naziv mora imati do 45 znakova!");
            if (artikl.Cijena < 0)
                throw new ArtiklException("Cijena ne smije biti negativna!");
            if (artikl.Kolicina < 0)
                throw new ArtiklException("Količina ne smije biti negativna!");
        }

        public async Task<bool> Remove(Artikl artikl)
        {
            using(var repo = new ArtiklRepository())
            {
                int affectedRows = await Task.Run(() => repo.Remove(artikl));
                return affectedRows > 0;
            }
        }

        public async Task<bool> Update(Artikl artikl)
        {
            ValidateItem(artikl);

            using (var repo = new ArtiklRepository())
            {
                int affectedRows = await Task.Run(() => repo.Update(artikl));
                return affectedRows > 0;
            }
        }
    }
```

ArtiklServices klasa ima metode GetAll() i Remove() koje samo pozivaju metode repozitorija. Metode Add() i Update() prije poziva repozitorija provjeravaju ispravnost proslijeđenog artikla na isti način tako da je napravljena zasebna metoda koja radi provjeru i poziva se u metodama. Iznimke se bacaju u metodi ValidateItem() koji provjerava naziv, cijenu i količinu artikla.

7.5.5.RacunServices

```
    public class RacunServices
    {
        public async Task<List<Racun>> GetALl()
        {
            using(var repo = new RacunRepository())
            {
                return await Task.Run(() => repo.GetAll().ToList());
            }
        }

        public async Task<bool> Add(Racun racun)
        {
            if (racun.StavkeRacuna.Count == 0)
                throw new RacunException("Morate imati najmanje jednu stavku u računu!");

            foreach(var item in racun.StavkeRacuna)
            {
                int amount = item.Artikl.Kolicina - item.Kolicina;
                if (amount < 0)
                    throw    new    RacunException($"Nedostaju    vam    {Math.Abs(amount)}
{item.Artikl}!");
            }

            using (var repo = new RacunRepository())
            {
                int affectedRows = await Task.Run(() => repo.Add(racun));
                return affectedRows > 0;
            }
        }

        public async Task<List<StavkeRacuna>> GetInvoiceItems(Racun racun)
        {
            using(var repo = new RacunRepository())
```

```
            {
                return await Task.Run(() => repo.GetInvoiceItems(racun.ID).ToList());
            }
        }
    }
```

Klasa RacunServices ima metode GetAll() i GetInvoiceItems() koje nemaju nikakvu validaciju, nego samo koriste metode repozitorija. Kada se dodaje novi račun osim provjere ima li račun stavke dodana je i provjera jesu li svi artikli koji su na računu dostupni. U foreach petlji prolazi se kroz svaku stavku računa i izračunava se stanje svakog artikla prije nego što se izda račun. Ukoliko se provjerom ustanovi da je stanje artikla manje od 0, korisniku se baca greška s informacijom koji artikl nedostaje da se izda račun i koja količina tog artikla nedostaje.

7.5.6. NarudzbaServices

```
    public class NarudzbaServices
    {
        public async Task<List<Narudzba>> GetAll()
        {
            using (var repo = new NarudzbaRepository())
            {
                return await Task.Run(() => repo.GetAll().ToList());
            }
        }

        public async Task<bool> Add(Narudzba narudzba)
        {
            if (narudzba.StavkeNarudzbe.Count == 0)
                throw new NarudzbaException("Morate imati najmanje jednu stavku u
narudžbi!");

            using(var repo = new NarudzbaRepository())
            {
                int affectedRows = await Task.Run(() => repo.Add(narudzba));
                return affectedRows > 0;
            }
        }

        public async Task<List<StavkeNarudzbe>> GetOrderItems(Narudzba narudzba)
        {
            using(var repo = new NarudzbaRepository())
            {
                return await Task.Run(() => repo.GetOrderItems(narudzba.ID).ToList());
            }
        }

        public async Task<bool> Update(Narudzba order)
        {
            if (order.StavkeNarudzbe.Count == 0)
                throw new NarudzbaException("Morate imati najmanje jednu stavku u
narudžbi!");

            using (var repo = new NarudzbaRepository())
            {
                int affectedRows = await Task.Run(() => repo.Update(order));
                return affectedRows > 0;
            }
        }

        public async Task<bool> Remove(Narudzba order)
        {
            if (order.Status == StatusNarudzbeEnum.Zakljucena)
                throw new NarudzbaException("Ne možete obrisati zaključenu narudžbu!");

            using(var repo = new NarudzbaRepository())
```

```
            {
                int affectedRows = await Task.Run(() => repo.Remove(order));
                return affectedRows > 0;
            }
        }
    }
```

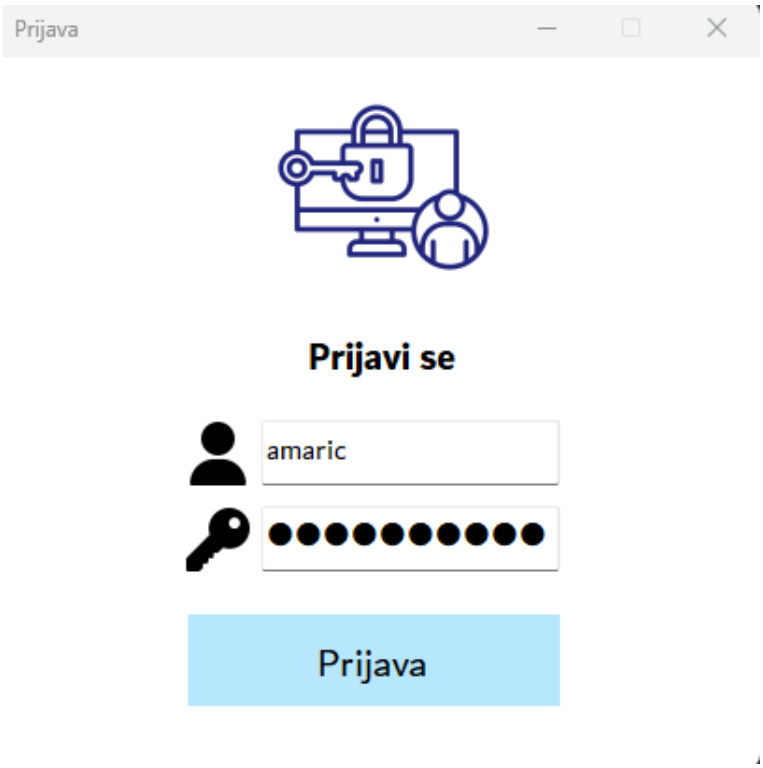
Klasa NarudzbaServices implementira metode GetAll(), GetOrderItems() koje ne moraju imati nikakvu validaciju podataka. Metode Add() i Update() provjeravaju broj stavki narudžbe jer narudžba nema smisla postojati ako nema nijednu stavku te ukoliko se desi takva situacija baca se iznimka. Metoda Remove() ovdje ima provjeru je li narudžba zaključena jer se takve narudžbe ne bi smjele obrisati u aplikaciji.

7.6. Prezentacijski sloj i grafičko sučelje

Prezentacijski sloj aplikacije sadrži Windows forme preko kojih je implementirano grafičko sučelje aplikacije. Na svakoj formi korišteni su razni tipovi elemenata kako bi se mogli korisniku ispravno prikazati podaci dohvaćeni iz baze podataka. Svaka forma ima svoj dizajn i programsku logiku iza pozadine. U ovom poglavlju prikazane su forme aplikacije kako izgledaju kada je aplikacija pokrenuta kada prikazuju podatke iz baze i objašnjena programska logika koja se nalazi iza grafičkog sučelja. Radi kompleksnosti aplikacije i brojnosti formi korisničkog sučelja, prikazane će biti samo određene forme koje koriste neke od servisa opisanih u prijašnjem poglavlju. Sve forme i cijeli kod aplikacije može se pogledati na GitHub repozitoriju čija se poveznica nalazi u poglavlju „Aplikacija“.

7.6.1. Login forma

Forma za prijavu ima jednostavno grafičko sučelje na kojem su bitni elementi polja za upis korisničkog imena i lozinke te gumb za prijavu (Slika 31). Ukoliko korisničko ime ili lozinka nisu točni, korisniku se prikazuje poruka s greškom. Ako su korisničko ime i lozinka ispravni, korisnika se pušta u aplikaciju.



Slika 31: Froma za prijavu

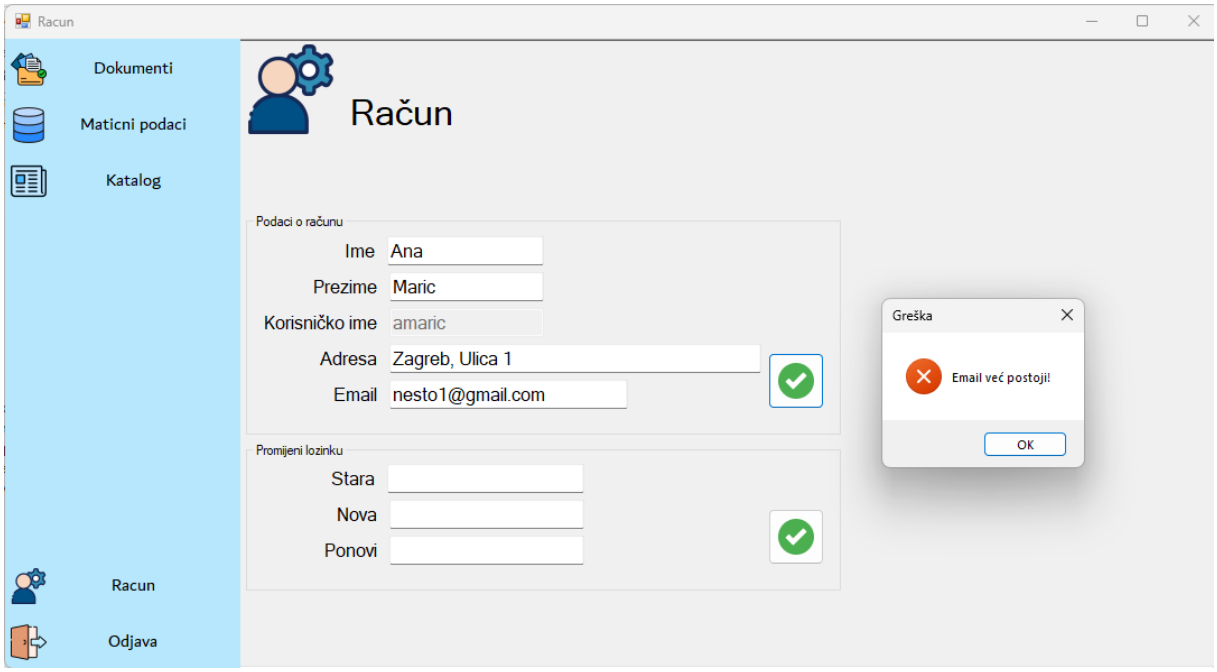
Programski kod forme za prijavu koristi metode koje se pozivaju na određeni događaj, tako na klik gumba se poziva metoda button1_Click() koji služi za promjenu forme. Najbitnija metoda je LoginUser() koja iz polja za unos korisničkog imena i lozinke provjerava ispravnost podataka na način da poziva metodu GetFarmaceutByKorime() klase servisa koja je zadužena za entitet farmaceuta. Ukoliko korisnik nije pronađen znači da je null i ispisuje se poruka kako korisničko ime ili lozinka nisu ispravni, isti je slučaj kada korisnik postoji, ali lozinka nije ispravna. Ukoliko su podaci za prijavu ispravni, otvara se glavna forma aplikacije.

```
public partial class Login : Form
{
    private FarmaceusServices farmaceusServices = new FarmaceusServices();
    public static Farmaceut User = null;
    public Login()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)
    {
        ChangeForm();
    }

    private void ChangeForm()
    {
        LoginUser();
        if (User != null) OpenMainForm();
    }

    private void OpenMainForm()
    {
        FrmMain frmMain = new FrmMain();
        Hide();
```

```
                frmMain.ShowDialog();
                User = null;
                txtPassword.Text = "";
                txtUsername.Text = "";
                Show();
            }

            private void LoginUser()
            {
                string username = txtUsername.Text;
                string password = txtPassword.Text;
                var farmaceut = farmaceusServices.GetFarmaceutByKorime(username);

                if (farmaceut == null)
                {
                    MessageBox.Show("Krivo    korisničko    ime    ili    lozinka",    "Greška",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }

                if (farmaceut.Lozinka == password)
                {
                    User = farmaceut;
                }
                else
                {
                    MessageBox.Show("Krivo    korisničko    ime    ili    lozinka",    "Greška",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }

            private void txtUsername_KeyDown(object sender, KeyEventArgs e)
            {
                if (e.KeyCode == Keys.Enter) ChangeForm();
            }

            private void txtPassword_KeyDown(object sender, KeyEventArgs e)
            {
                if (e.KeyCode == Keys.Enter) ChangeForm();
            }
        }
```

7.6.2.Korisnički račun

Forma korisničkog računa prikazuje korisnikove osobne podatke i omogućuje uređivanje istih. Promjena korisničkih podataka podijeljena je u dva dijela. Prvi dio omogućuje prikaz i promjenu osobnih podataka i kontakt emaila s time da korisnik ne može promijeniti korisničko ime. Druga kategorija je promjena lozinke prilikom čega korisnik mora upisati staru lozinku i dva puta novu. Ukoliko se proba napraviti promjena koja izaziva određenu grešku, na primjer na sloju baze podataka, greška se hvata i prikazuje u obliku upozorenja korisniku kao što je prikazano na slici 32 koja prikazuje promjenu emaila koji već postoji u bazi.



Slika 32: Forma promjene podataka s greškom za email

U programskoj logici forme prilikom učitavanja dohvaća se spremljeni korisnik sa statičnog člana klase Login forme čiji je kod prikazan u prethodnom opisu. Najbitnija metoda klase je UpdatePharmacist() koja preko servisa poziva metodu Update() i prosljeđuje joj objekt farmaceuta sa izmjenama koje je korisnik napravio preko korisničkog sučelja. Poziv metode servisa nalazi se u try bloku jer se u tom dijelu koda može desiti iznimka. Catch blokovi hvataju iznimku ovisno o tipu iznimke koja se izazvala. Prvi catch blok hvata FarmaceutException koja se baca na sloju poslovne logike i ispisuje njenu poruku. Drugi catch blok provjerava ako se desila iznimka DbUpdateException prilikom spremanja podataka u bazi. Preko uvjeta provjere je li greška SqlException sadržana, dobiva se točna greška iz baze podataka koja sadrži naziv greške u poruci. Ovisno koja iznimka se desila, ispisuje se poruka koju postavlja metoda GetErrorMessage(). Ako greška sadrži prekršaj UNIQUE ograničenja na farmaceutov email, vraća prikladnu poruku korisniku. Programski kod logike forme prikazana je u nastavku.

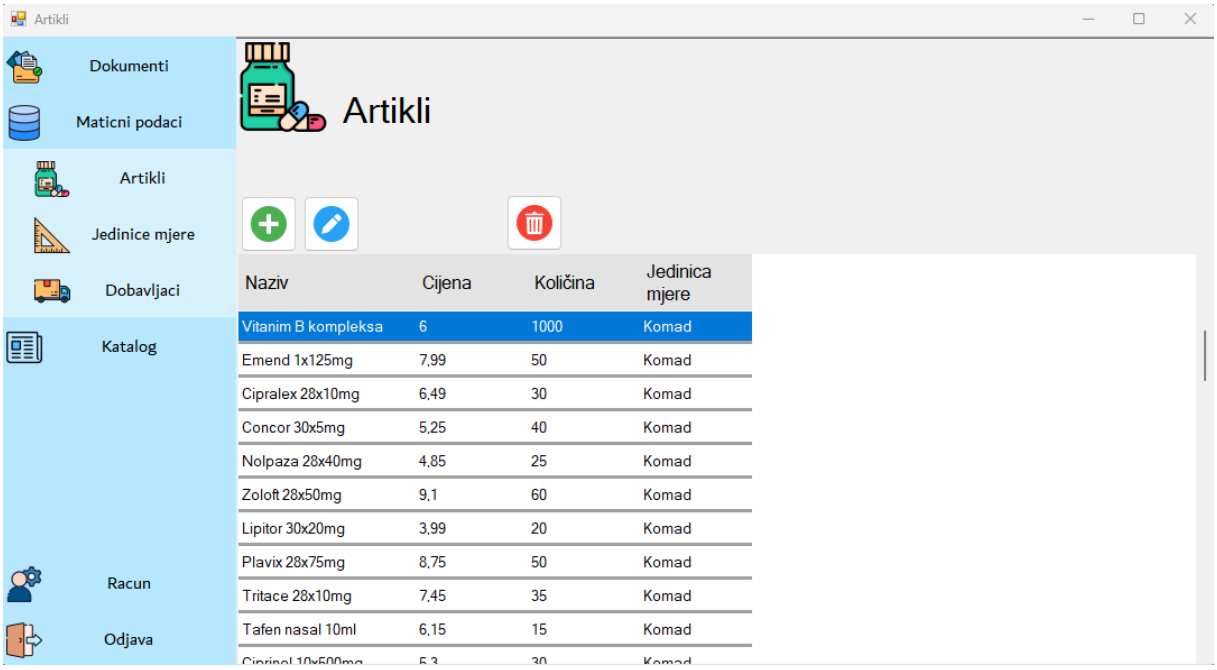
```
public partial class Account : Form
{
    private FarmaceusServices farmaceusServices = new FarmaceusServices();
    public Account()
    {
        InitializeComponent();
    }

    private void Account_Load(object sender, EventArgs e)
    {
        var pharmacist = Login.User;
        txtAddress.Text = pharmacist.Adresa;
        txtEmail.Text = pharmacist.Email;
        txtName.Text = pharmacist.Ime;
        txtSurname.Text = pharmacist.Prezime;
        txtUsername.Text = pharmacist.Korime;
    }

    private Farmaceut CreatePharmacist(Farmaceut pharmacist)
    {
```

```
            return new Farmaceut
            {
                ID = pharmacist.ID,
                Adresa = pharmacist.Adresa,
                Ime = pharmacist.Ime,
                Email = pharmacist.Email,
                Korime = pharmacist.Korime,
                Lozinka = pharmacist.Lozinka,
                Prezime = pharmacist.Prezime,
            };
        }

        private async void btnSave_Click(object sender, EventArgs e)
        {
            var pharmacist = CreatePharmacist(Login.User);
            pharmacist.Adresa = txtAddress.Text.Trim();
            pharmacist.Ime = txtName.Text.Trim();
            pharmacist.Prezime = txtSurname.Text.Trim();
            pharmacist.Email = txtEmail.Text.Trim();

            await UpdatePharmacist(pharmacist);
        }

        private async Task UpdatePharmacist(Farmaceut pharmacist)
        {
            try
            {
                await farmaceusServices.Update(pharmacist);
                Login.User = pharmacist;
                txtOld.Text = "";
                txtNew.Text = "";
                txtRepeat.Text = "";
                MessageBox.Show("Podaci       uspješno       ažurirani!",       "Uspjeh",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            } catch (FarmaceutException ex)
            {
                MessageBox.Show(ex.Message,         "Greška",         MessageBoxButtons.OK,
MessageBoxIcon.Error);
            } catch (DbUpdateException ex)
            {
                if (ex.InnerException?.InnerException is SqlException sqlEx)
                {
                    string message = GetErrorMessage(sqlEx);
                    MessageBox.Show(message,         "Greška",         MessageBoxButtons.OK,
MessageBoxIcon.Error);
                } else MessageBox.Show(GetErrorMessage(ex), "Greška", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            } catch (Exception ex)
            {
                MessageBox.Show("Neočekivana   greška",   "Greška",   MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        private string GetErrorMessage(Exception sqlEx)
        {
            string message = sqlEx.Message;
            if (message.Contains("UNIQUE_Farmaceut_Email")) return "Email već postoji!";
            return message;
        }

        private async void btnChangePassword_Click(object sender, EventArgs e)
        {
            var pharmacist = CreatePharmacist(Login.User);
            string oldPassword = txtOld.Text;
            string newPassword = txtNew.Text;
            string repeated = txtRepeat.Text;

            if (oldPassword != pharmacist.Lozinka) MessageBox.Show("Stara lozinka nije
ispravna!", "Greška", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else if (newPassword != repeated) MessageBox.Show("Nove lozinke se ne
podudaraju!", "Greška", MessageBoxButtons.OK, MessageBoxIcon.Error);
            else
            {
                pharmacist.Lozinka = newPassword;
                await UpdatePharmacist(pharmacist);
            }
```

```
        }
    }
```

7.6.3. Artikli

U grupi formi „Matični podaci“ nalazi se forma za artikle na kojoj je moguće uređivanje, dodavanje i brisanje artikala (Slika 33). Korisnik može obrisati artikl označavanjem artikla na listi te klikom na gumb s ikonicom kante za smeće. Ovisno o tome želi li se dodati artikl ili urediti postojeći, klikne se na gumb s oznakom plusa ili olovke. Dodavanje ili uređivanje artikla otvara novu formu u kojoj se upisuju ili izmjenjuju podaci o artiklima.



Slika 33: Forma za upravljanje artiklima

U programskom kodu prilikom učitavanja artikla, preko servisa se dohvaćaju svi artikli iz baze te se prilagođava prikaz artikla skrivanjem nepotrebnih podataka od korisnika kao što su povezane stavke primke na artikl ili preimenovanje zaglavlja stupca. Vidljive su metode za uređivanje ili dodavanje artikla koje pozivaju formu za prikaz detalja ili upis ovisno o tome koji je gumb pritisnut te se tako prosljeđuje objekt artikla koji se uređuje ili se ne prosljeđuje ništa ako se dodaje novi artikl. Prilikom brisanja kod se stavlja u try blok jer se može dogoditi greška u bazi ako označeni artikl postoji u stavkama nekog od dokumenata, tada se baca iznimka koja se ulovi i prema njezinom sadržaju korisniku se prikazuje prikladna poruka.

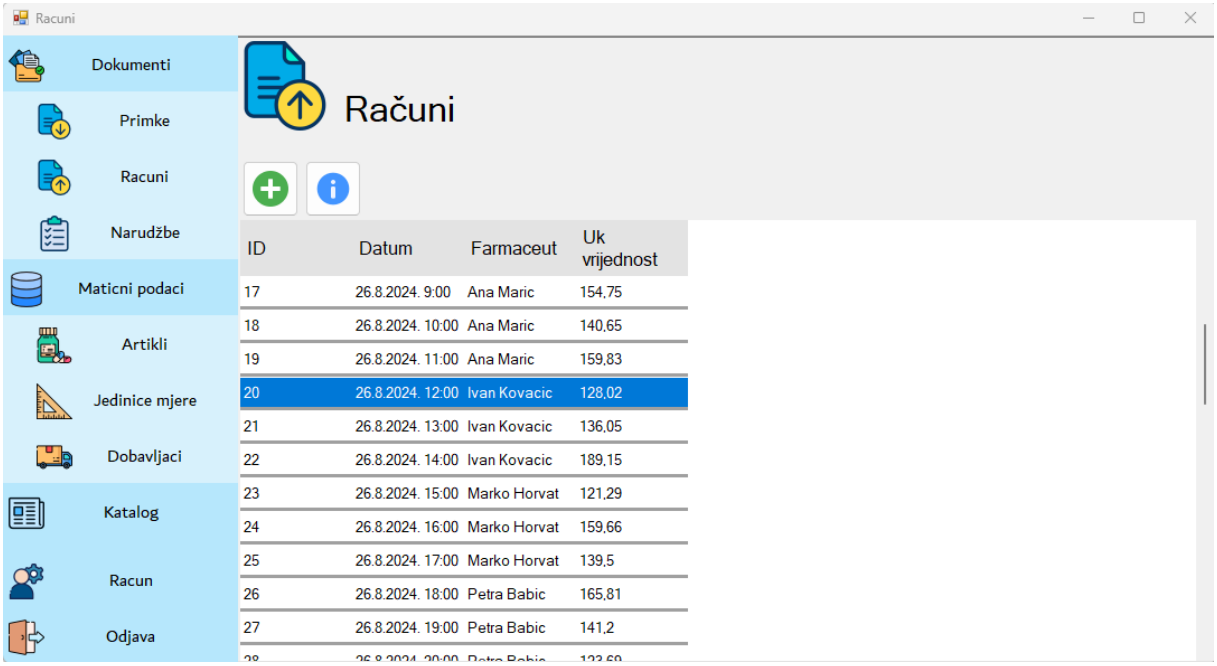
```
public partial class Items : Form
{
    private ArtiklServices artiklServices = new ArtiklServices();
    public Items()
```

```
        {
            InitializeComponent();
        }

        private async void Items_Load(object sender, EventArgs e)
        {
            await RefreshGUI();
        }

        private async Task RefreshGUI()
        {
            dgvItems.DataSource = await artiklServices.GetAll();

            dgvItems.Columns["ID"].Visible = false;
            dgvItems.Columns["JedinicaMjereID"].Visible = false;
            dgvItems.Columns["StavkeNarudzbe"].Visible = false;
            dgvItems.Columns["StavkePrimke"].Visible = false;
            dgvItems.Columns["StavkeRacuna"].Visible = false;
            dgvItems.Columns["StavkeRecepta"].Visible = false;

            dgvItems.Columns["JedinicaMjere"].HeaderText = "Jedinica mjere";
            dgvItems.Columns["Kolicina"].HeaderText = "Količina";
        }

        private async void btnEdit_Click(object sender, EventArgs e)
        {
            var item = dgvItems.CurrentRow?.DataBoundItem as Artikl;
            if (item != null)
            {
                ItemDetails itemDetails = new ItemDetails(item);
                itemDetails.ShowDialog();
                await RefreshGUI();
            }
        }

        private async void btnAdd_Click(object sender, EventArgs e)
        {
            ItemDetails itemDetails = new ItemDetails();
            itemDetails.ShowDialog();
            await RefreshGUI();
        }

        private async void btnDelete_Click(object sender, EventArgs e)
        {
            var item = dgvItems.CurrentRow?.DataBoundItem as Artikl;
            if (item != null)
            {
                try
                {
                    await artiklServices.Remove(item);
                } catch (DbUpdateException ex)
                {
                    if (ex.InnerException?.InnerException is SqlException sqlEx)
                    {
                        string message = GetErrorMessage(sqlEx);
                        MessageBox.Show(message,      "Greška",      MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    }       else       MessageBox.Show(GetErrorMessage(ex),       "Greška",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                } catch (Exception ex)
                {
                    MessageBox.Show("Neočekivana    greška"    +    ex.Message,    "Greška",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                await RefreshGUI();
            }
        }

        private string GetErrorMessage(Exception sqlEx)
        {
            string message = sqlEx.Message;
            if (message.Contains("unexpected number of rows (0)")) return "Ovaj artikl ne
postoji!";
            if  (message.Contains("FK_StavkeRecepta_Artikl"))  return  "Artikl  se  nalazi  u
receptima!";
            if  (message.Contains("FK_StavkeNarudzbe_Artikl"))  return  "Artikl  se  nalazi  u
narudžbama!";
```

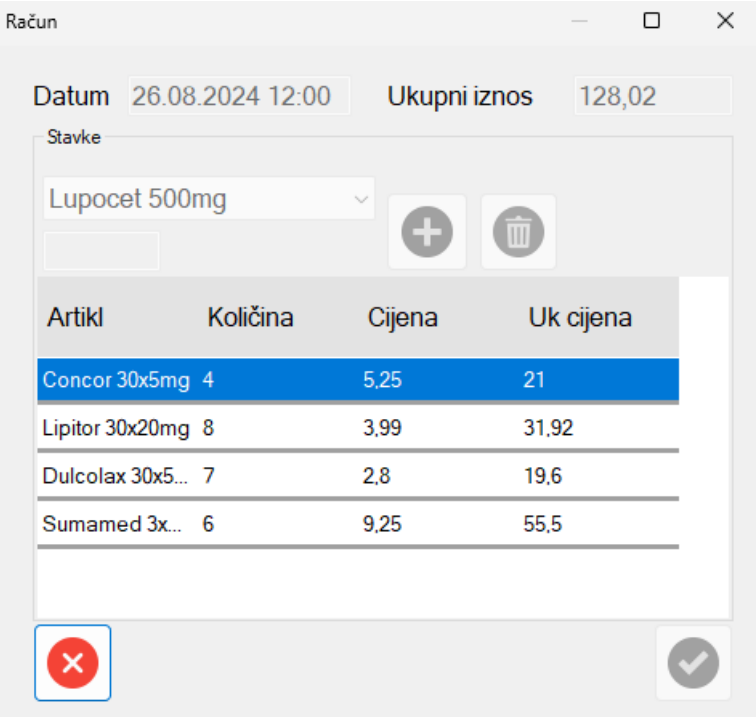
```
                if (message.Contains("FK_StavkePrimke_Artikl")) return "Artikl se nalazi u primkama!";
                if (message.Contains("FK_StavkeRacuna_Artikl")) return "Artikl se nalazi u računima!";
                return message;
            }
        }
```

7.6.4.Računi

Forma za račune nalazi se u grupaciji „Dokumenti“, a omogućuje pregled računa i njegovih stavki te kreiranje računa (Slika 34). Kreiranjem računa stanje artikala se smanjuje, a brisanje računa iz sustava nije predviđeno kao niti izmjena. U programskom kodu koji je prikazan ispod slika dohvaćaju se svi računi iz baze koji se moraju posebno obraditi kako bi se mogla prikazati ukupna vrijednost računa. To se odvija tako da se za svaki račun učitaju stavke i za svaku stavku računa se zbraja vrijednost dodanih artikala. Kada se skupe svi potrebni podaci radi se lista računa u novoj klasi RacunModelView koja je napravljena specijalno za prikaz računa u listi na korisničkom sučelju. U toj klasi se i sprema cijeli objekt računa kako bi se pregledavanjem računa iz liste isti mogao dohvatiti i kasnije na formi prikazati njegove stavke (Slika 35).



Slika 34: Forma računi

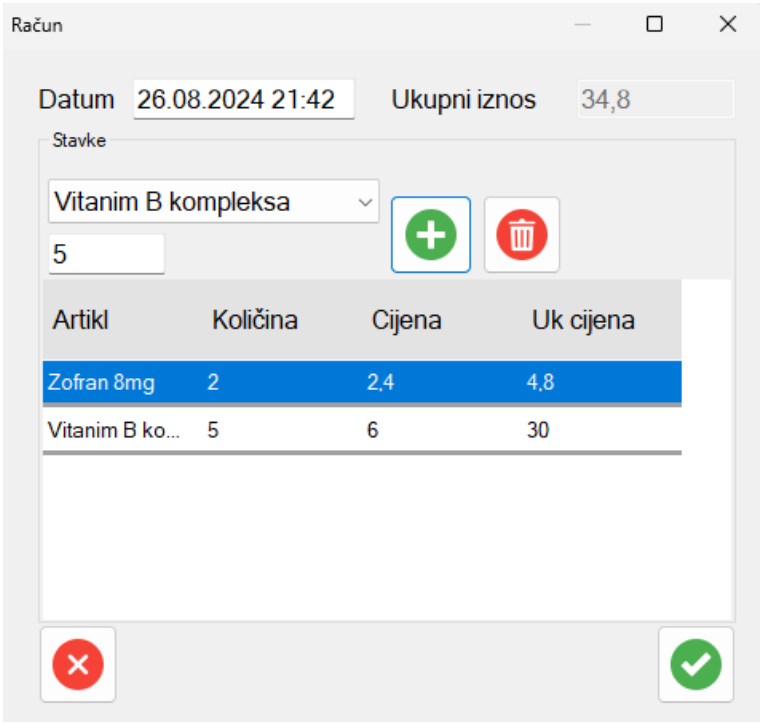


Slika 35: Prikaz detalja računa

```
public partial class Invoices : Form
{
    private RacunServices racunServices = new RacunServices();
    public Invoices()
    {
        InitializeComponent();
    }

    private async void Invoices_Load(object sender, EventArgs e)
    {
        await RefreshGUI();
    }

    private async Task RefreshGUI()
    {
        var invoicesView = new List<RacunViewModel>();
        var invoices = await racunServices.GetALl();
        foreach(var item in invoices)
        {
            item.StavkeRacuna = await racunServices.GetInvoiceItems(item);
            double sum = item.StavkeRacuna.Sum(x => x.Kolicina * x.Artikl.Cijena);
            invoicesView.Add(new RacunViewModel
            {
                ID = item.ID,
                Farmaceut = item.Farmaceut,
                Datum = item.Datum,
                UkupnaVrijednost = sum.ToString(),
                Racun = item
            });
        }

        dgvInvoices.DataSource = invoicesView;
        dgvInvoices.Columns["UkupnaVrijednost"].HeaderText = "Uk vrijednost";
        dgvInvoices.Columns["Racun"].Visible = false;
    }

    private async void btnAdd_Click(object sender, EventArgs e)
    {
        InvoiceDetails invoiceDetails = new InvoiceDetails();
        invoiceDetails.ShowDialog();
```

```
            await RefreshGUI();
        }

        private async void btnDetails_Click(object sender, EventArgs e)
        {
            var invoiceView = dgvInvoices.CurrentRow?.DataBoundItem as RacunViewModel;
            if (invoiceView != null)
            {
                var invoice = invoiceView.Racun;
                InvoiceDetails invoiceDetails = new InvoiceDetails(invoice);
                invoiceDetails.ShowDialog();
                await RefreshGUI();
            }
        }
    }
```

Forma za dodavanje računa sadrži se od elementa za unos datuma i skupa elemenata koji omogućuju dodavanje stavki na račun (Slika 36). U padajućem izborniku odabire se artikl koji se želi dodati na račun i upisuje se količina, da bi se stavka dodala potrebno je kliknuti na gumb sa simbolom plusa. Ako se stavka računa želi ukloniti, pritisne se gumb sa simbolom kante za smeće. Pri dodavanju i uklanjanju stavki, prikazuje se vrijednost svake stavke računa, kao i ukupni iznos računa. U programskom kodu klase forme ostavio sam samo najbitnije funkcije prilikom kreiranja računa budući da sadrži puno linija koda. Metoda btnAdd_Click() provjerava ispravnost količine artikla u stavki te poziva metodu AddItem(). U metodi se dohvaća varijabla računa spremljena u memoriji programa i čitaju se stavke. Ukoliko takva stavka već postoji samo se promijeni količina, ukoliko ne postoji, dodaje se u stavke računa. Brisanje stavki se odvija na sličan način, ali se koristi filtriranje stavki tako da se iz liste stavki izbaci stavka koja se želi ukloniti.



Slika 36: Forma za kreiranje računa

```
        public partial class InvoiceDetails : Form
        {
            private void btnAdd_Click(object sender, EventArgs e)
            {
                var item = cmbItems.SelectedItem as Artikl;
                int amount;
                bool isConverted = int.TryParse(txtAmount.Text, out amount);
                if   (!isConverted)   MessageBox.Show("Broj   mora   biti   cijeli!",   "Unos",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                else  if  (amount  <  1)  MessageBox.Show("Broj  mora  biti  veći  od  0!",  "Unos",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                else AddItem(item, amount);
            }

            private void AddItem(Artikl item, int amount)
            {
                var invoiceItem = new StavkeRacuna { ArtiklID = item.ID, Artikl = item, Kolicina
= amount };
                var invoiceItems = _invoice.StavkeRacuna.ToList();
                int indexOfItem = invoiceItems.IndexOf(invoiceItem);
                if (indexOfItem == -1) invoiceItems.Add(invoiceItem);
                else invoiceItems[indexOfItem] = invoiceItem;
                _invoice.StavkeRacuna = invoiceItems;
                RefreshGUI();
            }

            private void RefreshGUI()
            {
                var   invoiceItemsViewModel   =   _invoice.StavkeRacuna.Select(item   =>   new
StavkeRacunaViewModel
                {
                    Artikl = item.Artikl,
                    Cijena = item.Artikl.Cijena,
                    Kolicina = item.Kolicina,
                    StavkaRacuna = item
                }).ToList();

                double sum = 0;
                invoiceItemsViewModel.ForEach(x => sum += double.Parse(x.UkupnaCijena));
                txtSum.Text = sum.ToString();
                dgvInvoiceItems.DataSource = invoiceItemsViewModel;

                dgvInvoiceItems.Columns["StavkaRacuna"].Visible = false;
                dgvInvoiceItems.Columns["UkupnaCijena"].HeaderText = "Uk cijena";
                dgvInvoiceItems.Columns["Kolicina"].HeaderText = "Količina";
            }

            private void btnDelete_Click(object sender, EventArgs e)
            {
                var      itemView      =      dgvInvoiceItems.CurrentRow?.DataBoundItem      as
StavkeRacunaViewModel;
                if (itemView != null)
                {
                    var item = itemView.Artikl;
                    var invoiceItems = _invoice.StavkeRacuna.ToList();
                    invoiceItems = invoiceItems.FindAll(x => x.ArtiklID != item.ID);
                    _invoice.StavkeRacuna = invoiceItems;
                    RefreshGUI();
                }
            }
        }
```

8. Zaključak

Teorijskim dijelom rada predstavljene su relacijske baze podataka kao vrsta baza podataka koja koristi standardan jezik SQL kojim se na konzistentan način dohvaćaju podaci iz baze. Relacijske baze podataka čuvaju integritet podataka i u svakom trenutku je osigurano da su podaci u željenom stanju bilo da se radi o ažuriranju, brisanju ili dodavanu podataka. Mana relacijskih baza podataka je što imaju krutu strukturu jer jesnom kad se napravi struktura baze podataka teško ju je kasnije modificirati u budućnosti ukoliko se rade veće promjene nad relacijama. Praktičnim dijelom modeliranja baze podataka prikazano je kako je metodama analize i prepoznavanja tipova podataka kao i povezanosti poslovnih dokumenata i pravila moguće konstruirati bazu podataka za određenu domenu, konkretno za domenu ljekarne. Alatima SQL Server Management Studio i SQL Server pokazano je kako je moguće implementirati bazu podataka prema zahtjevima poslovanja.

Koristeći .Net platformu napravljena je aplikacija za korištenje baze podataka, prikazana je implementacija troslojne arhitekture aplikacije za Windows operacijski sustav napravljene pomoću tehnologije Windows Forms uz pomoć koje možemo napraviti kvalitetno korisničko sučelje kombinacijom elemenata grafičkog sučelja koje nudi. Ovim radom pokazano je da se podacima u bazi podataka može pristupiti i programski generirajući kontekst baze i klasa kao entiteta pomoću Entity Frameworka gdje pristup podacima ne izgleda kao standardni način preko čistih SQL upita već na način kao da iz objekata klase dohvaća svojstva. Uređivanje, promjena i brisanje podataka odvija se na način kao da mijenjamo vrijednosti programske varijable, a zapravo se promjene napravljene na takav način izražavaju u bazi podataka.

Izrađena baza podataka i aplikacija za domenu ljekarne pokriva osnovne zahtjeve poslovanja ljekarne i pokazuje površinsku kompleksnost takvog sustava. Implementacija baze podataka i programa za stvarnu upotrebu u ljekarni puno je kompleksnija od ove prikazane u radu. U aplikaciji bi morali biti implementirani korisnički zahtjevi koji bi zahtijevali složeniju strukturu aplikacije koja bi morala komunicirati s vanjskim servisima ili bazama podataka ostalih podružnica ljekarne. Konkretni nedostatak aplikacije implementirane za potrebe ovog završnog rada je što ne podržava istovremeni rad više korisnika jer direktno šalje promjene na bazu bez izolacija podataka, na primjer pomoću transakcija. U praksi aplikacija za ljekarne kao i baza trebala bi podržati rad više korisnika odjednom.

Popis literature

[1] M. Maleković i K. Rabuzin, *Uvod u baze podataka*. Varaždin: Mini-Print-Logo d.o.o., 2015.
[2] M. Maleković i M. Schatten, *Teorija i primjena baza podataka*. Redak, 2017.
[3] R. Manger, *Baze podataka*, 2. izd. Zagreb: Element d.o.o., 2014.
[4] S. Tkalac, *Relacijski model podataka*. Zagreb: DRIP, 1993.
[5] L. Wiese, *Advanced Data Management*. Berlin: De Gruyter, 2015.
[6] TMDB, „Getting Started“. Pristupljeno: 28. svibanj 2024. [Na internetu]. Dostupno na: https://developer.themoviedb.org/reference/tv-series-details
[7] DatabaseTown, „Relational Database Benefits and Limitations“. Pristupljeno: 14. svibanj 2024. [Na internetu]. Dostupno na: https://databasetown.com/relational-database-benefits-and-limitations/
[8] M. Pavlić, *Oblikovanje baza podataka*. Rijeka: Odjel za informatiku Sveučilišta u Rijeci, 2011.
[9] Actian, „What is SQL Server?“ Pristupljeno: 24. kolovoz 2024. [Na internetu]. Dostupno na: https://www.actian.com/what-is-sql-server/
[10] Microsoft, „What is SQL Server Management Studio (SSMS)?“ Pristupljeno: 24. kolovoz 2024. [Na internetu]. Dostupno na: https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms
[11] D. Rubin, „The Three Layered Architecture“. Pristupljeno: 25. kolovoz 2024. [Na internetu]. Dostupno na: https://medium.com/@deanrubin/the-three-layered-architecture-fe30cb0e4a6
[12] AWS, „What is .Net?“ Pristupljeno: 25. kolovoz 2024. [Na internetu]. Dostupno na: https://aws.amazon.com/what-is/net/
[13] A. Aggarwal, „Introduction to C# Windows Forms Applications“. Pristupljeno: 25. kolovoz 2024. [Na internetu]. Dostupno na: https://www.geeksforgeeks.org/introduction-to-c-sharp-windows-forms-applications/
[14] Simplilearn, „What Is C# Entity Framework? A Comprehensive Guide“. Pristupljeno: 25. kolovoz 2024. [Na internetu]. Dostupno na: https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/entity-framework-in-c-sharp

Popis slika

Slika 1: Konceptualna shema nastavnik-predmet 5
Slika 2: Prikaz dugova osoba u obliku grafa sa svojstvima (Prema [5]) 7
Slika 3: Dijagram entiteti-veze ljekarne 16
Slika 4: ERA model 18
Slika 5: Spajanje na SQL Server preko Windows Authentication 20
Slika 6: SQL Server Browse for more 20
Slika 7: Prikaz lokalnih SQL servera 21
Slika 8: Dodavanje novog korisnika na server 22
Slika 9: Kreiranje korisnika na serveru 22
Slika 10: Spajanje na server s dodanim korisnikom 23
Slika 11: Veza sa serverom preko dodanog korisnika 23
Slika 12: Dodavanje nove tablice 24
Slika 13: Prikaz tablice u SSMS 24
Slika 14: Automatsko generiranje vrijednosti primarnog ključa 25
Slika 15: Otvaranje forme za postavljanje ograničenja 25
Slika 16: Forma za dodavanje ograničenja 26
Slika 17: Otvaranje forme za postavljanje indeksa 26
Slika 18: Forma za dodavanje indeksa 27
Slika 19: Otvaranje forme za kreiranje vanjskih ključeva 27
Slika 20: Forma za kreiranje vanjskih ključeva 28
Slika 21: Specifikacija vanjskog ključa 28
Slika 22: Arhitektura aplikacije (Prema [11]) 29
Slika 23: Dodavanje nove stavke u projekt 32
Slika 24: Dodavanje modela baze podataka u projekt 32
Slika 25: Odabir opcije kreiranja modela 33
Slika 26: Dodavanje nove veze prema bazi 33
Slika 27: Upis podataka za spajanje s bazom podataka 34
Slika 28: Odabir veze prema bazi 34
Slika 29: Odabir objekata baze uključenih u projekt 35
Slika 30: Struktura projekta 36
Slika 31: Froma za prijavu 48
Slika 32: Forma promjene podataka s greškom za email 50
Slika 33: Forma za upravljanje artiklima 52
Slika 34: Forma računi 54
Slika 35: Prikaz detalja računa 55
Slika 36: Forma za kreiranje računa 56
Slika 37: Veza Jedinica mjere – Artikl 69

Slika 38: Veza Pacijent - Recept 69
Slika 39: Veza Ustanova - Liječnik.....70
Slika 40: Veza Recept - Liječnik 70
Slika 41: Veza Status narudžbe – Narudžba 70
Slika 42: Veza Narudžba – Dobavljač.....71
Slika 43: Veza Narudžba – Farmaceut 71
Slika 44: Veza Narudžba – Primka 71
Slika 45: Veza Dobavljač - Primka.....71
Slika 46: Veza Primka - Farmaceut 72
Slika 47: Veza Farmaceut - Račun 72
Slika 48: Veza Artikl - Stavke računa.....72
Slika 49: Veza Stavke računa - Račun 73

Popis tablica

Tablica 1: Relacija osoba 4
Tablica 2: Relacija nastavnik 5
Tablica 3: Relacija predmet 6
Tablica 4: Relacija predaje 6
Tablica 5: Relacija Svojstva (Prema [5]) 8
Tablica 6: Rječnik podataka za bazu podataka ljekarne 17
Tablica 7: Relacija Jedinica mjere 63
Tablica 8: Relacija Artikl 64
Tablica 9: Relacija Pacijent 64
Tablica 10: Relacija Ustanova 64
Tablica 11: Relacija Liječnik 65
Tablica 12: Relacija Farmaceut 65
Tablica 13: Relacija Recept 66
Tablica 14: Relacija Dobavljač 66
Tablica 15: Relacija Status narudžbe 67
Tablica 16: Relacija Narudžba 67
Tablica 17: Relacija Primka 68
Tablica 18: Relacija Račun 68
Tablica 19: Relacija Stavke 69

Prilozi

1. Opis relacija

U nastavku će biti opisane sve relacije koje se koriste u bazi podataka na način da će biti relacija biti prikazana u obliku tablica u kojima će se nalaziti naziv atributa, tip podataka te ograničenja nad atributom. Tipovi podataka napisani su u skladu s Microsoft SQL Serverom na kojemu će se nalaziti baza podataka.

1.1. Relacija Jedinica mjere

Relacija Jedinica mjere (Tablica 7) služi za spremanje jedinica mjere kako bi se mogle pridružiti artiklima. Sastoji se od ID-a koji je primarni ključ u koji se sprema znakovni niz od najviše 3 znaka duljine i naziv jedinica mjere u koji se sprema znakovni niz od najviše 10 znakova. Na naziv će biti postavljen UNIQUE ograničenje budući da nema smisla da postoje dvije jedinice mjere s istim nazivom.

Tablica 7: Relacija Jedinica mjere

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	NVARCHAR(3)	NOT NULL
Naziv	NVARCHAR(10)	NOT NULL, UNIQUE

1.2. Relacija Artikl

Relacija Artikl (Tablica 8) sastoji se od ID-a koji predstavlja primarni ključ i tip je podatka IDENTITY koji se automatski generira. Naziv artikla je niz znakova od maksimalno 45 znakova te će na njega biti postavljen UNIQUE ograničenje. Cijena je decimalni broj na koju će biti postavljeno ograničenje da ne može ići u minus. Količina kao i kod cijene ne može ići u minus, ali je cjelobrojni tip podataka. JedinicaMjereID predstavlja vanjski ključ na tablicu Jedinica mjere. Svi atributi relacije moraju biti ispunjeni kako bi se artikl dodao u bazu.

Tablica 8: Relacija Artikl

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Naziv	NVARCHAR(45)	NOT NULL, UNIQUE
Cijena	FLOAT	NOT NULL, >=0
Količina	INT	NOT NULL, >=0
JedinicaMjereID (Vanjski ključ)	NVARCHAR(3)	NOT NULL

1.3. Relacija Pacijent

Relacija Pacijent (Tablica 9) sastoji se od ID-a koji je primarni ključ relacije te se automatski generira. MBO je niz znakova od točno 9 znakova duljine koji ima UNIQUE ograničenje. Ime predstavlja niz znakova od maksimalno 20 znakova duljine, prezime 30, a adresa 200. Datum rođenja je tipa DATETIME i on je opcionalan kao i adresa. Svi ostali atributi su obavezni.

Tablica 9: Relacija Pacijent

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
MBO	NVARCHAR(9)	NOT NULL, UNIQUE
Ime	NVARCHAR(20)	NOT NULL
Prezime	NVARCHAR(30)	NOT NULL
Adresa	NVARCHAR(200)	NULL
DatumRodenja	DATETIME	NULL

1.4. Relacija Ustanova

Relacija Ustanova (Tablica 10) predstavlja radno mjesto liječnika u kojoj radi. Sastoji se od ID-a koji je primarni ključ relacija i automatski se generira. Naziv je niz znakova maksimalne duljine od 50 znaka na koji je postavljeno UNIQUE ograničenje. Adresa je niz znakova od maksimalno 200 znakova. Svi atributi relacije su obavezni.

Tablica 10: Relacija Ustanova

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Naziv	NVARCHAR(50)	NOT NULL, UNIQUE
Adresa	NVARCHAR(200)	NOT NULL

1.5. Relacija Liječnik

Relacija Liječnik (Tablica 11) sastoji se od ID-a koji je primarni ključ i automatski se generira. Ime je niz znakova od maksimalne duljine od 20 znaka, prezime od maksimalno 30, adresa maksimalno od 200, a telefon maksimalno od 20 znakova. Atribut UstanovaID predstavlja vanjski ključ na relaciju Ustanova. Svi atributi osim adrese i telefona su obavezni.

Tablica 11: Relacija Liječnik

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Ime	NVARCHAR(20)	NOT NULL
Prezime	NVARCHAR(30)	NOT NULL
Adresa	NVARCHAR(200)	NULL
Telefon	NVARCHAR(20)	NULL
UstanovaID (Vanjski ključ)	INT	NOT NULL

1.6. Relacija Farmaceut

Relacija Farmaceut (Tablica 12) sastoji se od ID-a koji predstavlja primarni ključ relacije te se automatski generira prilikom unosa. Ime je niz znakova od maksimalne duljine od 20 znaka, prezime maksimalno 30, lozinka maksimalno 50, adresa maksimalno 200 i email maksimalno 50 znakova. Atribut korime predstavlja korisničko ime farmaceuta kojim se prijavljuje u aplikaciju, sastoji se od maksimalno 20 znakova i mora biti jedinstveno kao i email.

Tablica 12: Relacija Farmaceut

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Ime	NVARCHAR(20)	NOT NULL
Prezime	NVARCHAR(30)	NOT NULL
Korime	NVARCHAR(20)	NOT NULL, UNIQUE
Lozinka	NVARCHAR(50)	NOT NULL
Adresa	NVARCHAR(200)	NULL
Email	NVARCHAR(50)	NOT NULL, UNIQUE

1.7. Relacija Recept

Relacija Recept (Tablica 13) sastoji se od ID-a koji je primarni ključ relacije te se automatski generira. Sastoji se od datuma kako bi se znalo u koje vrijeme je recept kreiran te od vanjskih ključeva LijecnikID i PacijentID koji su vezani na relacije Liječnik i Pacijent kako bi se znalo koji liječnik je napravio recept i za kojeg pacijenta. Svi atributi su obavezni.

Tablica 13: Relacija Recept

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Datum	DATETIME	NOT NULL
LijecnikID (Vanjski ključ)	INT	NOT NULL
PacijentID (Vanjski ključ)	INT	NOT NULL

1.8. Relacija Dobavljač

Relacija Dobavljač (Tablica 14) ima atribut ID koji je primarni ključ, automatski se generira. OIB je niz znakova koji mora sadržavati točno 11 znakova i mora biti jedinstven. Naziv se sastoji od maksimalno 45 znakova, IBAN od točno 21 i email do maksimalno 50. Isto kao i OIB, naziv, IBAN i email moraju biti jedinstveni. Adresa se sastoji od maksimalno 200 znakova. Svi atributi su obavezni.

Tablica 14: Relacija Dobavljač

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
OIB	NVARCHAR(11)	NOT NULL, UNIQUE
Naziv	NVARCHAR(45)	NOT NULL, UNIQUE
IBAN	NVARCHAR(21)	NOT NULL, UNIQUE
Email	NVARCHAR(50)	NOT NULL, UNIQUE
Adresa	NVARCHAR(200)	NOT NULL

1.9. Relacija Status narudžbe

Relacija Status narudžbe (Tablica 15) sprema moguća stanja u kojima narudžba može biti. Za potrebe ove aplikacije narudžba može biti u stanju „U izradi“ ili „Zaključena“. Dodana je tablica ukoliko bi došlo do zahtjeva za dodavanjem novih statusa kako bi se mogli lakše dodati i manipulirati u aplikaciji. Relacija se sastoji od atributa ID koji je primarni ključ relacije i automatski se generira. Naziv predstavlja niz znakova od maksimalne duljine 15 znakova te mora biti jedinstven.

Tablica 15: Relacija Status narudžbe

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Naziv	NVARCHAR(15)	NOT NULL, UNIQUE

1.10. Relacija Narudžba

Relacija Narudžba (Tablica 16) sastoji se od ID-a koji je primarni ključ relacije koji se automatski generira prilikom dodavanja. Atribut datum je tipa DATETIME kako bi se znalo u koje vrijeme je narudžba kreirana. StatusID je vanjski ključ na relaciju Status narudžbe. DobavljacID i FarmaceutID su vanjski ključevi na relacije Farmaceut i Dobavljač kako bi se znalo koji farmaceut je kreirao narudžbu i za kojeg dobavljača. Svi atributi su obavezni.

Tablica 16: Relacija Narudžba

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Datum	DATETIME	NOT NULL
DobavljacID (Vanjski ključ)	INT	NOT NULL
StatusID (Vanjski ključ)	INT	NOT NULL
FarmaceutID(Vanjski ključ)	INT	NOT NULL

1.11. Relacija Primka

Relacija Primka (Tablica 17) sastoji se od ID-a koji je primarni ključ i automatski se generira. DatumKnjizenja je atribut za spremanje točnog vremena kreiranje primke i tipa je DATETIME. DobavljacID, FarmaceutID i NarudzbaID su vanjski ključevi na tablice Dobavljač, Farmaceut i Narudžba. NarudzbaID je opcionalni atribut koji služi ako se kreirana primka želi referencirati na određenu narudžbu dok su svi ostali atributi obavezni.

Tablica 17: Relacija Primka

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
DatumKnjizenja	DATETIME	NOT NULL
DobavljacID (Vanjski ključ)	INT	NOT NULL
NarudzbaID (Vanjski ključ)	INT	NULL
FarmaceutID (Vanjski ključ)	INT	NOT NULL

1.12. Relacija Račun

Relacija Račun (Tablica 18) sastoji je od ID-a koji je primarni ključ relacije, automatski se generira prilikom dodavanja. Datum je tipa DATETIME i služi za bilježenje vremena kada je račun stvoren. FarmaceutID je vanjski ključ na relaciju Farmaceut kako bi se znalo koji farmaceut je napravio račun. Svi atributi relacije su obavezni.

Tablica 18: Relacija Račun

Naziv atributa	Tip podatka	Ograničenja
ID (Primarni ključ)	IDENTITY(1,1)	NOT NULL
Datum	DATETIME	NOT NULL
FarmaceutID (Vanjski ključ)	INT	NOT NULL

1.13. Relacija Stavke

Relacija Stavke (Tablica 19) potrebna je kako bi se bilježile stavke dokumenata. Dokumenti narudžba, primka, račun i recept trebaju imati stavke koje predstavljaju artikli. Svaka od navedenih tablica ima svoju relaciju stavke, no ovdje će biti općeniti opis tih relacija budući da su sve skoro identične samo što se koriste za druge dokumente. DokumentID i ArtiklID predstavljaju složeni primarni ključ, a zasebno su vanjski ključevi na Relaciju artikl i relacije Narudžba, Primka, Recept ili Račun ovisno za koji se dokument bilježe stavke. Količina je cijeli broj koji ne može biti manji od nule. Svi atributi relacije su obavezni.

Tablica 19: Relacija Stavke

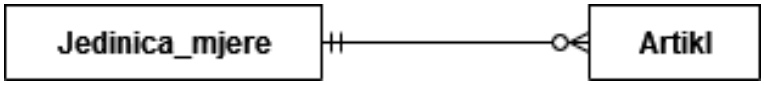
Naziv atributa	Tip podatka	Ograničenja
<DokumentID> (Vanjski, primarni ključ)	INT	NOT NULL
ArtiklID (Vanjski, primarni ključ)	INT	NOT NULL
Količina	INT	NOT NULL

2. Opis veza

U opisu veza biti će objašnjene veze između entiteta. Kao i kod relacija, veze između dokumenata i stavki neće biti prikazane na svakom dokumentu, nego na jednom primjeru budući da je za svaki dokument ista situacija.

2.1. Veza Jedinica mjere – Artikl

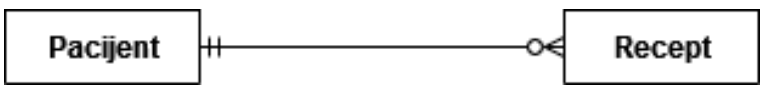
Svaki artikl mora imati jedinicu mjere. Jedinica mjere može biti povezana s jednim ili više artikala, a isto tako može samo postojati u bazi i ne imati pripadajući artikl (Slika 37). Artikl mora biti povezan s jednom jedinicom mjere.



Slika 37: Veza Jedinica mjere – Artikl

2.2. Veza Pacijent – Recept

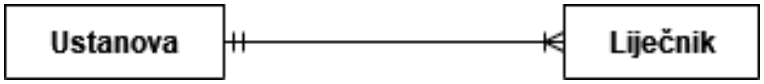
Kada se izrađuju recepti svaki recept mora biti izdan za jednog pacijenta (Slika 38). Za pacijenta ne mora biti izdan nijedan recept ali ih može biti i više.



Slika 38: Veza Pacijent - Recept

2.3. Veza Ustanova – Liječnik

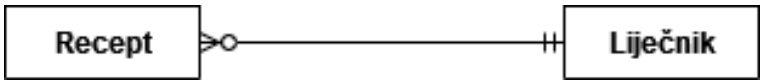
Ustanova mora imati jednog liječnika, ali može imati i više njih (Slika 39). Liječnik radi u jednoj i samo jednoj ustanovi.



Slika 39: Veza Ustanova - Liječnik

2.4. Veza Recept – Liječnik

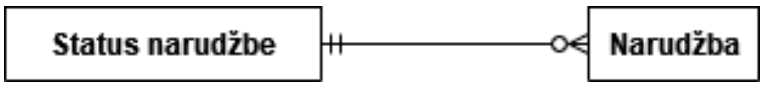
Liječnik može izdati recept, ali i ne mora, može izdati i više recepata (Slika 40). Recept koji postoji mora biti izdan od jednog liječnika.



Slika 40: Veza Recept - Liječnik

2.5. Veza Status narudžbe – Narudžba

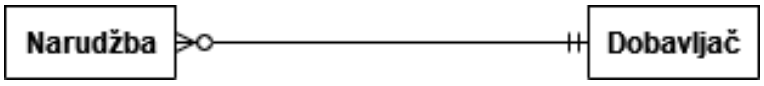
Narudžba mora imati neki status (Slika 41) dok status narudžbe može pripadati više narudžbi. Moguća je situacija da postoji status koji nije vezan za nijednu narudžbu.



Slika 41: Veza Status narudžbe – Narudžba

2.6. Veza Narudžba – Dobavljač

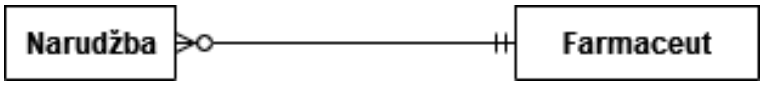
Narudžba mora biti kreirana za nekog dobavljača (Slika 42). Za dobavljača može biti kreirano više narudžbi, ali isto tako može postojati dobavljač za kojeg nije kreirana nijedna narudžba.



Slika 42: Veza Narudžba – Dobavljač

2.7. Veza Narudžba – Farmaceut

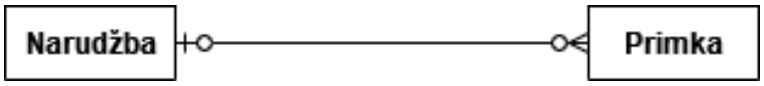
Narudžba mora biti kreirana od jednog farmaceuta (Slika 43). Farmaceut može kreirati više narudžbi, ali može postojati farmaceut koji nije kreirao niti jednu narudžbu.



Slika 43: Veza Narudžba – Farmaceut

2.8. Veza Narudžba – Primka

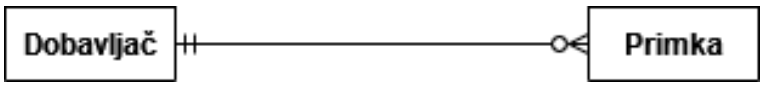
Primka može biti referencirana na narudžbu, ali i ne mora (Slika 44). Narudžba može biti referenca na više primki, ali ne mora biti referenca ni na jednu primku.



Slika 44: Veza Narudžba – Primka

2.9. Veza Dobavljač – Primka

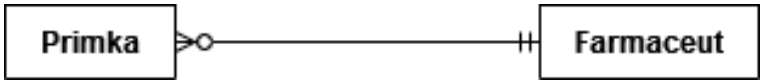
Primka kreirana u sustavu mora imati dobavljača prema kojem je napravljena (Slika 45). Za dobavljača u sustavu može biti upisano više primki, ali ne mora imati niti jednu upisanu primku.



Slika 45: Veza Dobavljač - Primka

2.10. Veza Primka – Farmaceut

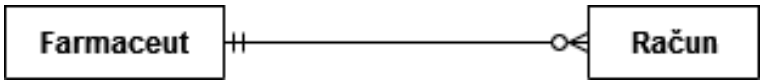
Farmaceut može napraviti više primki, no ne mora napraviti nijednu (Slika 46). Primka mora biti povezana s jednim farmaceutom kako bi se znalo tko je primku kreirao.



Slika 46: Veza Primka - Farmaceut

2.11. Veza Farmaceut – Račun

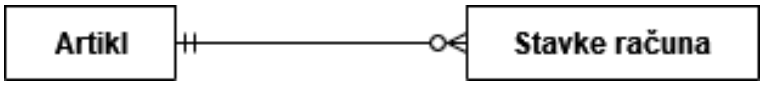
Farmaceut može kreirati više računa u sustavu (Slika 47), ali može postojati farmaceut koji nije kreirao niti jedan račun. Račun mora biti povezan s farmaceutom koji ga je kreirao.



Slika 47: Veza Farmaceut - Račun

2.12. Veza Artikl – Stavke računa

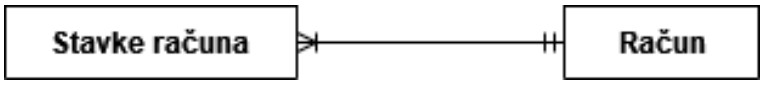
Svaka stavka računa mora biti povezana s jednim artiklom (Slika 48). Artikl može postojati na više stavki računa, ali isto tako može postojati artikl koji se ne nalazi ni na jednoj stavki. Ova veza vrijedi i za stavke recepta, narudžbe i primke.



Slika 48: Veza Artikl - Stavke računa

2.13. Veza Stavke računa – Račun

Svaki račun mora imati barem jednu stavku, ali naravno može imati i više njih (Slika 49). Stavka računa mora pripadati točno jednom računu. Veza vrijedi i za primku, narudžbu i recept.



Slika 49: Veza Stavke računa - Račun

3. SQL skripta baze podataka

```
USE [master]
GO
/****** Object:  Database [BAZA_LJEKARNA]    Script Date: 22.8.2024. 16:05:30 ******/
CREATE DATABASE [BAZA_LJEKARNA]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'BAZA_LJEKARNA', FILENAME = N'D:\Microsoft SQL
Server\MSSQL16.SQLEXPRESS\MSSQL\DATA\BAZA_LJEKARNA.mdf' , SIZE = 8192KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'BAZA_LJEKARNA_log', FILENAME = N'D:\Microsoft SQL
Server\MSSQL16.SQLEXPRESS\MSSQL\DATA\BAZA_LJEKARNA_log.ldf' , SIZE = 8192KB , MAXSIZE = 2048GB
, FILEGROWTH = 65536KB )
 WITH CATALOG_COLLATION = DATABASE_DEFAULT, LEDGER = OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET COMPATIBILITY_LEVEL = 160
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [BAZA_LJEKARNA].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
ALTER DATABASE [BAZA_LJEKARNA] SET ANSI_NULL_DEFAULT OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET ANSI_NULLS OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET ANSI_PADDING OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET ANSI_WARNINGS OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET ARITHABORT OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET AUTO_CLOSE ON
GO
ALTER DATABASE [BAZA_LJEKARNA] SET AUTO_SHRINK OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET AUTO_UPDATE_STATISTICS ON
GO
ALTER DATABASE [BAZA_LJEKARNA] SET CURSOR_CLOSE_ON_COMMIT OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET CURSOR_DEFAULT  GLOBAL
GO
ALTER DATABASE [BAZA_LJEKARNA] SET CONCAT_NULL_YIELDS_NULL OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET NUMERIC_ROUNDABORT OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET QUOTED_IDENTIFIER OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET RECURSIVE_TRIGGERS OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET  ENABLE_BROKER
GO
ALTER DATABASE [BAZA_LJEKARNA] SET AUTO_UPDATE_STATISTICS_ASYNC OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET DATE_CORRELATION_OPTIMIZATION OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET TRUSTWORTHY OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET ALLOW_SNAPSHOT_ISOLATION OFF
GO
ALTER DATABASE [BAZA_LJEKARNA] SET PARAMETERIZATION SIMPLE
```

```
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET READ_COMMITTED_SNAPSHOT OFF
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET HONOR_BROKER_PRIORITY OFF
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET RECOVERY SIMPLE
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET  MULTI_USER
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET PAGE_VERIFY CHECKSUM
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET DB_CHAINING OFF
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET FILESTREAM( NON_TRANSACTED_ACCESS = OFF )
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET TARGET_RECOVERY_TIME = 60 SECONDS
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET DELAYED_DURABILITY = DISABLED
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET ACCELERATED_DATABASE_RECOVERY = OFF
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET QUERY_STORE = ON
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET QUERY_STORE (OPERATION_MODE = READ_WRITE,
CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30), DATA_FLUSH_INTERVAL_SECONDS = 900,
INTERVAL_LENGTH_MINUTES = 60, MAX_STORAGE_SIZE_MB = 1000, QUERY_CAPTURE_MODE = AUTO,
SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200, WAIT_STATS_CAPTURE_MODE = ON)
	GO
	USE [BAZA_LJEKARNA]
	GO
	/****** Object:  Table [dbo].[Artikl]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Artikl](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Naziv] [nvarchar](45) NOT NULL,
		[Cijena] [float] NOT NULL,
		[Kolicina] [int] NOT NULL,
		[JedinicaMjereID] [nvarchar](3) NOT NULL,
	 CONSTRAINT [PK_Artikl] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Artikl_Naziv] UNIQUE NONCLUSTERED
	(
		[Naziv] ASC
	)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[Dobavljac]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Dobavljac](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[OIB] [nvarchar](11) NOT NULL,
		[Naziv] [nvarchar](45) NOT NULL,
		[IBAN] [nvarchar](21) NOT NULL,
		[Email] [nvarchar](50) NOT NULL,
		[Adresa] [nvarchar](200) NOT NULL,
	 CONSTRAINT [PK_Dobavljac] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Dobavljac_Email] UNIQUE NONCLUSTERED
	(
		[Email] ASC
	)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Dobavljac_IBAN] UNIQUE NONCLUSTERED
	(
```

```
		[IBAN] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Dobavljac_Naziv] UNIQUE NONCLUSTERED
	(
		[Naziv] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Dobavljac_OIB] UNIQUE NONCLUSTERED
	(
		[OIB] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[Farmaceut]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Farmaceut](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Ime] [nvarchar](20) NOT NULL,
		[Prezime] [nvarchar](30) NOT NULL,
		[Korime] [nvarchar](20) NOT NULL,
		[Lozinka] [nvarchar](50) NOT NULL,
		[Adresa] [nvarchar](200) NULL,
		[Email] [nvarchar](50) NOT NULL,
	 CONSTRAINT [PK_Farmaceut] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Farmaceut_Email] UNIQUE NONCLUSTERED
	(
		[Email] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Farmaceut_Korime] UNIQUE NONCLUSTERED
	(
		[Korime] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[JedinicaMjere]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[JedinicaMjere](
		[ID] [nvarchar](3) NOT NULL,
		[Naziv] [nvarchar](10) NOT NULL,
	 CONSTRAINT [PK_JedinicaMjere] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [IX_JedinicaMjere] UNIQUE NONCLUSTERED
	(
		[Naziv] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[Lijecnik]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Lijecnik](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Ime] [nvarchar](20) NOT NULL,
		[Prezime] [nvarchar](30) NOT NULL,
		[Adresa] [nvarchar](200) NULL,
		[Telefon] [nvarchar](20) NULL,
		[UstanovaID] [int] NOT NULL,
```

```
 CONSTRAINT [PK_Lijecnik] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Narudzba]    Script Date: 22.8.2024. 16:05:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Narudzba](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Datum] [datetime] NOT NULL,
	[DobavljacID] [int] NOT NULL,
	[StatusID] [int] NOT NULL,
	[FarmaceutID] [int] NOT NULL,
 CONSTRAINT [PK_Narudzba] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Pacijent]    Script Date: 22.8.2024. 16:05:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Pacijent](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[MBO] [nvarchar](9) NOT NULL,
	[Ime] [nvarchar](20) NOT NULL,
	[Prezime] [nvarchar](30) NOT NULL,
	[Adresa] [nvarchar](200) NULL,
	[DatumRodenja] [datetime] NULL,
 CONSTRAINT [PK_Pacijent] PRIMARY KEY CLUSTERED
(
	[ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UNIQUE_Pacijent_MBO] UNIQUE NONCLUSTERED
(
	[MBO] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Podaci]    Script Date: 22.8.2024. 16:05:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Podaci](
	[Naziv] [nvarchar](50) NOT NULL,
	[Adresa] [nvarchar](200) NOT NULL,
	[OIB] [nvarchar](11) NOT NULL,
	[Telefon] [nvarchar](20) NOT NULL,
	[Logo] [varbinary](max) NOT NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Primka]    Script Date: 22.8.2024. 16:05:30 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Primka](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[DatumKnjizenja] [datetime] NOT NULL,
	[DobavljacID] [int] NOT NULL,
	[NarudzbaID] [int] NULL,
	[FarmaceutID] [int] NOT NULL,
 CONSTRAINT [PK_Primka] PRIMARY KEY CLUSTERED
(
	[ID] ASC
```

```
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[Racun]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Racun](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Datum] [datetime] NOT NULL,
		[FarmaceutID] [int] NOT NULL,
	 CONSTRAINT [PK_Racun] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[Recept]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Recept](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Datum] [datetime] NOT NULL,
		[LijecnikID] [int] NOT NULL,
		[PacijentID] [int] NOT NULL,
	 CONSTRAINT [PK_Recept] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[StatusNarudzbe]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[StatusNarudzbe](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Naziv] [nvarchar](15) NOT NULL,
	 CONSTRAINT [PK_StatusNarudzbe] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_StatusNarudzbe_Naziv] UNIQUE NONCLUSTERED
	(
		[Naziv] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[StavkeNarudzbe]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[StavkeNarudzbe](
		[ArtiklID] [int] NOT NULL,
		[NarudzbaID] [int] NOT NULL,
		[Kolicina] [int] NOT NULL,
	 CONSTRAINT [PK_StavkeNarudzbe] PRIMARY KEY CLUSTERED
	(
		[ArtiklID] ASC,
		[NarudzbaID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[StavkePrimke]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
```

```
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[StavkePrimke](
		[PrimkaID] [int] NOT NULL,
		[ArtiklID] [int] NOT NULL,
		[Kolicina] [int] NOT NULL,
	 CONSTRAINT [PK_StavkePrimke] PRIMARY KEY CLUSTERED
	(
		[PrimkaID] ASC,
		[ArtiklID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[StavkeRacuna]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[StavkeRacuna](
		[ArtiklID] [int] NOT NULL,
		[RacunID] [int] NOT NULL,
		[Kolicina] [int] NOT NULL,
	 CONSTRAINT [PK_StavkeRacuna] PRIMARY KEY CLUSTERED
	(
		[ArtiklID] ASC,
		[RacunID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[StavkeRecepta]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[StavkeRecepta](
		[ReceptID] [int] NOT NULL,
		[ArtiklID] [int] NOT NULL,
		[Kolicina] [int] NULL,
	 CONSTRAINT [PK_StavkeRecepta] PRIMARY KEY CLUSTERED
	(
		[ReceptID] ASC,
		[ArtiklID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	/****** Object:  Table [dbo].[Ustanova]    Script Date: 22.8.2024. 16:05:30 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	CREATE TABLE [dbo].[Ustanova](
		[ID] [int] IDENTITY(1,1) NOT NULL,
		[Naziv] [nvarchar](50) NOT NULL,
		[Adresa] [nvarchar](200) NOT NULL,
	 CONSTRAINT [PK_Ustanova] PRIMARY KEY CLUSTERED
	(
		[ID] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
	 CONSTRAINT [UNIQUE_Ustanova_Naziv] UNIQUE NONCLUSTERED
	(
		[Naziv] ASC
	)WITH  (PAD_INDEX  =  OFF,  STATISTICS_NORECOMPUTE  =  OFF,  IGNORE_DUP_KEY  =  OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
	) ON [PRIMARY]
	GO
	ALTER TABLE [dbo].[Artikl]  WITH CHECK ADD  CONSTRAINT [FK_Artikl_JedinicaMjere] FOREIGN
KEY([JedinicaMjereID])
	REFERENCES [dbo].[JedinicaMjere] ([ID])
	GO
	ALTER TABLE [dbo].[Artikl] CHECK CONSTRAINT [FK_Artikl_JedinicaMjere]
	GO
	ALTER TABLE [dbo].[Lijecnik]  WITH CHECK ADD  CONSTRAINT [FK_Lijecnik_Ustanova] FOREIGN
KEY([UstanovaID])
```

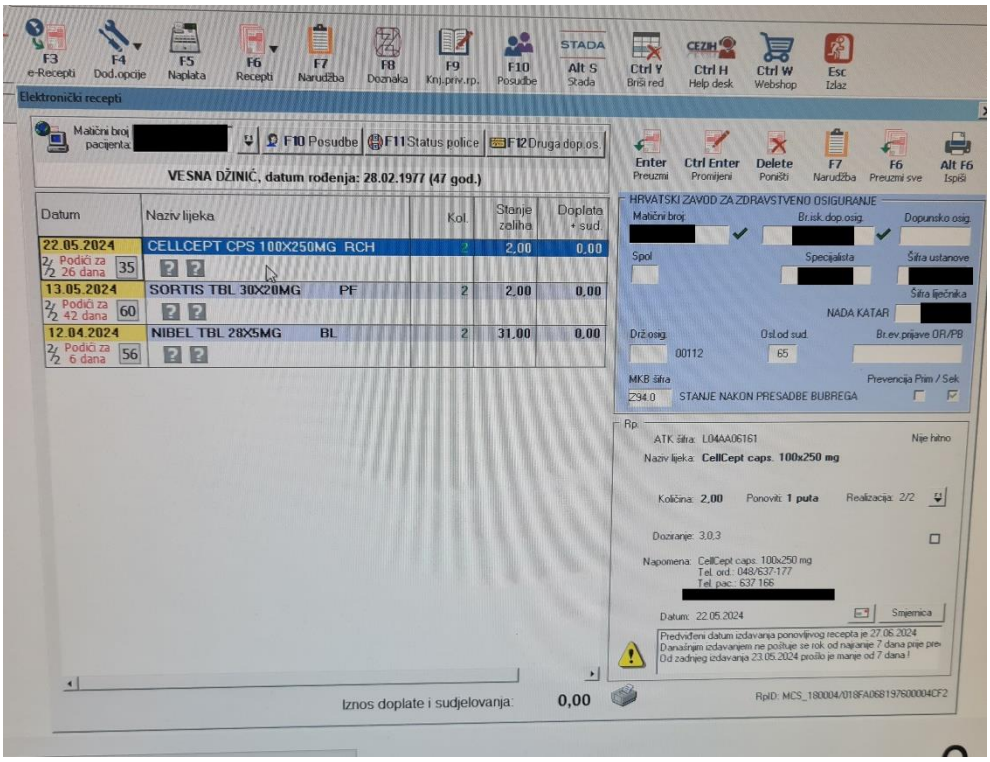
```
        REFERENCES [dbo].[Ustanova] ([ID])
        GO
        ALTER TABLE [dbo].[Lijecnik] CHECK CONSTRAINT [FK_Lijecnik_Ustanova]
        GO
        ALTER TABLE [dbo].[Narudzba]  WITH CHECK ADD  CONSTRAINT [FK_Narudzba_Dobavljac] FOREIGN
KEY([DobavljacID])
        REFERENCES [dbo].[Dobavljac] ([ID])
        GO
        ALTER TABLE [dbo].[Narudzba] CHECK CONSTRAINT [FK_Narudzba_Dobavljac]
        GO
        ALTER TABLE [dbo].[Narudzba]  WITH CHECK ADD  CONSTRAINT [FK_Narudzba_Farmaceut] FOREIGN
KEY([FarmaceutID])
        REFERENCES [dbo].[Farmaceut] ([ID])
        GO
        ALTER TABLE [dbo].[Narudzba] CHECK CONSTRAINT [FK_Narudzba_Farmaceut]
        GO
        ALTER TABLE [dbo].[Narudzba]  WITH CHECK ADD  CONSTRAINT [FK_Narudzba_StatusNarudzbe]
FOREIGN KEY([StatusID])
        REFERENCES [dbo].[StatusNarudzbe] ([ID])
        GO
        ALTER TABLE [dbo].[Narudzba] CHECK CONSTRAINT [FK_Narudzba_StatusNarudzbe]
        GO
        ALTER TABLE [dbo].[Primka]  WITH CHECK ADD  CONSTRAINT [FK_Primka_Dobavljac] FOREIGN
KEY([DobavljacID])
        REFERENCES [dbo].[Dobavljac] ([ID])
        GO
        ALTER TABLE [dbo].[Primka] CHECK CONSTRAINT [FK_Primka_Dobavljac]
        GO
        ALTER TABLE [dbo].[Primka]  WITH CHECK ADD  CONSTRAINT [FK_Primka_Farmaceut] FOREIGN
KEY([FarmaceutID])
        REFERENCES [dbo].[Farmaceut] ([ID])
        GO
        ALTER TABLE [dbo].[Primka] CHECK CONSTRAINT [FK_Primka_Farmaceut]
        GO
        ALTER TABLE [dbo].[Primka]  WITH CHECK ADD  CONSTRAINT [FK_Primka_Narudzba] FOREIGN
KEY([NarudzbaID])
        REFERENCES [dbo].[Narudzba] ([ID])
        GO
        ALTER TABLE [dbo].[Primka] CHECK CONSTRAINT [FK_Primka_Narudzba]
        GO
        ALTER TABLE [dbo].[Racun]  WITH CHECK ADD  CONSTRAINT [FK_Racun_Farmaceut] FOREIGN
KEY([FarmaceutID])
        REFERENCES [dbo].[Farmaceut] ([ID])
        GO
        ALTER TABLE [dbo].[Racun] CHECK CONSTRAINT [FK_Racun_Farmaceut]
        GO
        ALTER TABLE [dbo].[Recept]  WITH CHECK ADD  CONSTRAINT [FK_Recept_Lijecnik] FOREIGN
KEY([LijecnikID])
        REFERENCES [dbo].[Lijecnik] ([ID])
        GO
        ALTER TABLE [dbo].[Recept] CHECK CONSTRAINT [FK_Recept_Lijecnik]
        GO
        ALTER TABLE [dbo].[Recept]  WITH CHECK ADD  CONSTRAINT [FK_Recept_Pacijent] FOREIGN
KEY([PacijentID])
        REFERENCES [dbo].[Pacijent] ([ID])
        GO
        ALTER TABLE [dbo].[Recept] CHECK CONSTRAINT [FK_Recept_Pacijent]
        GO
        ALTER TABLE [dbo].[StavkeNarudzbe]  WITH CHECK ADD  CONSTRAINT [FK_StavkeNarudzbe_Artikl]
FOREIGN KEY([ArtiklID])
        REFERENCES [dbo].[Artikl] ([ID])
        GO
        ALTER TABLE [dbo].[StavkeNarudzbe] CHECK CONSTRAINT [FK_StavkeNarudzbe_Artikl]
        GO
        ALTER    TABLE    [dbo].[StavkeNarudzbe]        WITH    CHECK    ADD        CONSTRAINT
[FK_StavkeNarudzbe_Narudzba] FOREIGN KEY([NarudzbaID])
        REFERENCES [dbo].[Narudzba] ([ID])
        GO
        ALTER TABLE [dbo].[StavkeNarudzbe] CHECK CONSTRAINT [FK_StavkeNarudzbe_Narudzba]
        GO
        ALTER TABLE [dbo].[StavkePrimke]  WITH CHECK ADD  CONSTRAINT [FK_StavkePrimke_Artikl]
FOREIGN KEY([ArtiklID])
        REFERENCES [dbo].[Artikl] ([ID])
        GO
        ALTER TABLE [dbo].[StavkePrimke] CHECK CONSTRAINT [FK_StavkePrimke_Artikl]
        GO
```

```
	ALTER TABLE [dbo].[StavkePrimke]  WITH CHECK ADD  CONSTRAINT [FK_StavkePrimke_Primka]
FOREIGN KEY([PrimkaID])
	REFERENCES [dbo].[Primka] ([ID])
	GO
	ALTER TABLE [dbo].[StavkePrimke] CHECK CONSTRAINT [FK_StavkePrimke_Primka]
	GO
	ALTER TABLE [dbo].[StavkeRacuna]  WITH CHECK ADD  CONSTRAINT [FK_StavkeRacuna_Artikl]
FOREIGN KEY([ArtiklID])
	REFERENCES [dbo].[Artikl] ([ID])
	GO
	ALTER TABLE [dbo].[StavkeRacuna] CHECK CONSTRAINT [FK_StavkeRacuna_Artikl]
	GO
	ALTER TABLE [dbo].[StavkeRacuna]  WITH CHECK ADD  CONSTRAINT [FK_StavkeRacuna_Racun]
FOREIGN KEY([RacunID])
	REFERENCES [dbo].[Racun] ([ID])
	GO
	ALTER TABLE [dbo].[StavkeRacuna] CHECK CONSTRAINT [FK_StavkeRacuna_Racun]
	GO
	ALTER TABLE [dbo].[StavkeRecepta]  WITH CHECK ADD  CONSTRAINT [FK_StavkeRecepta_Artikl]
FOREIGN KEY([ArtiklID])
	REFERENCES [dbo].[Artikl] ([ID])
	GO
	ALTER TABLE [dbo].[StavkeRecepta] CHECK CONSTRAINT [FK_StavkeRecepta_Artikl]
	GO
	ALTER TABLE [dbo].[StavkeRecepta]  WITH CHECK ADD  CONSTRAINT [FK_StavkeRecepta_Recept]
FOREIGN KEY([ReceptID])
	REFERENCES [dbo].[Recept] ([ID])
	GO
	ALTER TABLE [dbo].[StavkeRecepta] CHECK CONSTRAINT [FK_StavkeRecepta_Recept]
	GO
	ALTER TABLE [dbo].[Artikl]  WITH CHECK ADD  CONSTRAINT [CK_Artikl_Cijena] CHECK
(([Cijena]>=(0)))
	GO
	ALTER TABLE [dbo].[Artikl] CHECK CONSTRAINT [CK_Artikl_Cijena]
	GO
	ALTER TABLE [dbo].[Artikl]  WITH CHECK ADD  CONSTRAINT [CK_Artikl_Kolicina] CHECK
(([Kolicina]>=(0)))
	GO
	ALTER TABLE [dbo].[Artikl] CHECK CONSTRAINT [CK_Artikl_Kolicina]
	GO
	ALTER TABLE [dbo].[Artikl]  WITH CHECK ADD  CONSTRAINT [CK_Artikl_Naziv] CHECK
(([Naziv]<>''))
	GO
	ALTER TABLE [dbo].[Artikl] CHECK CONSTRAINT [CK_Artikl_Naziv]
	GO
	ALTER TABLE [dbo].[Dobavljac]  WITH CHECK ADD  CONSTRAINT [CK_Dobavljac_Adresa] CHECK
(([Adresa]<>''))
	GO
	ALTER TABLE [dbo].[Dobavljac] CHECK CONSTRAINT [CK_Dobavljac_Adresa]
	GO
	ALTER TABLE [dbo].[Dobavljac]  WITH CHECK ADD  CONSTRAINT [CK_Dobavljac_Email] CHECK
(([Email]<>''))
	GO
	ALTER TABLE [dbo].[Dobavljac] CHECK CONSTRAINT [CK_Dobavljac_Email]
	GO
	ALTER TABLE [dbo].[Dobavljac]  WITH CHECK ADD  CONSTRAINT [CK_Dobavljac_IBAN] CHECK
((len([IBAN])=(21)))
	GO
	ALTER TABLE [dbo].[Dobavljac] CHECK CONSTRAINT [CK_Dobavljac_IBAN]
	GO
	ALTER TABLE [dbo].[Dobavljac]  WITH CHECK ADD  CONSTRAINT [CK_Dobavljac_Naziv] CHECK
(([Naziv]<>''))
	GO
	ALTER TABLE [dbo].[Dobavljac] CHECK CONSTRAINT [CK_Dobavljac_Naziv]
	GO
	ALTER TABLE [dbo].[Dobavljac]  WITH CHECK ADD  CONSTRAINT [CK_Dobavljac_OIB] CHECK
((len([OIB])=(11)))
	GO
	ALTER TABLE [dbo].[Dobavljac] CHECK CONSTRAINT [CK_Dobavljac_OIB]
	GO
	ALTER TABLE [dbo].[Farmaceut]  WITH CHECK ADD  CONSTRAINT [CK_Farmaceut_Adresa] CHECK
(([Adresa]<>''))
	GO
	ALTER TABLE [dbo].[Farmaceut] CHECK CONSTRAINT [CK_Farmaceut_Adresa]
	GO
	ALTER TABLE [dbo].[Farmaceut]  WITH CHECK ADD  CONSTRAINT [CK_Farmaceut_Email] CHECK
(([Email]<>''))
```

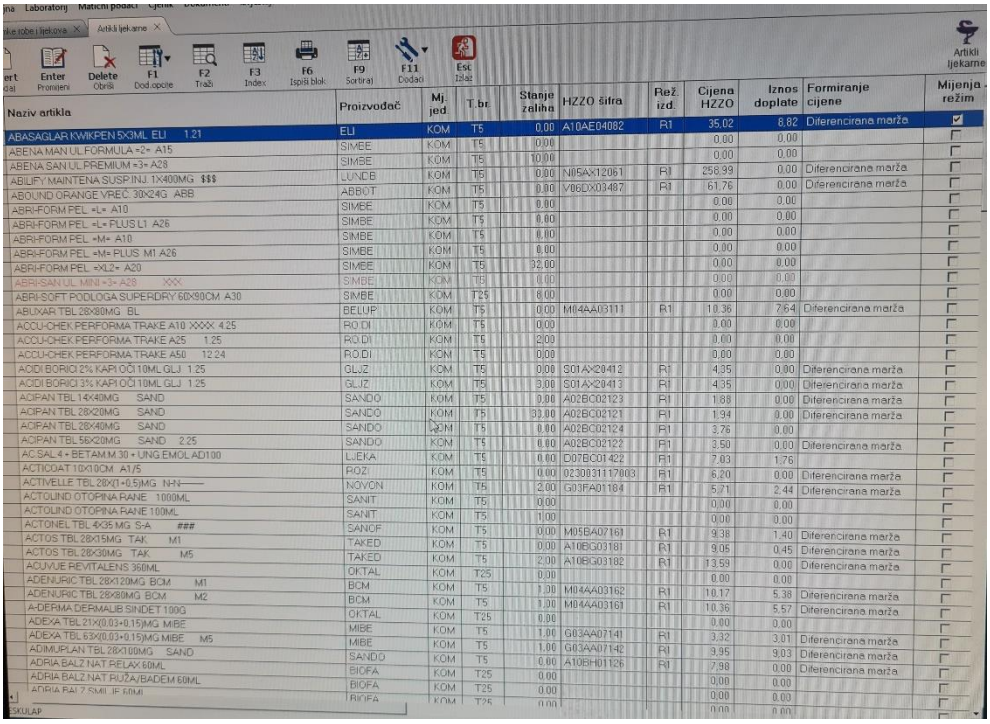
```
	GO
	ALTER TABLE [dbo].[Farmaceut] CHECK CONSTRAINT [CK_Farmaceut_Email]
	GO
	ALTER TABLE [dbo].[Farmaceut]  WITH CHECK ADD  CONSTRAINT [CK_Farmaceut_Ime] CHECK
(([Ime]<>''))
	GO
	ALTER TABLE [dbo].[Farmaceut] CHECK CONSTRAINT [CK_Farmaceut_Ime]
	GO
	ALTER TABLE [dbo].[Farmaceut]  WITH CHECK ADD  CONSTRAINT [CK_Farmaceut_Korime] CHECK
((len([Korime])>=(5)))
	GO
	ALTER TABLE [dbo].[Farmaceut] CHECK CONSTRAINT [CK_Farmaceut_Korime]
	GO
	ALTER TABLE [dbo].[Farmaceut]  WITH CHECK ADD  CONSTRAINT [CK_Farmaceut_Lozinka] CHECK
((len([Lozinka])>=(6)))
	GO
	ALTER TABLE [dbo].[Farmaceut] CHECK CONSTRAINT [CK_Farmaceut_Lozinka]
	GO
	ALTER TABLE [dbo].[Farmaceut]  WITH CHECK ADD  CONSTRAINT [CK_Farmaceut_Prezime] CHECK
(([Prezime]<>''))
	GO
	ALTER TABLE [dbo].[Farmaceut] CHECK CONSTRAINT [CK_Farmaceut_Prezime]
	GO
	ALTER TABLE [dbo].[JedinicaMjere]  WITH CHECK ADD  CONSTRAINT [CK_JedinicaMjere_ID] CHECK
(([ID]<>''))
	GO
	ALTER TABLE [dbo].[JedinicaMjere] CHECK CONSTRAINT [CK_JedinicaMjere_ID]
	GO
	ALTER TABLE [dbo].[JedinicaMjere]  WITH CHECK ADD  CONSTRAINT [CK_JedinicaMjere_Naziv]
CHECK  (([Naziv]<>''))
	GO
	ALTER TABLE [dbo].[JedinicaMjere] CHECK CONSTRAINT [CK_JedinicaMjere_Naziv]
	GO
	ALTER TABLE [dbo].[Lijecnik]  WITH CHECK ADD  CONSTRAINT [CK_Lijecnik_Adresa] CHECK
(([Adresa]<>''))
	GO
	ALTER TABLE [dbo].[Lijecnik] CHECK CONSTRAINT [CK_Lijecnik_Adresa]
	GO
	ALTER TABLE [dbo].[Lijecnik]  WITH CHECK ADD  CONSTRAINT [CK_Lijecnik_Ime] CHECK
(([Ime]<>''))
	GO
	ALTER TABLE [dbo].[Lijecnik] CHECK CONSTRAINT [CK_Lijecnik_Ime]
	GO
	ALTER TABLE [dbo].[Lijecnik]  WITH CHECK ADD  CONSTRAINT [CK_Lijecnik_Prezime] CHECK
(([Prezime]<>''))
	GO
	ALTER TABLE [dbo].[Lijecnik] CHECK CONSTRAINT [CK_Lijecnik_Prezime]
	GO
	ALTER TABLE [dbo].[Lijecnik]  WITH CHECK ADD  CONSTRAINT [CK_Lijecnik_Telefon] CHECK
(([Telefon]<>''))
	GO
	ALTER TABLE [dbo].[Lijecnik] CHECK CONSTRAINT [CK_Lijecnik_Telefon]
	GO
	ALTER TABLE [dbo].[Pacijent]  WITH CHECK ADD  CONSTRAINT [CK_Pacijent_Adresa] CHECK
(([Adresa]<>''))
	GO
	ALTER TABLE [dbo].[Pacijent] CHECK CONSTRAINT [CK_Pacijent_Adresa]
	GO
	ALTER TABLE [dbo].[Pacijent]  WITH CHECK ADD  CONSTRAINT [CK_Pacijent_Ime] CHECK
(([Ime]<>''))
	GO
	ALTER TABLE [dbo].[Pacijent] CHECK CONSTRAINT [CK_Pacijent_Ime]
	GO
	ALTER TABLE [dbo].[Pacijent]  WITH CHECK ADD  CONSTRAINT [CK_Pacijent_MBO] CHECK
((len([MBO])=(9)))
	GO
	ALTER TABLE [dbo].[Pacijent] CHECK CONSTRAINT [CK_Pacijent_MBO]
	GO
	ALTER TABLE [dbo].[Pacijent]  WITH CHECK ADD  CONSTRAINT [CK_Pacijent_Prezime] CHECK
(([Prezime]<>''))
	GO
	ALTER TABLE [dbo].[Pacijent] CHECK CONSTRAINT [CK_Pacijent_Prezime]
	GO
	ALTER TABLE [dbo].[Podaci]  WITH CHECK ADD  CONSTRAINT [CK_Podaci_Adresa] CHECK
(([Adresa]<>''))
	GO
	ALTER TABLE [dbo].[Podaci] CHECK CONSTRAINT [CK_Podaci_Adresa]
```

```
	GO
	ALTER TABLE [dbo].[Podaci] WITH CHECK ADD CONSTRAINT [CK_Podaci_Naziv] CHECK
(([Naziv]<>''))
	GO
	ALTER TABLE [dbo].[Podaci] CHECK CONSTRAINT [CK_Podaci_Naziv]
	GO
	ALTER TABLE [dbo].[Podaci] WITH CHECK ADD CONSTRAINT [CK_Podaci_OIB] CHECK
((len([OIB])=(11)))
	GO
	ALTER TABLE [dbo].[Podaci] CHECK CONSTRAINT [CK_Podaci_OIB]
	GO
	ALTER TABLE [dbo].[Podaci] WITH CHECK ADD CONSTRAINT [CK_Podaci_Telefon] CHECK
(([Telefon]<>''))
	GO
	ALTER TABLE [dbo].[Podaci] CHECK CONSTRAINT [CK_Podaci_Telefon]
	GO
	ALTER TABLE [dbo].[StatusNarudzbe] WITH CHECK ADD CONSTRAINT [CK_StatusNarudzbe_Naziv]
CHECK (([Naziv]<>''))
	GO
	ALTER TABLE [dbo].[StatusNarudzbe] CHECK CONSTRAINT [CK_StatusNarudzbe_Naziv]
	GO
	ALTER TABLE [dbo].[StavkeNarudzbe] WITH CHECK ADD CONSTRAINT
[CK_StavkeNarudzbe_Kolicina] CHECK (([Kolicina]>(0)))
	GO
	ALTER TABLE [dbo].[StavkeNarudzbe] CHECK CONSTRAINT [CK_StavkeNarudzbe_Kolicina]
	GO
	ALTER TABLE [dbo].[StavkePrimke] WITH CHECK ADD CONSTRAINT [CK_StavkePrimke_Kolicina]
CHECK (([Kolicina]>(0)))
	GO
	ALTER TABLE [dbo].[StavkePrimke] CHECK CONSTRAINT [CK_StavkePrimke_Kolicina]
	GO
	ALTER TABLE [dbo].[StavkeRacuna] WITH CHECK ADD CONSTRAINT [CK_StavkeRacuna_Kolicina]
CHECK (([Kolicina]>(0)))
	GO
	ALTER TABLE [dbo].[StavkeRacuna] CHECK CONSTRAINT [CK_StavkeRacuna_Kolicina]
	GO
	ALTER TABLE [dbo].[StavkeRecepta] WITH CHECK ADD CONSTRAINT [CK_StavkeRecepta_Kolicina]
CHECK (([Kolicina]>(0)))
	GO
	ALTER TABLE [dbo].[StavkeRecepta] CHECK CONSTRAINT [CK_StavkeRecepta_Kolicina]
	GO
	ALTER TABLE [dbo].[Ustanova] WITH CHECK ADD CONSTRAINT [CK_Ustanova_Adresa] CHECK
(([Adresa]<>''))
	GO
	ALTER TABLE [dbo].[Ustanova] CHECK CONSTRAINT [CK_Ustanova_Adresa]
	GO
	ALTER TABLE [dbo].[Ustanova] WITH CHECK ADD CONSTRAINT [CK_Ustanova_Naziv] CHECK
(([Naziv]<>''))
	GO
	ALTER TABLE [dbo].[Ustanova] CHECK CONSTRAINT [CK_Ustanova_Naziv]
	GO
	USE [master]
	GO
	ALTER DATABASE [BAZA_LJEKARNA] SET READ_WRITE
	GO
```

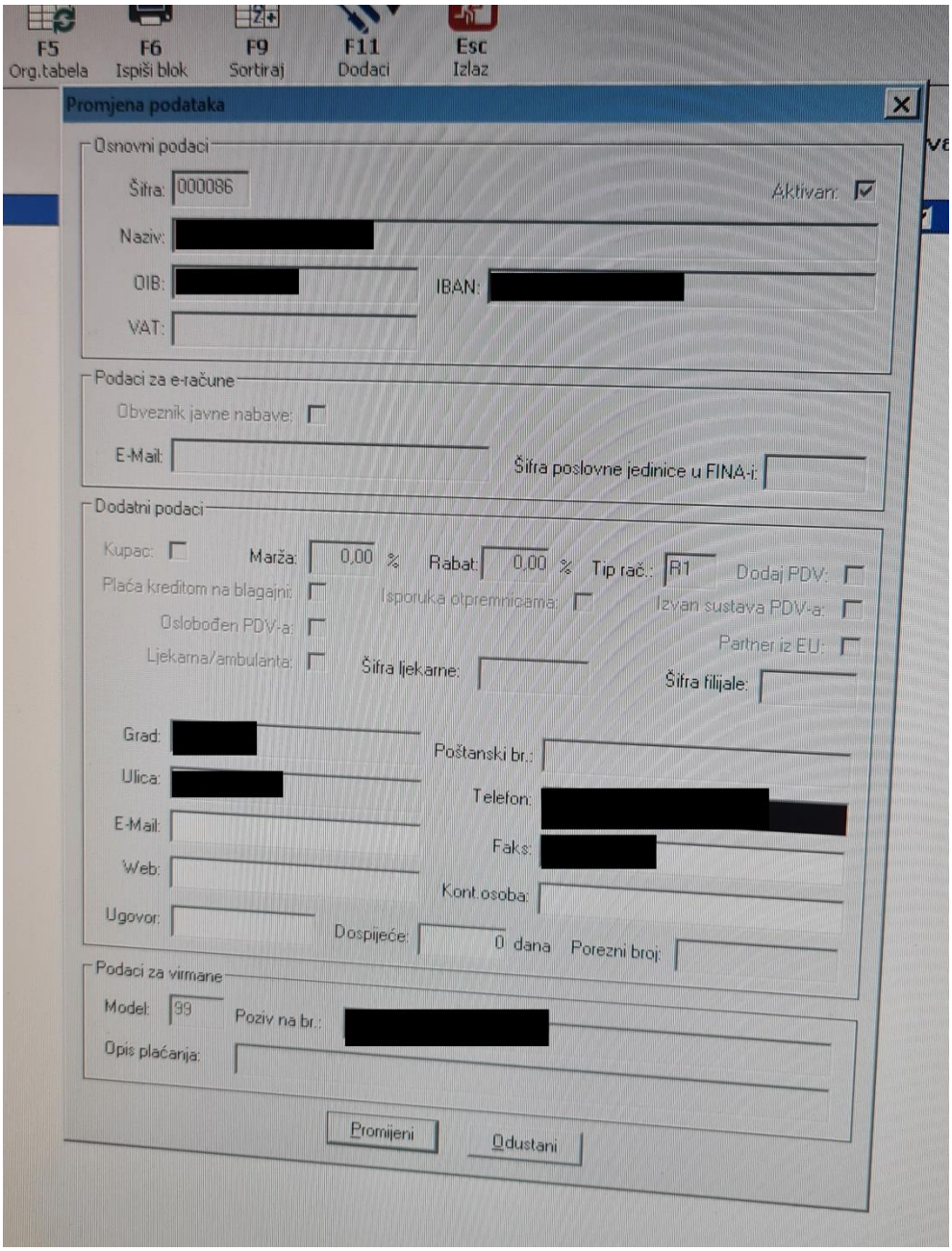
4. Slika zaslona aplikacije recepata



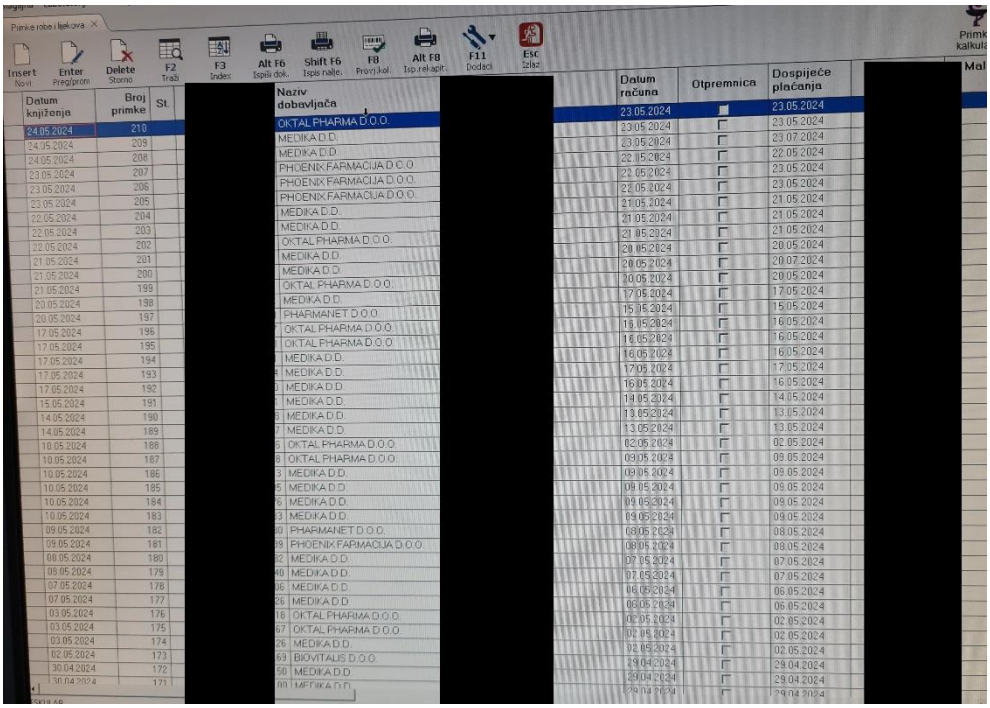
5. Slika zaslona aplikacije artikala



6. Slika zaslona aplikacije dobavljač



7. Slika zaslona aplikacije primke



8. Slika zaslona aplikacije narudžbe