Upotreba baza podataka temeljenih na dokumentima pri razvoju programskih proizvoda

Šajfar, Sanja

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Sanja Šajfar

UPOTREBA BAZA PODATAKA TEMELJENIH NA DOKUMENTIMA PRI RAZVOJU PROGRAMSKIH PROIZVODA

ZAVRŠNI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Sanja Šajfar

JMBAG: 0016136192

Studij: Informacijski sustavi

UPOTREBA BAZA PODATAKA TEMELJENIH NA DOKUMENTIMA PRI RAZVOJU PROGRAMSKIH PROIZVODA

ZAVRŠNI RAD

Mentor :

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, kolovoz 2022.

Sanja Šajfar

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.

Autorica potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Specifični korisnički zahtjevi nerijetko zahtijevaju i specifične pristupe u radu s podacima pri razvoju programskih proizvoda. Jedan od pristupa je i korištenje baza podataka temeljenih na dokumentima (engl. *document-based databases*) koje su se pokazale kao jako dobro rješenje pri implementaciji određenog seta specifičnih, ali i općenitih zahtjeva. Kroz izgradnju aplikacije u C# i izrade baze podataka u MongoDB prikazano je da dokument-baze podataka nude mogućnost rada s različitom vrstom podataka gdje pritom svaki dokument može izgledati drugačije i imati različit broj svojstva, nemaju relacijsku shemu i dokumenti se lagano mapiraju u objekte što ubrzava proces stvaranja aplikacija. S druge strane nedostatak im je neučinkovit rada s relacijama.

Ključne riječi: nereleacijske baze podataka, dokument-baze podataka, MongoDB, C#

Sadržaj

1. Uvod

Tema završnog rada je upotreba baza podataka temeljenih na dokumentima pri razvoju programskih proizvoda. Detaljno ću objasniti pojam baze podataka te njihovu podjelu i sustave za upravljanje bazama podataka. Usporedit ću dokument-baze podataka s relacijskim bazama podataka te navesti njihove prednosti i nedostatke. Također, dotaknut ću se MongoDB i C# koje ću koristiti pri izradi aplikacije.

Motivacija za odabir ove teme je bila želja da više naučim o nerelacijskim bazama podataka, posebno dokument-bazama podataka te kako koristiti MongoDB u sklopu C#. Motivirala me činjenica da ću napraviti svoju prvu aplikaciju koja koristi nerelacijsku bazu podataka jer sam do sada radila isključivo s relacijskim bazama podataka. Najveći razlog zašto sam izabrala ovu temu je zbog toga što sam htjela naučiti nova područja te sam ovu temu smatrala dovoljno izazovnom i zanimljivom.

Cilj ovog završnog rada jest objasniti važnost nerelacijskih baza podataka, konkretnije, dokument-baza podataka te njihove prednosti u odnosu na relacijske baze podataka. Završni rad sastoji se od sedam poglavlja.

U uvodnom poglavlju daje se kratki opis teme, motivaciju i cilj završnog rada te sažetak rada. U drugom poglavlju daje se definicija baze podataka te što su to sustavi za upravljanje bazama podataka. Na početku trećeg poglavlja opisane su relacijske baze podataka i sustavi za upravljanje relacijskim bazama podataka. Drugi dio trećeg poglavlja fokusiran je na SQL i veze između entiteta. Četvero poglavlje rada započinje s definicijom nerelacijskim bazama podataka gdje se više ulazi u njihovu povijest te vrste nerelacijskih baza podataka. Nadalje, u istom poglavlju opisana je svaka podvrsta nerelacijskih baza podataka s primjerima, a poseban fokus se stavlja na dokument-baze podataka. U petom poglavlju se opisuje MongoDB i C# što predstavlja uvod u praktični dio rada, a to je izrada aplikacije. Šesto poglavlje opisuje cijeli proces izrade aplikacije i njenu implementaciju. U završnom dijelu iznosi se zaključak u kojem je opisan sažetak završnog rada te mišljenje i komentar autorice.

2. Definicija baze podataka i sustavi za upravljanje bazama podataka

U ovom poglavlju je objašnjena definicija baze podataka i sustavi za upravljanje bazom podataka te njihova važnost. Detaljnije se analizira definicija baze podataka, kako bi se došlo do zaključka što smatram bazom podataka u ovom radu.

2.1. Definicija baze podataka

Baza podataka je organizirana kolekcija informacija koji su međusobno povezani [1].

Ako pogledamo definiciju što je baza podataka, može se doći do zaključka da su ključna riječi u toj definiciji **organizirana** kolekcija **povezanih** podataka. Bez riječi organizirana, baza podataka može biti čak i kutija u kojoj se spremaju stari računi bez ikakve organizacije. Stoga je bitno reći da se baza podataka sastoji se od podataka koji su međusobno povezani. Baze podataka se koriste kako bi se omogućilo što lakše upravljanje podacima. Kako onda prepoznati što je sve baza podataka? Na primjer, ako ste u potrazi za točno određenim podatkom te jedini način da dođete do tog podatka jest tako što morate pročitati ostale podatke, to nije baza podataka. Zbog toga i kutija u kojoj spremate svoje stare račune koji nisu organizirani nije baza podataka, jer jedini način da pronađete točno određeni račun jest da prolazite i čitate sve račune. Baza podataka bi trebala olakšati i odmah bez potrebe da čitate ostale podatke pronaći podatak koji tražite.

Baze podataka mogu biti u digitalnom ili fizičkom obliku. Jedan od primjera fizičkih baza podataka, na papiru, su TV preglednik i telefonski imenik. Zbog napretka tehnologije baze podataka na papiru se danas gotovo ni ne koriste nego je sve prošlo u digitalan oblik. Što se tiče digitalnih oblika, većina ljudi se susrela s više baza podataka iako to nije ni svjesna. Dobar primjer je direktorij diska koji pokazuje koji pokazuje koji sadržaj se nalazi na disku te njihova veličina, sve su to baze podataka.

Možemo vidjeti da postoje raznovrsni oblici baza podataka. O kojoj od tih baza podataka jest riječ u ovom radu? U ovom radu će se detaljnije fokusirati na baze podataka orijentirane prema dokumentima. Potrebno je reći da je riječ o bazama podataka koje spremaju podatke u obliku dokumenata koji su organizirani i povezani. Dakako, kako bi se moglo raditi s bazama podataka, potrebni su sustavi za upravljanje bazama podataka koji će detaljnije biti objašnjeni u nastavku.

2.2. Sustavi za upravljanje bazama podataka

Sustavi za upravljanje bazama podataka (engl. *Database Management Systems)* su programi koji upravljaju i izvršavaju radnje s podacima u bazama podataka. Skraćeno se za sustave za upravljanje bazama podataka koristi akronim SUBP [1].

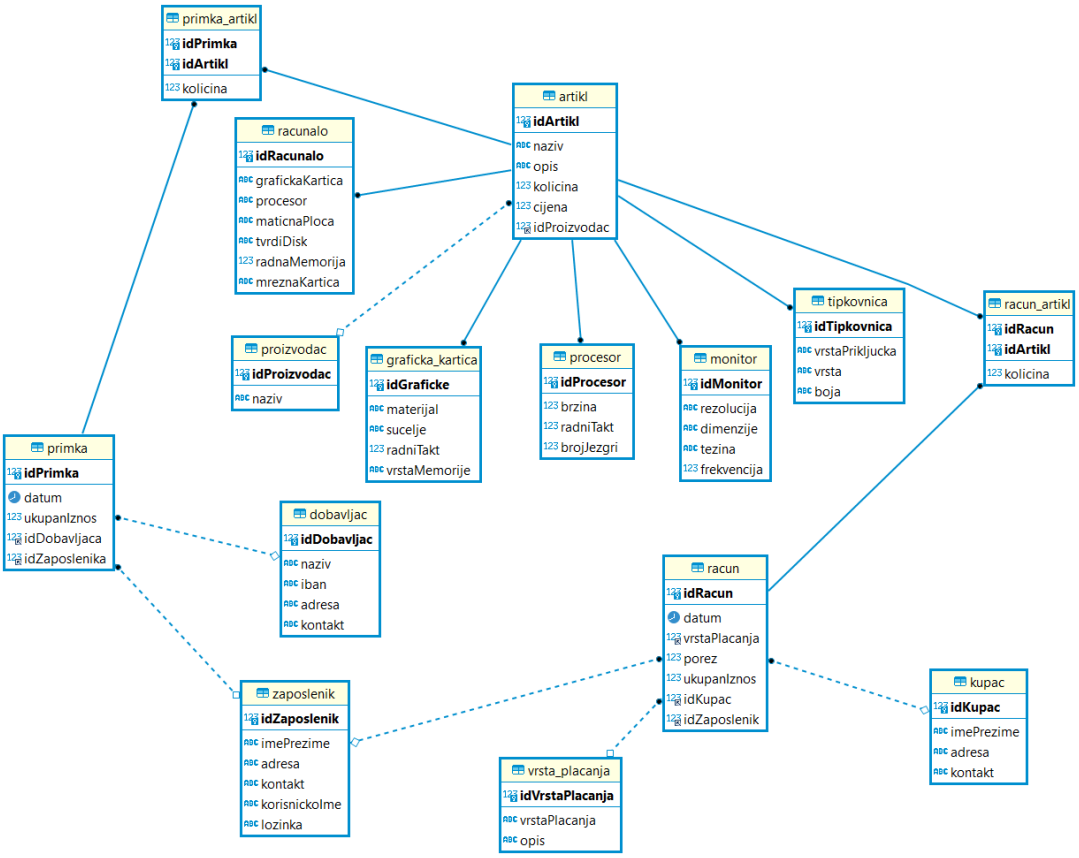
U definiciji se spominje upravljanje i izvršavanje radnji s podacima, time se želi reći

da sustavi za upravljanje bazama podataka omogućuju korisniku da radi s podacima koji se nalaze u bazi podataka na način da korisnik može spremati, preuzimati, dodavati, brisati ili izmijeniti podatke. Bitno je napomenuti da sustavi za upravljanje bazama podataka definiraju shemu podataka, što se odnosi na strukturu podataka. Nakon što znamo čemu služe sustavi za upravljanje bazama podataka, potrebno je postaviti pitanje koje je njihova primarna svrha i cilj? Na to pitanje će se odgovoriti u sljedećem potpoglavlju.

2.2.1. Primarni cilj sustava za upravljanje bazama podataka

Primarni cilj SUBP jest da omogućuje pohranu i dohvaćanje podataka iz baze podataka. Sustavi za upravljanje bazama podataka su važni jer bez postojanja pravila i propisa nije moguće održavati i učinkovito upravljati podacima [1]. Primjerice, prilikom kreiranja nove tablice uzimamo u obzir određenu zajedničku karakteristiku po kojemu ćemo grupirati podatke i staviti ih u tablicu. Što kada imamo kreiranih više tablica? Potrebno je postaviti veze između tablica. Kako bi se postavila veza, odnos, između dvije ili više tablica potrebno je postojanje zajedničkog atributa između tablica. Svaki sustav za upravljanje bazama podataka se razlikuje te se podaci ne spremaju uvijek u tablice. Ukoliko se podaci spremaju u tablice, riječ je o relacijskim bazama podataka koje će biti detaljnije objašnjene kasnije.

Sustavi za upravljanje bazama podataka su napravljeni kako bi se učinkovito moglo upravljati velikim brojem informacija [1]. Riječ "upravljanje" u ovom slučaju odnosi se na definiranu strukturu za pohranu informacija i osiguravanje mehanizma za rad s informacijama (definiranje, spremanje, dohvaćanje i ažuriranje podataka). Tako na primjer, na slici 1 se nalazi primjer fizičkog modela podataka u MySQL-u, to jest konkretnije, vidi se ERA model koji sadrži više entiteta s njihovim pripadajućim atributima i odnosima. ERA model je vrsta fizičkog modeliranja podataka na kojemu se vide entiteti s pripadajućim atributima i veze između tih entiteta što će biti kasnije detaljnije objašnjeno.



Slika 1: Primjer fizičkog modela podataka za MySQL

Također, sustavi za upravljanje bazama podataka trebaju omogućiti određeni sigurnosni sustav kako bi sve informacije bile zaštićene i kako bi se onemogućio neovlašteni pristup podacima. SUBP imaju svoje prednosti i nedostatke koji će detaljno biti objašnjeni u nastavku.

2.2.2. Prednosti sustava za upravljanje bazama podataka

Danas su sustavi za upravljanje bazom podataka neizbježni i koriste se često, s razlogom. Njihova upotreba korisniku omogućuje mnoge prednosti koje će u nastavku biti navedene i objašnjene.

Jedan od glavnih prednosti jest kontrola redundancije. Višak podataka se odnosi na dupliciranje podataka (spremanje istih podataka više puta). U sustavima upravljanja bazama podataka izbjegava se nepotrebno dupliciranje podataka. Kontrolom redundancije uklanjanja se dodatno vrijeme za obradu velikog volumena podataka što rezultira uštedom prostora za pohranu. Sljedeća prednost je poboljšano dijeljenje podataka među korisnicima. SUBP omogućuje korisnicima da dijele podatke u bilo kojem programu. Ujedno, SUBP omogućuje integritet podataka. Pojam integritet podataka odnosi se na točnost i dosljednost podataka. Prilikom stvaranja baze podataka potrebno je obratiti pozornost na integritet podataka. Dobra baza podataka osigurat će integritet podataka, primjerice, korisnik bi mogao slučajno pokušati unijeti telefonski broj u polje za datum koje samo podržava unos podataka tipa *date*. Ako sustav ima ograničenje nad integritetom podataka, spriječit će korisnika u toj pogrešci [2].

Već je bilo spomenuto da su SUBP korisni zbog zaštite podataka u bazama podataka. Moguće je postaviti sigurnosni sustav preko kojeg pristup podacima imaju samo osobe putem odgovarajućeg kanala, tako se mogu omogućiti autorizacijske provjere kada se pokušava pristupiti dosta osjetljivim podacima. Korištenje SUBP omogućuje i dosljednost podataka. Uklanjanjem viška nepotrebnih podataka se uveliko smanjuje mogućnosti nedosljednosti. Jedna od najbitnijih prednosti, ako ne i najbitnija, jest učinkovit pristup podacima. U SUBP podacima upravlja SUBP i sav pristup podacima provodi se putem SUBP-a što je ključ za učinkovitu obradu podataka. Nezavisnost podataka u SUBP je također jedna od prednosti kao i skraćeno vrijeme razvoja i održavanja aplikacija koje se koriste SUBP, s obzirom na to da već sam sustav podržava mnoge važne funkcije koje su zajedničke mnogim aplikacijama [2].

2.2.3. Nedostaci sustava za upravljanje bazama podataka

Nakon što su navedene prednosti, bitno je napomenuti da SUBP imaju i nedostatke. Jedan nedostatak jest složenost sustava, što ovisi od sustava do sustava, ali većina sustava je kreirana kako bi pružila što bolje i što više korisnih funkcionalnosti samom korisniku što rezultira složenošću sustava. Nadalje, troškovi upravljanja SUBP su jedno od većih nedostataka SUBP. Od SUBP se očekuje da imaju brzi procesor i veliku memoriju te da su svi podaci pohranjeni sigurno što rezultira velikim troškovima vezanih uz hardver, softver i samo poslovanje. Uz to, redovito se nadograđuje hardver i ažurira softvera što predstavlja veliki trošak. Posljednji nedostatak je što SUBP mogu biti generalizirani softveri, tu se misli da se SUBP može pristupiti iz ugla da se koriste za više programskih alata i sustava, a ne za određen alat, što rezultira poteškoćama u radu određenih aplikacijama i sporijem radom sustava [2].

Bez obzira na nedostatke, korištenje SUBP se i dalje uveliko isplati jer ima više prednosti nego nedostataka. Nakon što je objašnjeno svrha SUBP, potrebno je objasniti jednu od najkorištenijih vrsta baza podataka, a to su relacijske baze podataka.

3. Relacijske baze podataka

Prije nego počnemo pričati o bazama podataka orijentiranim prema dokumentima, potrebno je objasniti relacijske baze podataka. Već je bilo objašnjeno što su baze podataka i čemu služe, a u daljnjem tekstu će biti objašnjena jedna vrsta baza podataka, a to su relacijske baze podataka.

U današnje vrijeme postoje različite vrste baza podataka, ali danas najpoznatije i najkorištenije su te u kojima se podaci spremaju u tablice, a to su relacijske baze podataka. Relacijske baze podataka su zbirke podataka koji su organizirani u skup tablica iz kojih se može pristupiti podacima [3]. Drugim riječima, podaci su pohranjeni u više tablica koje su međusobno povezane, zbog njihove povezanosti te tablice se nazivaju relacijama.

Za primjer ću uzeti sliku 2 na kojoj se nalazi tablica artikla za prodaju u kojoj se nalaze bitni podaci poput `ArtiklID`, `Naziv_artikla` i `Cijena`. Redovi u tablici predstavljaju entitete koje imaju svoje atribute koji ih opisuju. Redak nazivamo n-torka dok se stupci nazivaju atributi. U jednoj relaciji ne smiju postojati dvije jednake n-torke, to jest dva jednaka reda. Naravno ovo su samo osnovni pojmovi vezani uz relacijske baze podataka, u nastavku će se ti pojmovi detaljnije objasniti.

Artikli

ArtiklID	Naziv artikla	Cijena
1	Usisavač	450
2	Fen za kosu	150
3	Tepih	500
4	Radni stol	600

Slika 2: Primjer fizičkog modela podataka za MySQL

3.1. Sustavi za upravljanje relacijskim bazama podataka

Sustav koji upravlja relacijskim bazama podataka (nadalje SURBP) je program koji omogućava stvaranje, ažuriranje i upravljanje relacijskim bazama podataka [3]. Takva baza podataka je izgrađena od entiteta. Entitet je pojedinačni element u bazi podataka koji može imati svoje atribute te međusobne veze između njih. Razlozi zašto je danas toliko česta uporaba SURBP-a je zbog tablica koje se koriste u svrhu pohranjivanje podataka, na taj način se izbjegava redundancija te se postiže učinkovitiji unos podataka u bazu podataka zbog relacijske sheme i integriteta podataka [3]. Samim podacima je lako pristupiti putem grafičkog korisničkog sučelja (engl. *GUI*), što omogućuje mnogim korisnicima da imaju ovlast pristupiti, čitati, pisati ili ažurirati podatke u bazi.

SURBP ima dosta prednosti. Jedna od prednosti je količina podataka koji se mogu spremati te činjenica da tim podacima mogu pristupiti više korisnika u isto vrijeme. Također, zbog toga što su podaci spremljeni u tablice oni su korisnicima lagano čitljivi što rezultira lakšom upotrebom alata. SURBP mogu imati administratore koji imaju mogućnost kontroliranja pristupa bazi podataka. Tako je moguće da određeni korisnici imaju ograničen pristup, dok drugi imaju veće ovlasti. SURBP imaju standardizirani jezik upita, SQL, koji omogućuje rad s podacima. SQL također omogućava i učinkovitu razmjenu podataka između više različitih baza podataka (čak i između baza podataka koje su pohranjene u različitim sustavima). U nastavku će on biti detaljnije objašnjen.

3.2. SQL

Strukturirani upitni jezik (engl. *Structured Query Language - SQL*) je jezik koji se koristi prilikom rada s relacijskim bazama podataka u svrhu traženja, ažuriranja ili brisanja podataka iz relacijske baze podataka. SQL je danas standardizirani jezik za rad s relacijskim bazama podataka, standardiziran je putem ISO i ANSI standarda [4]. Putem SQL-a se mogu postaviti upiti kako bi se dobile potrebne informacije iz baze podataka, na isti način funkcioniraju i izvješća. Bez obzira na činjenicu da je riječ o standardiziranom jeziku, postoje dosta sustava za rad s bazom podataka koji su proširili SQL za lakši rad s bazom podataka.

SQL se koristi za sve vrste rada s bazom podataka od različitih korisnika, poput administratora sustava, administratora baze podataka, administratora sigurnosti do programera. U današnje vrijeme, SQL nudi naredbe koje se lagano nauče i koje su dosljedne i primjenjive za sve korisnike [4]. Osnovne naredbe se mogu naučiti već u nekoliko sati, a one malo kompliciranije se mogu svladati već u nekoliko dana, što je jedan od razloga, uz već prije spomenutu standardizaciju, zbog čega je taj jezik toliko popularan.

Budući da u SQL-u postoje različite naredbe, one se mogu kategorizirati u jednu od sljedećih kategorija koje možemo smatrati podjezicima te će svaki od njih biti detaljnije opisan u sljedećim potpoglavljima:

- Jezik za definiranje podataka (engl. *Data Definition Language - DDL*)
- Jezik za upravljanje podacima (engl. *Data Manipulation Language - DML*)
- Jezik za upite nad podacima (engl. *Data Query Language - DQL*)
- Jezik za kontrolu podataka (engl. *Data Control Language - DCL*)

3.2.1. DDL

Jezik koji sadrži naredbe za kreiranje i izmjenu strukture objekata u bazi podataka. Ti objekti mogu biti tablice, indeksi, sheme i pogledi. Naredbe koje se mogu koristiti su sljedeće:

- `CREATE` - koristi se za kreiranje nove tablice sa sljedećom sintaksom `CREATE TABLE [naziv_tablice] ([definicije_stupcaca])[parametri_tablice];`

- `ALTER` - koristi se za izmjenu postojeće tablice. Točnije, ova naredba može dodati nove, izbrisati postojeće ili promijeniti postojeće stupce u tablici uz korištenje sljedeće sintakse `ALTER objekt tip objekt naziv parametri;`
- `DROP` - koristi se za brisanje tablica, indeksa i pogleda uz sljedeću sintaksu `DROP objekt tip objekt naziv;`. Važno je napomenti kako je ova naredba nepovratna, tj. jednom obrisani objekti ne mogu više biti vraćeni osim ponovnim korištenjem `CREATE` naredbe.
- `TRUNCATE` - koristi se za brisanje svih podataka iz tablice sa sintaksom `TRUNCATE TABLE naziv_tablice;`. Za razliku od lstinlineDROP naredbe, ova naredba briše samo podatke iz tablice čuvajući samu strukturu tablice kako bi se ponovno mogla iskoristiti.

3.2.2. DML

Jezik koji sadrži naredbe za upravljanje podacima u bazi, tj. omogućava umetanje, brisanje i izmjenu podataka. Naredbe koje se mogu koristiti su sljedeće:

- `UPDATE` - koristi se za izmjenu jednog ili više zapisa u tablici. Sintaksa je `UPDATE [naziv_tablice] SET [naziv_stupca = vrijednost] WHERE [uvjet];`.
- `INSERT` - koristi se za umetanje jednog ili više zapisa u tablicu. Sintaksa je `INSERT INTO [naziv_tablice] [stupac/ci] VALUES [vrijednost/i];`.
- `DELETE` - koristi se za brisanje jednog ili više zapisa iz tablice. Sintaksa je `DELETE FROM [naziv_tablice] WHERE [uvjet];`.

3.2.3. DQL

Jezik koji se koristi za izvršavanje upita nad podacima iz tablica. Za dohvaćanje podataka koristi se `SELECT` naredba. Sintaksa je `SELECT [naziv(i)_stupca] FROM [naziv_tablice] WHERE [uvjeti] |opcionalni_elementi|`. Kada se upit izvrši, kreira se privremena tablica koja sadrži dohvaćene podatke te se ista prikazuje ili šalje prema vanjskom programu. Za potrebe ovog rada opcionalni elementi se neće koristiti tako da su u nastavku nabrojeni samo neki od njih. Opcionalni elementi mogu biti `ORDER BY` koji služi za sortiranje, `GROUP BY` koji služi za grupiranje podataka, `HAVING` koji služi za filtriranje nad grupiranim podacima te agregirajuće funkcije `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`.

3.2.4. DCL

Jezik koji sadrži naredbe za upravljanje pravima i dozvolama u bazi podataka. Naredbe koje se mogu koristiti su:

- `GRANT` - koristi se za davanje prava pristupa korisnicima. Sintaksa je `GRANT naziv_prava ON objekt TO korisnik;`.
- `REVOKE` - koristi se za uklanjanje prava pristupa korisnicima. Sintaksa je `REVOKE naziv_prava ON objekt FROM korisnik;`

Možemo zaključiti kako je SQL vrlo moćan jezik koji sadrži sve potrebne naredbe za rad s bazama podataka. U sljedećem poglavlju saznat će se na koji su način strukturirani podaci u relacijskim bazama podataka.

3.3. Entiteti i veze između njih

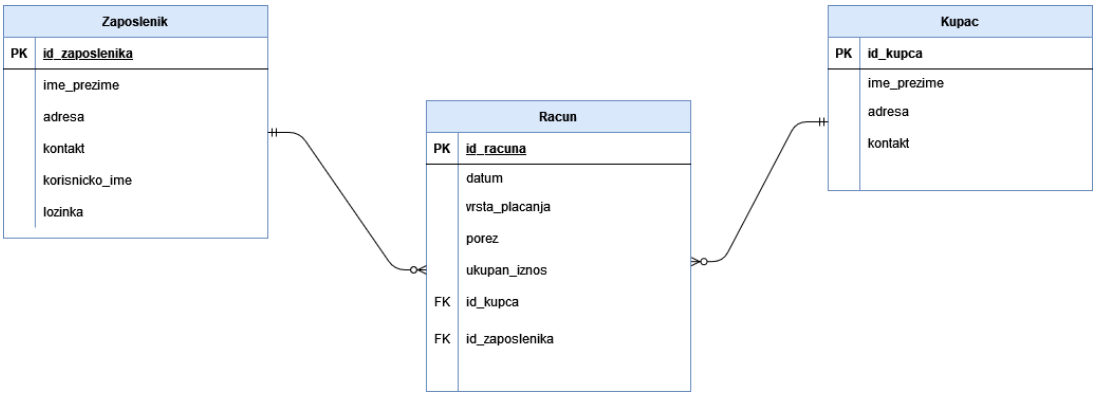
Kada se govori o relacijskim bazama podataka potrebno je poznavati tri različita pojma vezana uz relacijsku bazu podataka, a to su: entitet, atribut i veze. Entitet je objekt ili komponenta podataka. Entitet može biti objekt iz stvarnog svijeta koji postoji, primjerice, student na fakultetu. Nadalje, s entitetom se veže i pojam atribut. Atribut opisuje svojstvo entiteta. Već je bilo rečeno da je entitet student, atribut vezan uz taj entitet bi mogao biti njegovo ime i prezime ili godina studija. Većina baza podataka sadrži više entiteta koji se povezuju pomoću veza. Veze prikazuju odnose među entitetima.

Postoje tri vrste veza između entiteta, s obzirom na brojnost, a to su [3]:

- 1:1 - jedan zapis u tablici A je povezan s točno jednim zapisom u tablici B
- 1:N - jedan zapis u tablici A može biti povezan s više zapisa iz tablice B
- N:M - više zapisa u tablici A može biti povezano s više zapisa u tablici B

Uz to, u vezama imamo i opcionalnost koja nam govori mora li neki entitet sudjelovati u vezi ili ne.

Na slici 3 može se vidjeti primjer veza između entiteta. Entiteti su u ovom slučaju zaposlenik, račun i kupac. Svaki entitet ima svoje atribute i svoj jedinstveni primarni ključ po kojemu se mogu identificirati, u ovom slučaju je to njihov `id`. Što se tiče veza između ovih entiteta, svaki zaposlenik može izdati nula ili više računa, kao što i kupac može doći u trgovinu više puta kroz dan i obaviti više kupnji, time dobiti više računa ili ne dobije niti jedan račun, ako ništa ne kupuje. S druge strane, ako je račun izdan, odnosno kupac je kupio proizvode, tom računu odgovara točno jedan kupac koji je kupio određene proizvode i točno jedan zaposlenik koji je taj račun izdao te zbog toga oni moraju obavezno sudjelovati u toj vezi. Potrebno je naglasiti da bi se ova veza između entiteta mogla uspostaviti, bilo je potrebno pronaći vrijednost atributa u tablici zaposlenik koji odgovara vrijednosti atributa u tablici račun. U ovom slučaju je to `id` zaposlenika. U tablici račun `id` zaposlenika je vanjski ključ, vanjski ključ je ključ koji povezuje dvije tablice te je on ujedno i primarni ključ druge tablice (u ovom slučaju primarni ključ tablice zaposlenik).



Slika 3: Primjer veza između entiteta

Ovime je završeno poglavlje o relacijskim bazama podataka. One su danas najkorištenija vrsta baza podataka prvenstveno zbog načina na koji se podaci spremaju (tablice) te vrlo moćnog SQL-a koji omogućava izvršavanje različitih akcija nad podacima i samom bazom podataka. U sljedećem pogljavu će se dati uvod u nerelacijske baze podataka.

4. Nerelacijske baze podataka (NoSQL)

Kad je riječ o bazama podataka, relacijske baze podataka su dugo vremena predstavljale standard koji se koristio. Takve baze podataka su imale dosta prednosti, ali i nekih nedostataka. Jedan od bitnih nedostataka jest rukovanje s velikim količinama podataka (engl. *big data*) te ograničena mogućnost rada sa složenijim vrstama podataka, poput nestrukturiranih podataka. Sredinom 2000-ih Internet je postao iznimno popularan, a relacijske baze podataka jednostavno nisu mogle držati korak s protokom informacija koje su zahtijevali korisnici, kao i većom raznolikošću tipova podataka koji su nastali ovom evolucijom. Sve je to dovelo do nerelacijskih baza podataka [5].

4.1. Povijest nerelacijskih baza podataka

Skraćenicu NoSQL prvi je puta upotrijebio Carlo Strozzi 1998. godine imenujući, svoju laganu "relacijsku" bazu podataka otvorenog koda koja nije koristila SQL, *Strozzi NoSQL*. Riječ je o SUBP koji je *shell-based*, a radi pod operativnim sustavima nalik Unixu. Njegov naziv NoSQL se koristi zbog činjenice da sustav ne koristi SQL tijekom izrade upita za rad s bazom podataka. Naravno, NoSQL tada se bitno razlikuje od onog koji se koristi danas. Izraz NoSQL može značiti *No SQL systems* (nema SQL sustava) ili uobičajeniji *Not Only SQL* (ne samo SQL), kako bi se naglasila činjenica da neki sustavi mogu podržati jezike upita slične SQL-u [5]. Početkom 2000-ih NoSQL se kreće ubrzano razvijati. Tada nastaju graf-baze podataka *Neo4j*, objekt-baza podataka za Javu i .NET *db40*, dokument-baza podataka *CouchDB* i Googleova ključ-vrijednost baza podataka *BigTable*. U drugoj polovici 2000-ih nastaju dokument-baza podataka *MongoDB*, ključ-vrijednost baza podataka *Redis* i *Cassandra*.

Posljednjih desetak godina s ekspanzijom računarstva u oblaku (engl. *cloud computing*), pojavili su se problemi s uslugama koje koriste Internet i koji zahtijevaju da velika količina podataka dolazi u prvi plan te da se intenzivno koriste ti podaci. Za tvrtke poput Facebooka i Google-a, koje gledaju na web kao ogromno spremište podataka, obrada tih podataka putem SURBP-a pokazala se nedovoljno učinkovita s više aspekata. Relacijske baze moraju poštovati određen set pravila u relacijskom modelu koji ograničava fleksibilnost, pogotovo kod velike količine nestrukturiranih podataka te je s većim radnim opterećenjem, s kojom se relacijske baze ne slažu, potrebno povećati performanse. Relacijske baze podataka koriste vertikalno skaliranje te je za potrebe povećanja performansi potrebno povećati performanse u jednom stroju. To može biti skupa operacija te također ograničavajuća u prostoru. Uvođenjem horizontalnog skaliranja, koristi se veći broj manjih strojeva koji zajedno čine mrežu strojeva (klaster). U tu svrhu nastaju baze podataka u oblaku (engl. *cloud*), čija arhitektura i način obrade su drugačiji. To ujedno znači da oni zahtijevaju i drugačiji način integracije zbog kojeg dolazi do bavljenja s velikom količinom podataka. Pojam "velika količina podataka" ne znači nužno samo velika količina podataka u svom obujmu, nego i o kompleksnoj količini koja se sastoji od različitih podatkovnih struktura koje uključuju strukturirane, polu-strukturirane ili nestrukturirane podatke [5].

Budući da se u mreži strojeva radi o distribuiranom sustavu tu postoje i neki nedostaci.

CAP teorem nam govori da nije moguće garantirati da tri željena svojstva budu zadovoljena u istom trenutku, već samo dva te zbog toga arhitekt sustava mora znati kompromis kojeg odabire kod korištenja ovakvih sustava. Ta svojstva su:

- Konzistentnost (engl. *Consistency*) - svi čvorovi imaju iste kopije podataka za transakcije
- Dostupnost (engl. *Availability*) - svaki zahtjev za čitanje ili pisanje će biti izvršen ili odbačen
- Tolerancija particija (engl. *Partition tolerance*) - sustav nastavlja s radom čak i u slučaju greške u mreži

Nerelacijske baze podataka (u nastavku NoSQL) su mogle zadovoljiti sve potrebne kriterije za održavanje i rad s okolinom u oblaku za razliku od tradicionalnih SUBP. Zbog toga danas NoSQL predstavlja bolju alternativu za obradu velikih količina podataka, a da pritom osigura potrebnu skalabilnost.

4.2. Vrste nerelacijskih baza podataka

Kako bi se nerelacijske baze podataka bolje razumjele, potrebno ih je podijeliti u više kategorija. Što se tiče vrsta nerelacijskih baza podataka, njih možemo podijeliti u četiri kategorije, a to su [6]:

- **Ključ-vrijednost sustavi** (engl. *Key-value systems*) - spremnici ključ-vrijednost su poprilično pojednostavljeni, ali su učinkovit i moćan model. Imaju jednostavno aplikacijsko sučelje (engl. *API*) u kojem ključ-vrijednost spremište dopušta korisniku da pohranjuje podatke na manje shemski način, koristeći indekse i vrijednosti podataka.
- **Stupčane baze podataka** (engl. *Columnar databases*) - kod stupčanih sustava naglasak je na stupcima, odnosno spremnici u ovoj vrsti NoSQL-a su zapravo hibridni red/stupac spremnici koji pohranjuju podatke. Iako dijeli koncept skladištenja podataka *column-by-column*, ova vrsta ne pohranjuje podatke u tablicama nego u masovno distribuiranim arhitekturama gdje je svaki ključ povezan s jednim ili više stupaca (atributa).
- **Graf-baze podataka** (engl. *Graph-oriented Store databases*) - vrste baza podataka koji pohranjuje podatke u obliku grafa. Graf se sastoji od čvorova i veza, gdje čvorovi djeluju kao objekti, a veze kao odnos između objekata. Koristi tehniku koja se zove *index free adjacency* , što znači da se svaki čvor sastoji od izravnog pokazivača koji pokazuje na susjedni čvor.
- **Dokument-baze podataka** (engl. *Document databases)*- odnosi se na baze podataka koje pohranjuju podatke u obliku dokumenata. Takav način spremanja podataka nudi odlične performanse i mogućnost horizontalne skalabilnosti. Dokumenti unutar dokument-baze podataka donekle su slični zapisima u relacijskim bazama podataka, ali su mnogo više fleksibilniji s obzirom na to da ne koriste shemu.

Detaljnije će se svaka od gore navedenih vrsta nerelacijskih baza podataka opisati u sljedećim poglavljima.

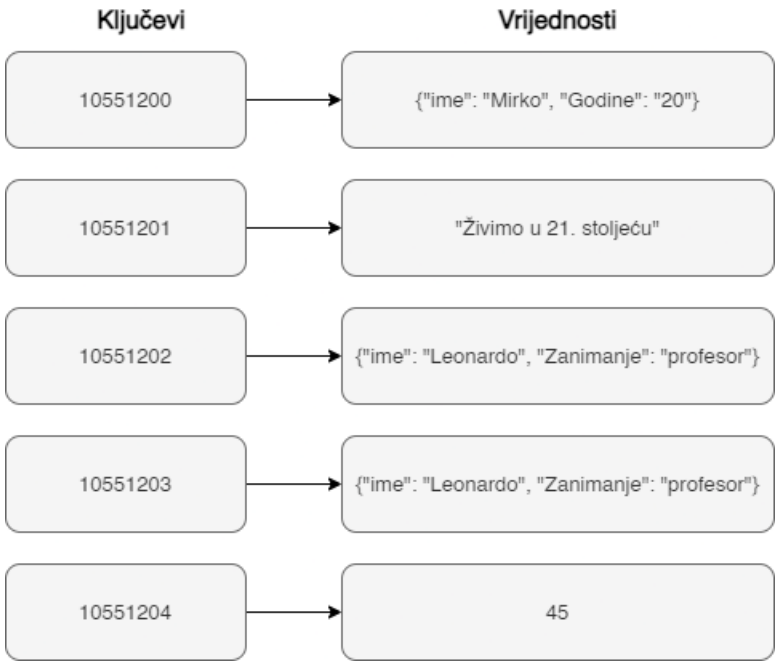
4.3. Ključ-vrijednost sustavi

Model ključ-vrijednost baze podataka je najjednostavnija vrsta nerelacijskih baza podataka. Riječ je o jednostavnoj hash tablici, koja se koristi prvenstveno kada je sav pristup bazi podataka preko primarnog ključa. Pritom je potrebno napomenuti, da baza podataka sprema podatke kao kolekciju parova ključeva s pripadajućom vrijednosti [5]. Zamislite tipičnu tablicu u SURBP-u s dva stupca, na primjer, `stupac_id` i `ime`, pri čemu `stupac_id` predstavlja ključ dok stupac `ime` pohranjuje vrijednost. U SURBP-u stupac `ime` bio bi ograničen za spremanje podataka tipa string. U ovom slučaju je moguće imati ključ i vrijednost, a pritom zadržati ih kao par. Ukoliko `stupac_id` ne postoji tada se stvara novi unos, a ako već postoji tada se nova vrijednost prepisuje preko trenutne vrijednosti [5].

Klijent koji koristi navedeni mehanizam može dobiti vrijednost za ključ, staviti vrijednost za ključ ili izbrisati ključ iz spremišta podataka. Vrijednost se pohranjuje u obliku zbirke binarnih podataka, odnosno binarno velikog objekta (engl. *Binary large object - BLOB*) u spremište podataka. Budući da model ključ-vrijednost uvijek koristi primarni ključ za pristup bazi, općenito ima izvrsne performanse i lako ga je skalirati. Neke od popularnih baza podataka koje koriste navedeni model su Riak, Redis, Memcached DB, Berkeley DB, HamsterDB i Amazon DynamoDB [7].

U nekim modelima ključ-vrijednost, poput Redisa, vrijednost koja se sprema ne mora nužno biti objekt, može biti bilo koja struktura podataka. Redis tako na primjer podržava spremanje popisa, skupova i raspona. Sve ove značajke omogućuju da se Redis koristi na više načina, za razliku od standardnog sustava ključ-vrijednost.

Na slici 4 je prikazan primjer jednostavne tablice gdje se koriste ključevi s njima pripadajućim vrijednostima. Kada gledamo par ključ-vrijednost potrebno je istaknuti da ključ uvijek ima jedinstveni identifikator. Tako se omogućuje pristup vrijednosti koja je povezana s tim jedinstvenim ključem. Stoga možemo vidjeti da je ključ u ovom primjeru broj 10551200 dok mu je pripadajuća vrijednost string "Živimo u 21. stoljeću".



Slika 4: Primjer ključ-vrijednost modela

4.3.1. Karakteristike ključ-vrijednost baze podataka

Svaka baza podataka ima svoje karakteristike po kojima se razlikuje od ostalih pa tako i ključ-vrijednost baza podataka. Karakteristike ključ-vrijednost baze podataka su [7]:

- **Skalabilnost** - jedna od najvećih prednosti u usporedbi s relacijskim bazama podataka je činjenica da su spremišta ključ-vrijednost beskonačno skalabilna na horizontalni način. U usporedbi s relacijskim bazama podataka gdje je proširenje okomito i konačno, ovo može biti velika prednost za složene i veće baze podataka.
- **Jednostavnost** - zbog korištenja minimalne podatkovne strukture. Tako na primjer, ako želimo spremiti informacije o korisnicima, moguće je to napraviti preko SURBP-a, ali koristeći ključ-vrijednost bazu podataka. Podaci se spremaju u objekt koji predstavlja vrijednost kojemu se dodijeli određen ključ. Tako se proces znatno olakšao te se nisu trebali nigdje definirati tipovi podataka koji se spremaju niti definirati shema u SQL-u.
- **Dosljednost** - primjenjivo samo za operacije na jednom ključu, budući da su te operacije `GET`, `PUT` ili `DELETE`.
- **Brzina** - zbog svoje jednostavnosti one omogućuju visoku razinu propusnosti. Brzina im se očitava činjenicom da spremaju podatke u radnu memoriju gdje s njima i rade, a ukoliko usporedimo čitanje i pisanje vrijednosti u radnu memoriju s čitanjem i pisanje po tvrdom disku, poznato je da je rad s memorijom višestruko brži.
- **Mobilnost** - budući da nemaju strukturni jezik upita, ključ-vrijednost spremišta se lako premještaju iz jednog sustava u drugi bez potrebe za novom arhitekturom ili promjenom

kôda. Kao takav, prelazak sa starog operacijskog sustava na novi ne uzrokuje ozbiljne poremećaje kao što bi to bio slučaj s relacijskom bazom podataka.

Nakon što su objašnjene karakteristike ključ-vrijednost baza podataka, postavlja se pitanje gdje se one koriste? Njihova uporaba će biti detaljnije objašnjena u sljedećem poglavlju.

4.3.2. Uporaba ključ-vrijednost baza podataka

Ključ-vrijednost baze podataka se mogu koristiti za različite situacije, ali često se koriste kada je potrebno pohraniti podatke o web sesijama. Općenito gledano, svaka je web sesija jedinstvena i prilikom početka sesije dodjeljuje joj se `sesija_id` vrijednost. Aplikacije koje pohranjuje `sesija_id` na tvrdi disk ili u SURBP imat će veliku korist ako pređu na model ključ-vrijednost, budući da se sve o sesiji može pohraniti jednim PUT zahtjevom ili dohvatiti pomoću GET zahtjeva. Takav rad s jednim zahtjevom čini ga vrlo brzim jer je sve o sesiji pohranjeno u jednom objektu. Nadalje, ključ-vrijednost baze podataka koriste se prilikom pohranjivanja postavki korisničkih računa. Gotovo svaki korisnik ima svoj jedinstveni identifikator, korisničko ime ili druge atribute, kao i postavke kao što su vremenska zona, jezik, kojim korisnik ima pristup. Sve se to može staviti u objekt, pa dobivanje preferencija korisnika može se dobiti jednostavnim GET zahtjevom. Ključ-vrijednost baze podataka se također koriste prilikom prikazivanja prilagođenih oglasa korisnicima na temelju njihovog profila [7].

Na sličan način se ključ-vrijednost baze podataka koristi tijekom e-trgovina. Korisnik kada kupuje preko interneta uvijek ima pristup košarici, kao što želimo da nam ta košarica uvijek bude dostupna (bilo to u preglednicima, za vrijeme trajanja sesija...), sve te informacije o kupnji se mogu staviti u vrijednost gdje je ključ korisnički identifikator. Još jedna dobra uporaba ključ-vrijednost baza podataka jest kada dolazi do privremenog povećanja kupovine proizvoda, kao na primjer povećanje kupovine tijekom blagdana (za Božić ili Crni petak). Umjesto da trgovine ulažu u infrastrukturu koja se neće koristiti tijekom cijele godine, brza i jednostavna skalabilnost ključ-vrijednost sustava omogućuje trgovinama kupnju jednog ili više privremenih fragmenta (engl. *database shards*) kako bi se proširio prostor za pohranu za pomoć u obradi podataka tijekom sezonskih valova povećane kupnje.

Ključ-vrijednost baze podataka su samo jedna podvrsta nerelacijskih baza podataka, u nastavku će detaljnije biti opisane stupčane baze podataka.

4.4. Stupčane baze podataka

Stupčano usmjerene baze podataka podržavaju podatke kao stupce umjesto redaka, kako bi se optimizirali upiti nad velikim skupovima podataka. Upiti nad redovima zahtijevaju memoriju i zahtijevaju veliki pristup disku, posebno ako svaki redak ima mnogo stupaca. Stupčano usmjerena baza podataka ima stupce koji su grupirani u obitelji (engl. *column family*), a svaka obitelj stupaca može imati neograničen broj stupaca. Na taj način je puno lakše postaviti upit svim stupcima za sve retke [8].

Stupčane baze podataka pohranjuju podatke u grupiranim stupcima, a ne u redove podataka. One koriste koncept koji se naziva prostor ključava (engl. *keyspace*). Prostor ključeva može sadržavati obitelji stupaca ili super stupce (engl. *super column*), svaki super stupac sadrži jednu ili više obitelji stupaca, a svaka obitelj stupaca sadrži najmanje jedan stupac ili više redaka stupaca. Svaki redak u obitelji stupaca ima jedinstveni ključ, a svaki stupac unutar retka sadrži naziv, vrijednost i vremensku oznaku. Koncept prostor ključeva je sličan shemi u relacijskom modelu, a ključna razlika je u tome što se u relacijskom modelu koriste tablice, a u ključnom prostoru se koriste obitelji stupaca ili super stupci [8].

Najpoznatije stupčano usmjerene baze podataka su Googleove BigTable, HBase i Cassandra. Nakon što su opisani dijelovi stupačnih baza podataka u sljedećem poglavlju će se na konkretnim primjerima objasniti njihov način rada.

4.4.1. Način rada stupčanih baza podataka

Sustavi pohrane podataka u obliku stupaca potpuno okomito particioniraju bazu podataka u zbirku pojedinačnih stupaca koji se pohranjuju zasebno. Svaki stupac se sprema zasebno na disk što omogućuje da se unutar baze podataka izvrše upiti koji čitaju samo potrebne atribute, umjesto da se čitaju svi redci s diska. Slična je korist i pri prijenosu podataka od glavne memorije do registara procesora, poboljšavajući ukupnu iskoristivost raspoloživog I/O (engl. *input/output*) i memorijskog opsega. Budući da se podaci obično čitaju iz memorije i upisuju u memoriju u blokovima, pristup orijentiran prema stupcima znači da svaki blok koji sadrži podatke za tablicu, sadrži podatke za jedan od stupaca [8].

Na slici 5 imamo primjer stupčane baze podataka koja sadrži više obitelji stupaca. Na prvi pogled, ova baza podataka izgleda kao relacijska baza podataka, razlika je u tome na koji način se ona čita.

Uzmimo za primjer da želimo dohvatiti sve nogometne klubove koji se nalaze u *primier* ligi. Relacijska baza podataka bi čitala podatke po redovima s lijeva na desno dok ne dođe do kraja. Iako bi upit nad ovom bazom podataka rezultate dobio brzo, zamislite da imate istu takvu bazu koja sadrži milijun redova s podacima, tada bi vrijeme izvršavanja upita trajalo mnogo duže, što nije praktično!

Id	Liga	Naziv_igraca	Nogometni_klub	Pozicija	Odigrane_utakmice
1	Premier Liga	Gabriel Martinelli	Arsenal	Napadač	57
2	Premier Liga	Mateo Kovačić	Chelsea	Vezni	115
3	Premier Liga	Trent Alexander-Arnold	Liverpool	Branič	161
4	Premier Liga	David de Gea	Manchester United	Vratar	377
5	Premier Liga	Harry Kane	Tottenham Hotspur	Napadač	212
6	Premier Liga	Benjamin Mendy	Manchester City	Branič	50
7	Major League Soccer	Jonathan Bond	Los Angeles Galaxy	Vratar	55
8	Major League Soccer	Carlos Darwin Quintero	Houston Dynamo	Napadač	59

Slika 5: Stupčana baza podataka s više obitelji stupaca

Sada pogledajmo ponovno istu tu bazu i izvršimo isti upit, ali iz ugla stupčane baze podataka. Ona će prvo pročitati stupac `liga` te zatim krenuti na stupac `naziv_igraca`, ali će

prepoznati da joj taj stupac nije potreban za traženi upit pa će ga preskočiti te ponovno pročitati stupac `nogometni_klub` i tu stati. Čitat će podatke u traženim stupcima odozgo prema dolje. Postavlja se pitanje, kako računalo zna koju ligu treba dodijeliti kojem nogometnom klubu kada dohvaća podatke tako? Način na koji to radi je da stupčane baze podataka svakom retku dodjeljuje jedinstveni ključ, što omogućuje da se brzo upare više stupaca. U ovom slučaju je jedinstveni ključ `row_id`.

Možemo vidjeti da ovakav pristup čitanja podataka vrlo koristan. Tako kada se dohvaća stupac `nogometni_klub`, baza prepoznaje da u stupcu `liga` postoje samo 6 zapisa koja su u *primer* ligi. Što znači da kada krene čitati stupac `nogometni_klub`, pročitat će samo timove od 1 do 6 jer su oni jedini koji se nalaze u *primer* ligi, a ostale neće gledati što rezultira velikom uštedom vremena. Naravno to nisu jedine prednosti, u nastavku će biti navedene ostale prednosti i nedostaci stupčanih baza podataka.

4.4.2. Prednosti i nedostaci stupčanih baza podataka

Prednost stupčanih baza podataka uključuju kompresiju podataka, visoke performanse sa zbirnim funkcijama (ZBROJ, COUNT, itd.) te skalabilnost [8]. Njihova prednost se očituje najviše kada analiziraju veliku količinu podataka ili kompleksnije podatke jer su u odnosu na SURBP mnogo brži. Još jedna prednost stupčanih baza podataka je mogućnost kompresije. Podaci u jednom stupcu uvijek su iste vrste, na primjer string ili integer. Budući da su svi podaci u stupcu jedan do drugog i iste vrste oni se mogu kompresirati učinkovitije.

Međutim, pisanje novih podataka u stupčane baze podataka može zahtijevati više vremena. Tako se podaci u bazu podataka koja se temelji na redovima mogu upisati samo u jednoj operaciji, ali u bazi podataka temeljenoj na stupcima potrebno je pisati svaki stupac zasebno, jedan po jedan. Stoga su stupčane baze podataka bolje za obradu podataka s malim brojem stupaca i većim brojem redaka.

4.5. Graf-baze podataka

Prije nego što objasnim što je graf-baza podataka, potrebno je navesti fundamentalne koncepte od kojih se sastoje graf-baze podataka, a to su:

- **čvorovi** - glavni entitet u graf-bazama podataka, možemo ih zamisliti kao redove u relacijskoj bazi podataka
- **veze** - veze između čvorova (entiteta). To bi bili vanjski ključevi u relacijskoj bazi podataka.
- **svojstva čvorova** - parovi ključ-vrijednost pohranjeni unutar čvorova ili veza.
- **oznake** - koriste se za grupiranje čvorova u skupove.

Graf-baze podataka koriste se za pohranu mrežnih podataka koji se sastoje od čvorova, veza između čvorova, svojstva čvorva i oznaka. Primjer mrežnih modela uključuje društvene

mreže, cestovne mreže i IT mreže gdje su entiteti visoko povezani. Graf-baza podataka koristi strukture grafova za semantičke upite s čvorovima, vezama i svojstvima za predstavljanje i pohranu podataka. Ključni koncept cijelog sustava je graf. Graf povezuje stavke podataka u spremište s kolekcijom čvorova i vezama, a veze predstavljaju veze između čvorova. Veze omogućuju izravno povezivanje podataka u spremištu [9].

U relacijskim bazama podataka entiteti su pohranjeni kao tablice. Upiti o odnosima među entitetima su učinjeni izvođenjem JOIN-a nad tim tablicama, operacijom koja zahtijeva puno računanja i memorije. Nasuprot tome, u NoSQL bazi podataka koja omogućuje spremanje grafova prioritet su veze. Svaki entitet sadrži popis zapisa veza povezanih s drugim čvorovima. Prilikom izvođenja upita kako bi se pronašle veze među čvorovima, baza podataka će koristiti taj popis za pronalaženje čvorava kako bi se skratilo vrijeme pretraživanja.

Kod graf-baza podataka, veze između podataka imaju najveći prioritet. Upiti kojima se pregledavaju veze su brzi jer su stalno pohranjeni u radnoj memoriji. Graf-baze podataka namjenski su izgrađene za spremanje i kretanje između veza. Veza uvijek sadrži početni čvor, završni čvor, vrstu i smjer, te može opisati odnose između roditelja i djeteta i slično. Ne postoji ograničenje u broju i vrsti odnosa koje čvor može imati [9].

Primjeri graf-baze podataka su Neo4J, InfiniteGraph, JSON L i OrientDB.

4.5.1. Način rada graf-baza podataka

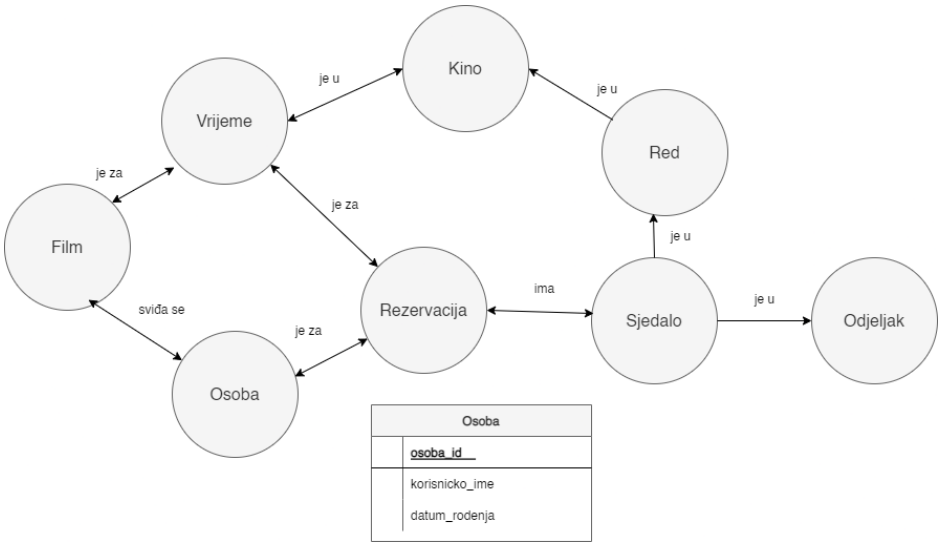
U graf-bazi podataka, povezani podaci su jednako (ili više) važni od pojedinačnih podatkovnih točaka. Takav pristup povezivanja znači da se veze i odnosi održavaju kroz svaki dio životnog ciklusa podataka unutar skalabilnog, pouzdanog sustava baza podataka [9]. Koristeći taj način, modeli podataka su jednostavniji od onih koji bi se proizveli s relacijskim bazama podataka ili drugim obliku nerelacijskih baza podataka. To također znači da aplikacija ne mora zaključivati podatkovne veze pomoću veza poput vanjskih ključeva.

Postoji mnogo načina na koje se čvorovi grafova mogu grupirati i odvojiti. Grafovi se mogu grupirati zajedno pomoću učinkovitih algoritama. Jedan takav algoritam naziva se hijerarhijsko grupiranje. Hijerarhijsko grupiranje je metoda koja strateški organizira povezane stavke u klastere. Krajnji rezultat je grupa kolekcija, svaka se razlikuje od ostalih, a stavke unutar svakog klastera približno su slične jedna drugoj. U nekim slučajevima, cijela graf-baza može se podijeliti u više čvorova u mreži u kojima je sve međusobno povezano. Strojno učenje može se primijeniti kako bi se utvrdilo gdje bi se podijele trebale dogoditi kako ne bi došlo do redundancije. Grafovi se mogu koristiti za generaliziranje podataka u strojnom učenju kako bi se utvrdile značajke, na primjer, imate čvor osobe i zanimanje, a sada možete generalizirati osobu na profesiju [9].

Određeni sustavi za upravljanje graf-bazama podataka koriste tehnologije dizajnirane za pohranu i upravljanje grafova. Drugi sustavi koriste relacijske, stupčaste ili objektno orijentirane baze podataka kao svoj sloj za pohranu. Neizvorna pohrana često je sporija od izvornog pristupa jer se sve grafičke veze moraju prevesti u drugi model podataka. Obrada grafova najučinkovitije je sredstvo obrade jer povezani čvorovi fizički ukazuju jedan na drugog u bazi poda-

taka. Neizvorni izvori za obradu grafova koriste druga sredstva za obradu operacija kreiranja, čitanja, ažuriranja i brisanja (CRUD) koje nisu optimizirane za rukovanje povezanim podacima [10].

Na slici 6 je prikazan jednostavan model grafa za rješavanja problema rezervacije mjesta u kinu. Pomoću takve sheme bi se rezervirala mjesta za gledanja filma na određenom odjeljku u tom kinu. Naravno svaki odjeljak ima i svoj red, svoj broj sjedala koji želimo rezervirati, ali može se vidjeti da je ova shema vrlo pregledna. Svaka linija, veza, s jednom ili više strelica predstavlja vezu ili odnos između čvorova i tvori usmjereni graf. Svojstvo ovog grafa je da daje ime mjestima gdje čvorovi i veze mogu imati svojstva, kao na primjer `ddatum_roenja` za čvor osoba.



Slika 6: Primjer graf-baze podataka (Morgante, 2020)

,

4.5.2. Najpoznatiji sustavi graf-baza podataka

Već sam ranije spomenula sustave graf-baza podataka, a u ovom poglavlju ću detaljnije opisati Neo4j i OrientDB. Razlog zašto sam izabrala baš ta dva sustava graf-baza podataka su jer je Neo4j najpoznatiji sustav, dok s druge strane OrientDB je malo manje poznatiji, ali prvi višemodelni sustav. Oba sustava pohranjuju podatke u izvornom obliku u obliku grafova te su besplatna za početnu razinu, iako oba sustava nude opcije koje se plaćaju koje su pogodnije za aplikacije većeg opsega.

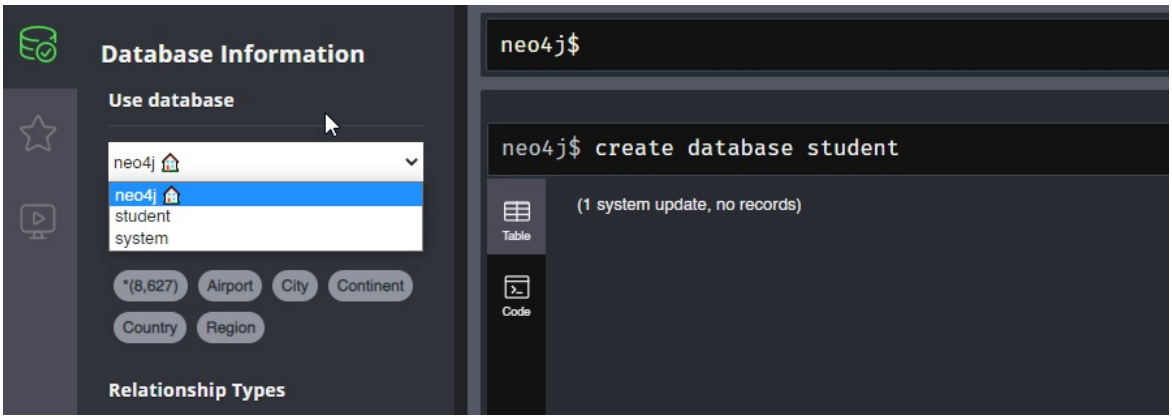
4.5.2.1. Neo4j

Neo4j je najpoznatija *open-source* graf-baza podataka koja je razvijena pomoću Java tehnologije. To je visoko skalabilna baza podataka bez shema.

Pruža fleksibilan, jednostavan, a opet moćan model podataka, koji se može lako promijeniti prema aplikacijama i industrijama Spring Data Neo4j, dio veće obitelji Spring Data, omogućuje jednostavno konfiguriranje i pristup Neo4j Graph bazama podataka iz Spring apli-

kacija. Pruža deklarativni jezik upita za vizualno predstavljanje grafova, koristeći ASCII sintaksu [10].

Neo4j pruža mogućnost korištenja sustava online putem preglednika. Kako bi se kreirala baza podataka, potrebno je napisati `create database [naziv_baze]` Na slici 7 se može vidjeti kako izgleda Neo4j korisničko sučelje. U lijevom djelu se nalaze sve kreirane baze dok se u sredini pišu naredbe. Nakon što se napiše upit, pritisne se tipka Enter te Neo4j pošalje obavijest ako je uspješno izvršio naredbu. Ako želimo raditi upite nad bazom student sa slike, potrebno se je prebaciti na nju tako da se ona označi u padajućem izborniku s lijeve strane ili se napiše upit koji je oblika `:use [[naziv_baze]]` koji nas automatski prebaci na potrebnu bazu.



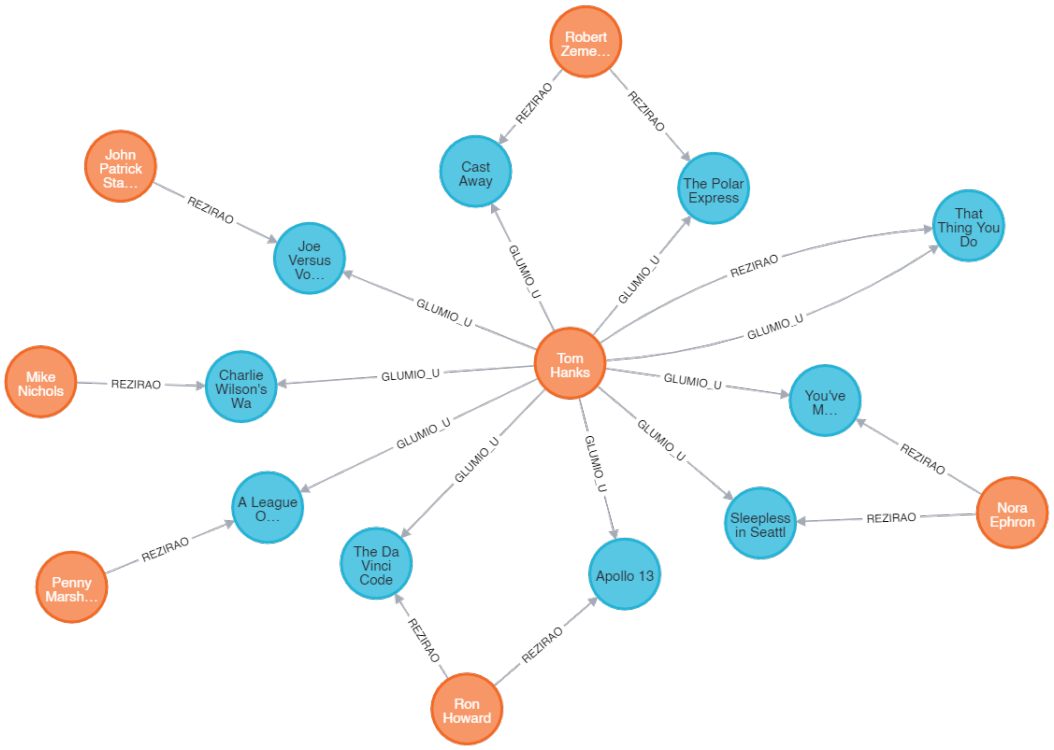
Slika 7: Primjer baze podataka u Neo4j

,

Ukoliko želimo izbrisati bazu podataka, koristi se naredba oblika `drop database [naziv_baze]`. Kako bi se kreirao čvor s oznakom koristi se navedena klauzula: `CREATE ([naziv_cvora]:[naziv_oznake] {[naziv_atributa]:[vrijednost_atributa]}`.

Tako ako bih htjela dodati čvor o sebi, on bi izgledao ovako: `CREATE (Sanja:Osoba {ime:'Sanja Šajfar', rodenje:1999})`. `MATCH` klauzula se koristi za traženje uzorka opisanog u njoj [12]. Tako bi upit koji bi nam vratio sve čvorove izgledao ovako, `MATCH (cvorovi)RETURN cvorovi`

Na slici 8 sam napravila graf-bazu podataka koja prikazuje filmove u kojima je glumio Tom Hanks koristeći Neo4j.



Slika 8: Primjer graf-baze podataka u Neo4j

,

4.5.2.2. OrientDB

OrientDB je prvi višemodelni *open-source* NoSQL SUBP koji kombinira moć grafova i fleksibilnost dokumenata u jednu skalabilnu operativnu bazu visokih performansi. OrientDB je osmišljen od temelja s performansama kao ključnom specifikacijom.

Nudi bolje korištenje RAM-a, može prelaziti različite dijelove ili cijela stabla i grafove zapisa u velikoj brzini, a da veličina zapisa ne predstavlja problem. Konkretnije, prelazi dijelove ili cijela stabla i grafove zapisa u milisekundama [10].

Za razliku od Neo4j, OrientDB ne nudi mogućnost isprobavanja sustava online, nego ga je potrebno prvo skinuti na računalo. Kako bi se kreirala baza podataka lokalno, koristi se sljedeća klauzula: `CREATE DATABASE LOCAL:/usr/local/orientdb/databases/test]` [13]. Ukoliko želite izbrisati bazu podataka koristite `DROP DATABASE [naziv_baze]`.

Ako želite kreirati čvor, sintaksa ide ovako: `CREATE VERTEX V1 SET name='mirko'`. Nakon što uspješno kreiramo čvor, sustav nam vraća poruku uspjeha: `Created vertex 'V1#14:1{name:mirko} v1' in 0.00350 sec(s).` Tako možemo dodati više čvorova, a da bi ih povezali potrebno je korsititi njihove brojeve koji su se generirali prilikom izrade. Tako na primjer, da bi povezali čvor `V1` s drugim čvorm, to bismo učinili na ovaj način: `CREATE EDGE FROM #14:2 TO #13:2`.

Ako usporedim moje iskustvo upotrebe Neo4j i OrientDB, mogu reći da mi je jasno zašto je Neo4j popularniji. Sama činjenica da nudi mogućnost isprobavanja sustava online bestplatno

je velika prednost, a osobno mi je i sintaksa bila lakša iako su sintakse oba sustava vrlo slične. Također mi je priručnik Neo4j bio bolji za čitati jer je imao više primjera (i grafičkih i tekstualnih).

Do sada sam objasnila gotovo sve podkategorije nerelacijskih baza podataka, a posljednja, ujedno i ona koju ću najdetaljnije opisati u nastavku jest dokument-baza podataka.

4.6. Dokument-baze podataka

Dokument-baze podataka (također poznate kao baze podataka temeljene na dokumentima ili spremište dokumenata) su najpoznatija vrsta nerelacijskih baza podataka. Riječ je o bazama podataka koje pohranjuju podatke u obliku dokumenata, zbog toga su i dokumenti glavni elementi ove vrste baze podataka. Dokument je uređeni skup ključeva s njima pripadajućim vrijednostima. Dokumenti se nalaze u kolekcijama koje tvore bazu podataka [6].

Kolekcije se mogu usporediti s tablicama u relacijskim bazama podataka gdje jedan dokument odgovara jednom retku u tablici. Bitna razlika dokument-baza podataka u odnosu na relacijske baze podataka jest u tome što dokument sustavi koriste dinamičke sheme. Dinamičke sheme omogućuju da svaki dokument unutar kolekcije može imati drugačiju strukturu. Stoga, dinamičke sheme omogućuju da se u kolekcije spremaju različite vrste podataka, bilo to strukturirani podaci, nestrukturirani podaci (tekst) ili polustrukturirani podaci (XML) [14].

Mnogi sustavi za dokument-baze podataka koriste JSON (engl. *JavaScript Object Notation*) format za spremanje podataka. JSON podržava sve osnovne tipove podataka: brojeve, nizove, Boolean, string ili objekte (u ovom slučaju su to JSON objekti). Također, sustavi za upravljanje dokument-bazama podataka često nude mogućnost dodavanja sekundarnih indeksa što rezultira bržom obradom upita. Takvi sustavi se često služe particioniranje to jest, zbog veće efikasnosti se podaci raspodjele na više servera ako se radi s većom količinom podataka. Nerijetko se zbog toga rade i kopije baza podataka na više servera. Najpoznatije primjeri takvih dokument-baza podataka su MongoDB, CouchDB, DocumentDB, SimpleDB, PostgreSQL, OrientDB, Elasticsearch i RavenDB

U nastavku će detaljnije biti objašnjen način rada dokument-baza podataka te njihova analiza.

4.6.1. Način rada i analiza dokument-baza podataka

Dokument-baza podataka po funkciji se značajno razlikuju od tradicionalnih relacijskih baza podataka. Relacijske baze podataka obično pohranjuju podatke u zasebne, povezane tablice (kako ih je definirao programer), dopuštajući pojedinačne objekte da se rasporede po nekoliko tablica. No, u dokument-bazama podataka sve informacije o danom dokumentu ili objektu pohranjene su u jednoj instanci. Nema potrebe za mapiranjem objekta i veza među njima pri učitavanju podataka u bazu podataka ili pri preuzimanju podataka iz baze. Iz tog razloga, dokument-baze podataka obično su brže, iako to ovisi o načinu na koji se koriste. Relacijske baze podataka bile bi brže da su svi potrebni podaci za upit sadržani u jednom retku tablice. U stvarnosti, upit mora pretraživati na nekoliko lokacija, dohvatiti podatke ugniježđene

u različite tablice, zatim ih sastaviti prije nego što izbaci rezultat, dok dokument-baze podataka sadrže sve potrebne podatke na jednom mjestu [14].

Kao što je već rečeno, dokumenti su u središtu dokument-baza podataka. Definicija dokumenta se može razlikovati ovisno o sustavu, ali možemo reći da svi dokumenti rade na istom principu, oni kodiraju enkapsulirane podatke (ili informacije) u jedan od standardiziranih formata/kodiranja. Vrste kodiranja mogu biti JSON, XML, YAML ili binarne vrste kao što je BSON.

Gledajući na dokumente kao koncept, oni se mogu grubo usporediti s konceptom objekt. U oba koncepta nema potrebe za krutim ili standardnim dizajnom i mogu se mijenjati ključevi, određeni dijelovi ili odjeljci. Spremišta dokumenata dopuštaju različite vrste dokumenta u svakom spremištu, a polja unutar svakog dokumenta su opcionalna. Često se može kodirati s različitim sustavima kodiranja. Na primjer, isječak kôda 1 daje dobar primjer dokumenta u JSON formatu.

```
{
    "ime": "Mirko"
    "prezime" : "Miric",
    "godine" : 25
 }
```

Isječak kôda 1: Primjer JSON objeta

Međutim, u istoj bazi se može nalaziti drugi dokument u XML formatu, kao u isječku kôda 2:

```
<kontakt>
    <ime>Harry</ime>
    <prezime>Potter</prezime>
    <broj>1235550888</broj>
    <adresa>
        <adresa1>4 Privet Dv.</adresa1>
        <grad>Little Whinging</grad>
        <postanskiBroj>31115</postanskiBroj>
        <drzava>UK</drzava>
    </adresa>
</kontakt>
```

Isječak kôda 2: Primjer XML objekta

U strukturnom sastavu svakog dokumenta u bazi postoji jedinstvo, ali se polja razlikuju. Informacije i dizajn dokumenta obično se naziva sadržajem dokumenta na koji se možemo pozivati uređivanjem ili dohvaćanjem. Relacijske baze podataka za usporedbu sadrže sva ista polja, ali su neka neiskorištena ako za njih nema podataka, dok dokumenti ne sadrže prazna polja. Novi se podaci mogu dodavati zapisima bez potrebe za ažuriranjem svakog drugog zapisa radi dijeljenja slične strukture. U smislu skalabilnosti, ako treba ažurirati baze dokumenata, to se može ograničiti na nove unose, a ne na cijelu bazu podataka [15].

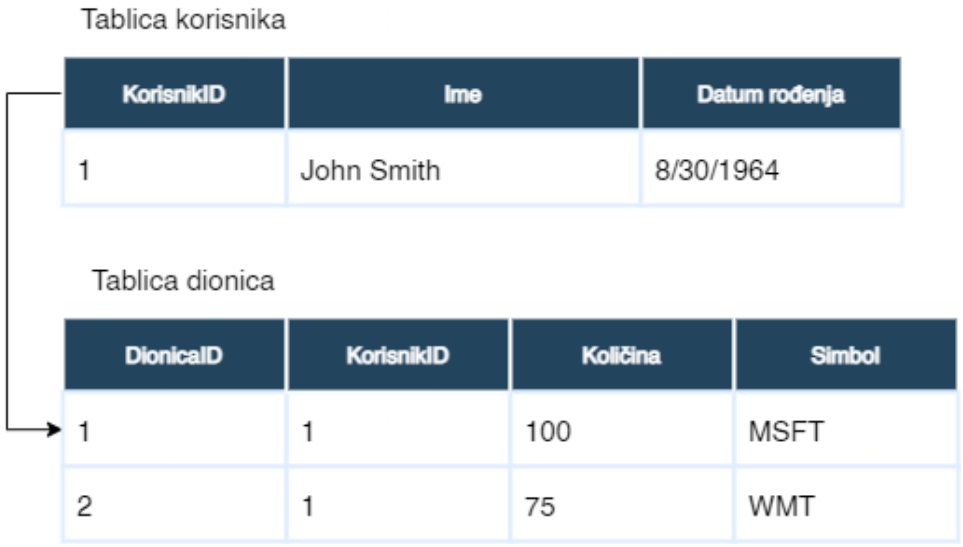
Kako bi se mogli dodavati, mijenjati, brisati i postavljati upiti, svaki dokument dobiva jedinstveni id. Identifikator nije osobito važan jer se za upućivanje na određeni dokument može koristiti cijeli put ili jednostavni niz brojeva. Prilikom upita nad podacima, traže se sami dokumenti, podaci se uzimaju izravno iz dokumenata, a ne iz stupaca unutar baze podataka [15].

Tako na primjer na slici 9 možemo vidjeti tri različita dokumenta koji se nalaze u istoj bazi podataka. Prvi dokument ima više različitih atributa, od kojih je jedno `ime`. Ipak, to ne sprječava drugi dokument da sadrži i koristi sličan atribut `punoIme`, a ne ponovno `ime`, jer ne postoje pravila o shemi. Slično, treći dokument pohranjuje atribut `punoIme` koje ima ugrađena različite atribute `ime` i `prezime`. Dokument-baza podataka može bez problema podržati ovakve dokumente. Naravno, ovo je samo primjer i obično ne bi imalo smisla miješati nazive atributa i oblike poput ovoga, samo zato što možete. Ovo je jedna od velikih prednosti dokument-baza podataka jer si sami možete razviti svoju shemu kako je potrebno.

Slika 9: Primjer baze podataka s dokumentima (Lobel, 2015)

SURBP koriste krutu strukturu podataka sa stupcima i redcima, gdje jedno umetanje stupca utječe na cijelu tablicu što nije prikladno za agilni razvoj koji zahtijeva prilagodljive procese i bržu isporuku softvera. Relacijski model može se previše pojednostaviti, struktura podataka gdje su samo stupci i redci nisu prirodan način za pohranu podataka. Dijeljenje podataka u stupce manje je fleksibilno od složenijih shema podataka koje se vide u dokument-bazama podataka. S druge strane, dokument-baze podataka usredotočene su na dokumente slične JSON-u. Takva spremišta podataka čine prirodno i fleksibilno rješenje za programere koji mogu brže raditi s agilnim softverom što rezultira visokom produktivnošću, zbog čega je model podataka orijentiram prema dokumenata u modernim NoSQL bazama podataka postao popularna alternativa tabličnim SURBP. Također, dokument-baze podataka nude mogućnost izmjenjivanja dizajna u bilo kojoj fazi bez temeljnog narušavanja strukture.

Na slici 10 možemo za primjer uzeti tablicu korisnika s imenom John i imamo zasebnu tablicu dionica s više redaka za sve Johnove dionice. Ove dvije tablice povezane su s `KorisnikID`, čija je vrijednost 1 za John-a. Ako bismo navedene tablice koristili u aplikaciji, za preuzimanje podataka o korisnicama dionica, potrebno je koristiti `JOIN` kako bismo dobili sve potrebne podatke. Sama aplikacija treba uzeti u obzir kako uzeti jedan izmijenjeni objekt (u ovom slučaju korisnik i njegove dionice) te promijeniti bazu podataka ažuriranjem korisnika i dionica tih korisnika. To zahtijeva znatne napore te nije toliko isplativo i korisno.



Slika 10: Primjer relacijske baze podataka korisnika i dionica (Lobel, 2015)

U dokument-bazama podataka obično se radi suprotno. Na slici 11 vidimo jedan dokument koji sadrži korisnika i njegovu imovinu. U ovom slučaju nema potrebe za `JOIN` prilikom izvođenja upita, niti razdvajanjem objekta na različita mjesta prilikom spremanja promjena. S modelom bez shema, eliminira se ogroman napor koju bi aplikacija inače imala da koristi relacijske baze podataka.

Dokument

```
{
"id": "1",
"ime": "John Smith",
"datumRodenja": "1964-30-08",
"dionice": [
    {"kolicina": 100, "simbol": "MSFT"},
    {"kolicina": 75, "simbol": "WMT"}
    ]
}
```

Slika 11: Primjer dokumenta unutar kolekcije unutar dokument-baze podataka (Lobel, 2015)

Upiti nad ovom vrstom baze podataka su mnogo moćniji u odnosu na upite nad relacijskim bazama podataka. Kompatibilnost JSON dokumenta je također jedna od prednosti korištenja ove vrste baza podataka. Kao standard neovisan o jeziku, čitljiv od ljudi i bez intenziteta podataka, JSON se naširoko koristi za razmjenu podataka i pohranu. Treba uzeti u obzir da dokument-baza podataka je podskup ili nadskup drugih nerelacijskih modela podataka, što znači da je moguće kodirati podatke kako god je potrebno u danom trenutku (bilo to par ključ-vrijednost, geoprostorni i vremenski podaci ili graf). Svaki je dokument neovisna jedinica, lakše se distribuira po poslužiteljima bez uništavanja podataka [14].

Potrebno je znati da tradicionalne, relacijske baze podataka, zahtijevaju postojanje polja za svaki dio informacija i svaki unos. Kada informacija nije dostupna, ćelija je prazna. Međutim, i dalje mora postojati u bazi podataka. . Dokument-baze podataka imaju mnogo fleksibilniju strukturu i ne zahtijevaju dosljednost. Takva baza podataka može obraditi velike količine nestrukturiranih podataka. Također, nove se informacije mogu mnogo lakše integrirati. Nema potrebe dodavati nova polja s podacima u svaki skup podataka, samo odgovarajuća u spremištu dokumenata [15].

Mogućnost rada s polustrukturiranim podacima omogućuju rješavanje izazova koji se teže mogu riješiti koristeći SURBP. Takav izazov bi bio kada baza podataka zahtijeva promjenu svog modela. Relacijske baze podataka zahtijevaju od programera da zatraže od administratora baze podataka da preuzme nadzor nad bazom, što znači da za uporabu ovih baza podataka treba više ljudi. Zbog toga novije tvrtke koje su još u brzom rastu i imaju manje zaposlenika, ne mogu si priuštiti takve baze podataka te zbog njihove neučinkovitosti s radom s nestrukturiranim podacima.

Međutim, baze podataka orijentirane prema dokumentima nemaju baš relacijski kapacitet. Spremišta dokumenata nisu prikladna da bi se mogla koristiti tako. To je glavni nedostatak baza podataka orijentiranih na dokumente. Tu se misli da dokument-baze podataka nisu namijenjene za korištenje relacija, tipa da se pokušaju povezati dokumenti po poljima jer će to brzo postati složeno i komplicirano. Ipak treba imati na umu da je riječ o podvrsti nerelacijskih baza podataka pa je jasno da nisu namijenjene relacijama. Također jedan od nedostataka dokument-baze podataka jest taj što imaju slabu atomarnost. Tako na primjer ako želimo napraviti promjene u dvije kolekcije to će zahtijevati pokretanje dva odvojena upita što krši pravilo atomarnosti, to ovisno o situaciji može predstavljati i pozitivnu stranu dokument-baza podataka.

4.6.2. Uporaba dokument-baza podataka

Mnogi programeri odabiru Oracle i SQL umjesto nerelacijskih baza podataka jer misle da se redci i stupci čitaju brže od dokumenata. Također, dokument-baze podataka nisu prikladne za uporabu ako se radi s podacima koji imaju puno relacija. Bez obzira na to, dokument-baze podataka u današnje vrijeme zauzimaju sve veći udio tržišta koje su prije zauzimale relacijske baze podataka [6].

Dokument-baze podataka su prikladne za korištenje kada se radi s podacima koji nemaju jednaku veličinu polja i koji se znatno razliku. Tada se ti podaci mogu pohraniti u obliku dokumenta gdje svaki dokument ima posebne karakteristike i posebnu veličinu. Ovakve baze podataka se mogu koristiti za sustave upravljanja sadržajem, softver za blog te se često koriste u aplikacijama vezanim uz zdravlje. Danas su ovakve baze podatak toliko često korištene za raznovrsna područja da je teško suziti njihovu uporabu.

Jedan primjer gdje se dokument-baze podataka koristi je eBay. eBay je multinacionalna tvrtka koja pruža platformu za prodaju proizvoda između ljudi. Za većinu podataka se koristi MongoDB kao oblik pohrane ili kategorizacije podataka (npr. kategorizacija proizvoda za prodaju ili za prijedloge za pretraživanje).

Kao što možemo vidjeti, dokument-baze podataka imaju mnogo više prednosti nego nedostataka. Većina nedostataka je vezana uz relacije među podacima što nije niti njihov primarni cilj. U nastavku će biti detaljnije objašnjen sustav za dokument-baze podataka, MongoDB.

5. Tehnologije i alati

U ovom poglavlju će ukratko biti objašnjen MongoDB i C#. Potrebno je poznavati osnove ovih tehnologija kako bi se razumio praktični dio rada, tj. aplikacija za čiju su izradu korištene obje tehnologije. Razlog zašto su izabrane baš te dvije tehnologije zbog njihove popularnosti i česte upotrebe. MongoDB je jedan od najpoznatijih sustava za upravljanje nerelacijskim bazama podataka, dok je C# jedan od najkorištenijih programskih jezika na svijetu.

5.1. MongoDB

MongoDB je sustav za upravljanje bazama podataka dizajniran za Internet i web aplikacije otvorenog koda. Riječ je o bazi podataka baziranoj na dokumentima. Model podataka i strategije postojanosti izgrađene su za visoku propusnost čitanja i pisanja. Model podataka o dokumentima u MongoDB-u olakšava izgradnju jer ima podršku za rad s nestrukturiranim podacima [17].

MongoDB-ovi dokumenti kodirani su u formatu sličnom JSON-u, nazvanom BSON. BSON prirodno odgovara modernom objektno orijentiranom programiranju. BSON se isprva čini sličnom BLOB-u (*binary large object*), ali postoji važna razlika, MongoDB baza podataka razumije interne BSON-ove. To znači da MongoDB može posegnuti unutar BSON objekta, čak i ugniježđenih objekata koristeći točkastu notaciju. Točkasta notacija znači da se može svojstvima objekta pristupiti tako da se navede naziv objekta, nakon čega slijedi točka, nakon koje slijedi naziv svojstva kojemu se želi pristupiti. To omogućuje MongoDB-u izgradnju indeksa i podudaranje objekata s izrazima upita na najvišoj razini i ugniježđenih BSON ključeva. MongoDB također podržava bogate upite i potpune indekse. To ga razlikuje od ostalih baza dokumenata u koji se zasebni poslužiteljski sloj koristi za rukovanje složenim upitima [17].

MongoDB je međuplatformska baza orijentirana prema dokumentima koja pruža visoke performanse, visoku dostupnost i jednostavnu skalabilnost što su jedne od prednosti korištenja baze podataka orijentirane prema dokumentima.

5.1.1. Način rada MongoDB-a

MongoDB radi na konceptu kolekcija i dokumenata. Baza podataka je fizički spremnik za kolekcije. Svaka baza podataka dobiva vlastiti skup datoteka na datotečnom sustavu. Jedan MongoDB poslužitelj obično ima više baza podataka [17].

Kolekcija je grupa MongoDB dokumenata. To je ekvivalent tablici u relacijskim bazama podataka. Kolekcija postoji unutar jedne baze podataka, pritom kolekcije ne primjenjuju shemu. Dokumenti unutar kolekcije mogu imati različita polja. Obično svi dokumenti u kolekciji imaju sličnu ili povezanu svrhu te imaju dinamičku shemu kako bi mogli sadržavati različite vrste podataka unutar svakog dokumenta [18].

Za lakše razumijevanje MongoDB-a na tablici 1 se vidjeti usporedba relacijskih baza podataka i MongoDB-a. Tako na primjer, slično tablicama u relacijskim bazama podataka u

MongoDB-u bi to bile kolekcije.

Relacijska baza podataka	MongoDB
baza podataka	baza podataka
tablica	kolekcija
redak	dokument
stupac	polje
spajanje tablica	ugrađeni dokumenti
primarni ključ	primarni ključ (zadan primarni ključ, pružan od MongoDB)

Tablica 1: Usporedba između RDBMS-a i MongoDB-a

5.1.2. Karakteristike MongoDB-a

Podaci se u MongoDB pohranjuju u obliku dokumenata u BSON formatu. Riječ je o najpopularnijem *open-source* programu koji radi s NoSQL baza podataka. MongoDB jest dobar zbog replikacije koji postiže korištenjem skupa replika. Replikacija je proces kopiranja podataka iz središnje baze podataka u jednu ili više baza podataka. Skup replika je skupina *mongod* instanci koje poslužuju isti skup podataka. *mongod* je primarni daemon proces za MongoDB sustav. Obrađuje zahtjeve nad podacima, upravlja pristupom podacima i izvodi operacije upravljanja u pozadini [18].

U replici je jedan čvor primarni čvor koji prima sve operacije pisanja. Sve ostale instance, sekundarne, primjenjuju operacije iz primarne tako da imaju isti skup podataka. Skup replika može imati samo jedan primarni čvor. Set replika je grupa od dva ili više čvorova (općenito za set su potrebna minimalno dva čvora). U setu replika jedan čvor je primarni čvor, a preostali čvorovi su sekundarni. Svi se podaci repliciraju s primarnog na sekundarni čvor. U vrijeme greške ili održavanja, bira se novi primarni čvor. Nakon oporavka neuspjelog čvora, taj čvor se ponovno pridružuje skupu replika i radi kao sekundarni čvor [18].

Jedna od karakteristika po čemu je MongoDB dobio na popularnosti jest *auto-sharding*. *Sharding* je postupak pohranjivanja zapisa podataka u više strojeva (više baza podataka) i to je MongoDB-ov pristup za rad s velikim podacima i rastom podataka. Kako se veličina podataka povećava, jedna baza podataka možda neće biti dovoljan za spremanje podatke niti pružiti prihvatljiv protok čitanja i pisanja nad bazom. Tu u igru dolazi *sharding*, koji rješava problem horizontalnim skaliranjem. S sharding, dodajemo još baza podataka za podršku rastu i količini podataka te zahtjevnim operacija čitanja i pisanja. MongoDB također nudi bogate upite nad bazom te brza ažuriranja [18].

5.2. C#

C# je *high-leve*l, općeniti programski jezik opće namjene koji se može koristiti za obavljanje širokog raspona zadataka i ciljeva koji se protežu kroz različite profesije. C# se prvenstveno koristi na Windows .NET okviru, iako se može primijeniti na platformu otvorenog koda. Ovaj vrlo svestrani programski jezik objektno je orijentiran programski jezik (OOP) koji je u

današnje vrijeme dosta popularan i korišten.

U usporedbi s dugogodišnjim jezicima poput Pythona i PHP-a, C# je mlad dodatak obitelji programera s gotovo dvadeset godina postojanja. Krivulja učenja za C# relativno je niska u usporedbi sa složenijim jezicima poput Jave, iako ga nije tako jednostavno naučiti kao Python. Jezik je 2000. godine razvio Microsoftov Anders Hejlsberg, danski softverski inženjer. Od svibnja 2021. C# je bio na 4. mjestu PYPL popularnosti programskog jezika, odmah iza Jave i JavaScript -a. Podaci korišteni za sastavljanje ovog indeksa temelje se na tome koliko često ljudi traže vodič u različitim programskim jezicima u Googleu [19].

C# je programski jezik visoke razine, što znači da je u usporedbi s jezicima niske razine jednostavniji za čitanje i pisanje, apstraktniji, što ga čini lakšim za upotrebu i odličnim izborom za početnike. Osim čitljivosti, C# se također može koristi za automatiziranje složenih zadataka koji zahtijevaju puno vremena za postizanje manjih rezultata. Ovaj programski jezik također je statistički upisan (engl. *statistically-typed*), što znači da se pogreške otkrivaju prije nego što se aplikacija pokrene. To uvelike olakšava otkrivanje malih nedostataka u kodu koje bi inače bili gotovo neprimjetne [19].

5.2.1. Za što se sve C# koristi?

Kao i drugi programski jezici opće namjene, C# se može koristiti za stvaranje niza različitih programa i aplikacija: mobilnih aplikacija, stolnih aplikacija, usluga temeljenih na oblaku (engl. *cloud-based*), web stranica, poslovnog softvera i puno igara. Iako je C# izuzetno svestran, postoje tri područja u kojima se najčešće koristi, a to su izrada web stranica, windows aplikacije i izrada igra [20].

C# se često koristi za razvoj profesionalnih, dinamičkih web stranica na .NET platformi ili softvera otvorenog koda. Budući da je ovaj jezik objektno orijentiran, često se koristi za razvoj web stranica koje su nevjerojatno učinkovite, lako skalabilne i jednostavne za održavanje. C# je stvorio Microsoft za Microsoft, pa je lako vidjeti zašto se najčešće koristi za razvoj Windows aplikacija za stolna računala. C# aplikacijama je potreban Windows .NET okvir kako bi funkcionirale u najboljem redu, pa je najsnažniji slučaj korištenja ovog jezika razvoj aplikacija i programa koji su specifični za arhitekturu Microsoftove platforme. Ovaj se jezik često koristi za stvaranje igara poput Rimworlda na Unity Game Engineu. Unity je daleko najpopularniji dostupan *engine* za igre na kojem je izgrađeno više od trećine najboljih i najčešće igranih igara u industriji. C# se besprijekorno integrira s Unity strojem i može se koristiti na gotovo svim modernim mobilnim uređajima ili konzolama zahvaljujući tehnologiji za više platformi poput Xamarina [20].

5.2.2. Prednosti C#-a

C# jezik je brz programski jezik, njegovo kompajliranje i vrijeme izvođenja je prebrzo. Vrlo je jednostavan jezik za korištenje. Daje strukturirani pristup razbijanju problema na dijelove. Također, ima bogat skup bibliotečkih funkcija i tipova podataka. Riječ je o objektno orijentiranom programskom jeziku, što ga čini lakšim za održavanje i razvoj u odnosu na proce-

duralno orijentirane programske jezike. Osim toga, C# programiranje podržava enkapsulaciju podataka, nasljeđivanje, polimorfizam, sučelja [20].

Koristeći C# ne može se izvesti nesigurna pretvorba podataka iz, na primjer, double u Boolean. Njegovi tipovi podataka (primitivni tipovi) inicijaliziraju se na nula, a referentni tipovi (objekti i klase) kompajler automatski inicijalizira na null. Jedna od prednosti jest i interoperabilnost. Interoperabilnost je proces koji omogućuje C# programima da učine gotovo sve što domaća C ++ aplikacija može učiniti. Ukratko, jezična interoperabilnost je sposobnost koda da stupi u interakciju s kodom koji je napisan pomoću drugog programskog jezika. Može pomoći maksimiziranju ponovne uporabe koda i, prema tome, poboljšati učinkovitost razvojnog procesa. C# jezik pruža podršku za korištenje COM objekata, bez obzira na to koji se jezik koristio za njihovo stvaranje. Međutim, također podržava i posebnu značajku koja omogućuje programu da pozove bilo koji izvorni API. Nadalje, skalabilan i ažuriran je [20].

U C#-u instaliran je vrlo učinkovit sustav koji prikuplja i briše smeće koje se automatski nalazi u sustavu. C# je vrlo učinkovit u upravljanju sustavom jer ne stvara nered u sustavu. Glavna prednost jezika C# je snažna sigurnosna kopija memorije. Programski jezik C# sadrži veliku rezervnu memoriju tako da se problem curenja memorije (engl. *memory leakage problem*) i druge takve vrste problema ne pojavljuju kao što se to događa u slučaju jezika C++. Nadalje, treba se spomenuti i bogata klasa biblioteka koje olakšavaju implementaciju mnogih funkcija. Uz to, aplikacija napisana na .NET-u imat će bolju integraciju i sposobnost tumačenja u usporedbi s drugim NET tehnologijama [20].

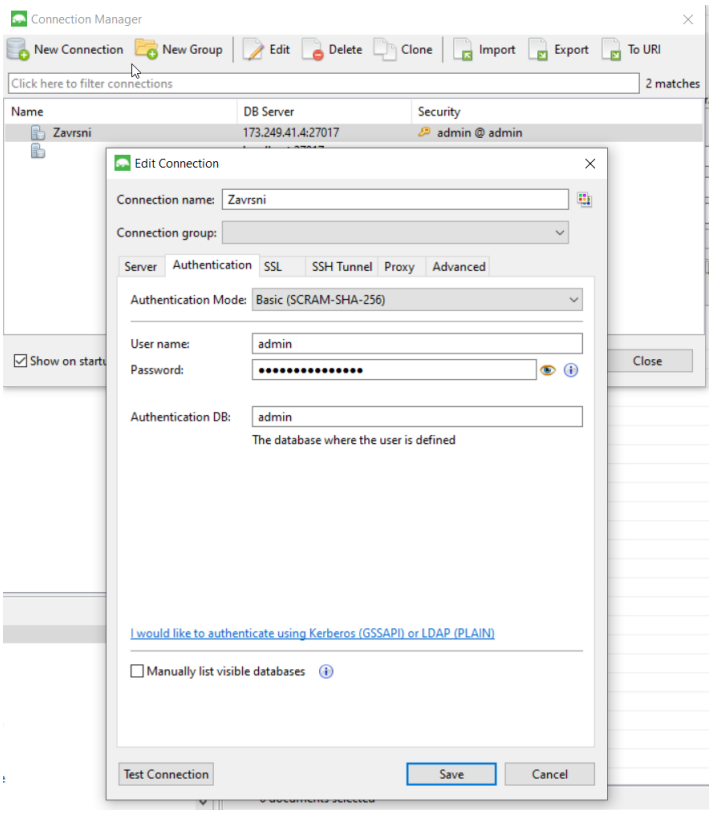
6. Implementacija aplikacije

U završnom djelu rada implementirat će se aplikacija koristeći MongoDB kao sustav za upravljanje dokument-bazama podataka. Za implementaciju aplikacije korišten je C# programski jezik. Sav programski kôd je pisan u Visual Studiju. Visual Studio je razvojno okruženje (engl. IDE – *integrated development environment*) za više programskih jezika (C++, F#, Python, Visual Basic) a u ovom radu će se prvenstveno koristiti za pisanje programski kôda u C#-u [21]. Potrebno je napomenuti da je tema završnog rada upotreba baza podataka temeljenih na dokumentima pri razvoju programskih proizvoda.

Aplikacija je kreirana kako bi prikazala sve podatke vezano uz kupce i kupovinu proizvoda. Unutar aplikacije nalaze se pet forma koje prikazuju kolekcije iz baze podataka koje sadrže informacije o kupcima, proizvodima koji se prodaju, kategorijama po kojima su proizvodi raspoređeni, informacije o prodanim proizvodima te plaćanju proizvoda. Aplikacija ima više funkcionalnosti. Nudi mogućnost prikaza podataka iz baze podataka u zasebnim formama. Na svakoj formi je prikazana druga kolekcija iz baze podataka, u određenim formama je omogućeno sortiranje ili pretraživanje kolekcije, dok se u drugima se nudi mogućnost filtriranja po specifičnim parametrima koje korisnik izabere. Također, na jednoj formi nudi se mogućnost rada s podacima iz baze gdje se podaci mogu dodati, ažurirati i brisati. Prilikom dodavanja novih podataka u bazu, popis se automatski osvježava te korisnik odmah vidi dodane podatke na postojećem popisu.

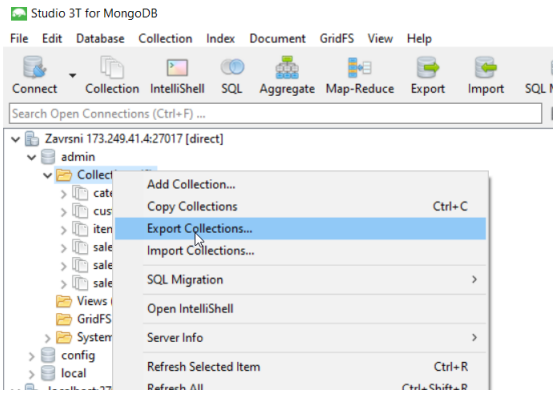
6.1. Kreiranje baze podataka

Prvo što je bilo potrebno napraviti jest kreirati bazu podataka. Sustav dokument-baza podataka koji sam odlučila koristiti jest MongoDB. Imala sam tu sreću da mi je jedna tvrtka dala pristup svojoj test bazi podataka, ali koja je pisana u SQL-u. Moj zadatak je bio te podatke premjestiti u MongoDB. Način na koji sam podatke prebacila u MongoDB jest da sam prvo instalirala program koji se naziva Studio 3T (može se koristiti i Robo 3T jer su programi gotovo identični). Studio 3T je vrsta grafičkog korisničkog sučelja koji podržava relacijske i nerelacijske baze podataka. Tamo sam upisala sve potrebne podatke kako bih se uspješno povezala na testnu bazu podataka kao što se može vidjeti na slici 12.



Slika 12: Uspostava konekcije na SQL bazu

Nakon uspješnog povezivanja odlučila sam da sve te podatke iz te baze podataka želim premjestiti u novu bazu podataka. Razlog tome jest jer sam htjela isprobati i druge MongoDB alate kako bih vidjela njihove mogućnosti, a ne zaustaviti se na samo jednom. Studio 3T nudi mogućnost izvoza kolekcija što mi se pokazala kao odlična stvar jer sam odlučila cijelu bazu podataka premjestiti na MongoDB Cloud Atlas. Način na koji sam izvezla kolekcije jest da sam desnim klikom kliknula na cijelu bazu i izabrala *"Export Collections"* što se može vidjeti na slici 13.

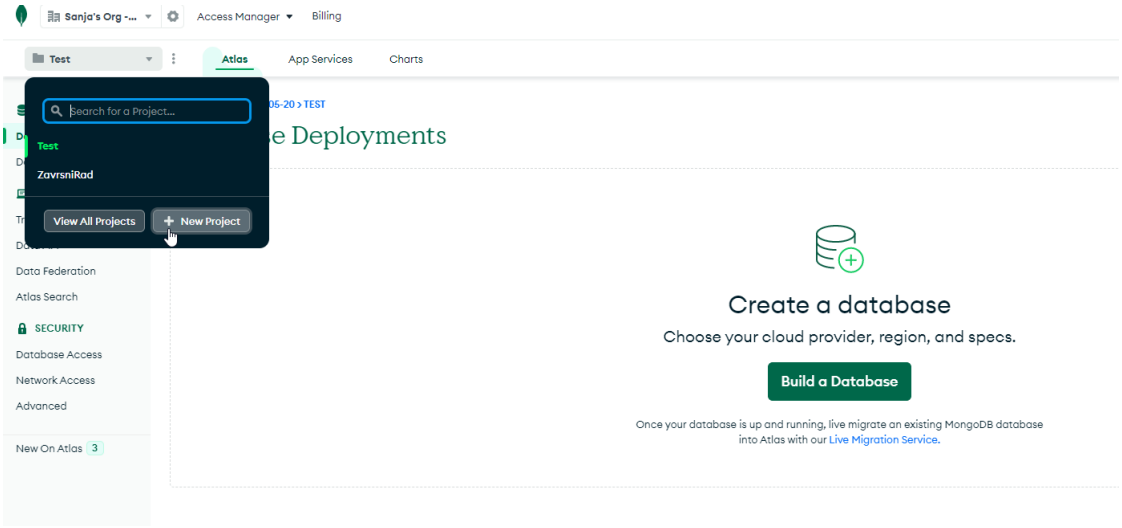


Slika 13: Izvoz kolekcija

MongoDB ima više alata dostupnih za lakši rad s podacima, a potrebno je spomenuti dva alata koja sam koristila, a to su MongoDB Cloud Atlas i MongoDB Compass. MongoDB Cloud Atlas sastoji se od sveobuhvatnog paketa podatkovnih proizvoda koji ubrzavaju i pojed-

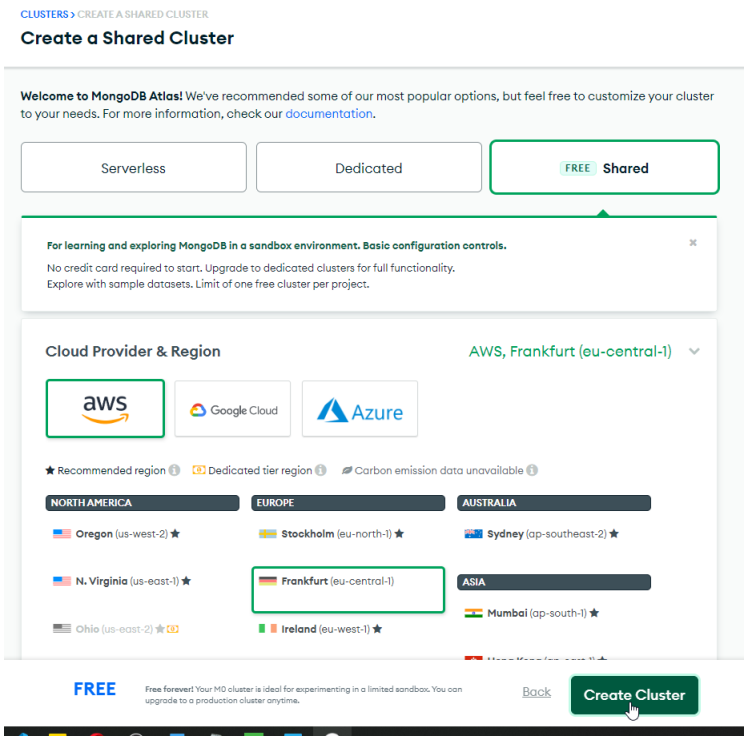
nostavljuju način na koji se radi s podacima u bilo kojoj aplikaciji. MongoDB Atlas jedina je globalno distribuirana baza podataka koja koristi više oblaka. Nudi mogućnost raspoređivanja podataka u više od 70 regija po cijelom svijetu te je moguće kreirati klastere koji koriste više oblaka. Što se tiče MongoDB Compassa je grafičko korisničko sučelje za MongoDB [22].

Potrebno je ulogirati se u MongoDB Cloud te skinuti MongoDB Compass na svoje računalo. Nakon toga sam napravila novi projekt unutar Clouda. Način na koji se to radi možete vidjeti na slici 14, u lijevom kutu kliknete New Project te mu dodijelite naziv. Sada je potrebno kreirati bazu podataka, to se radi na tako da se klikne zeleni gumb u sredini "Build a Database".



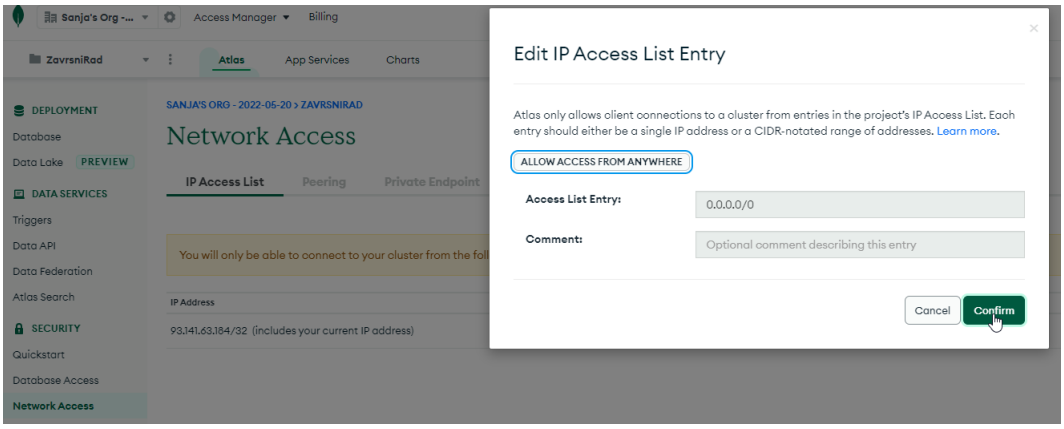
Slika 14: Kreiranje novog projekta unutar MongoDB Clouda

Zatim se samo odabere opcija "Free" te je moguće izabrati još naprednih postavki poput naziva klastera ili veličine baze što sam odlučila ostaviti zadane postavke jer radim s manjom količinom podataka. Postupak možete vidjeti na slici 15. Potrebno je također kreirati korisnika koji ima pristup bazi podataka.



Slika 15: Kreiranje baze podataka i klastera

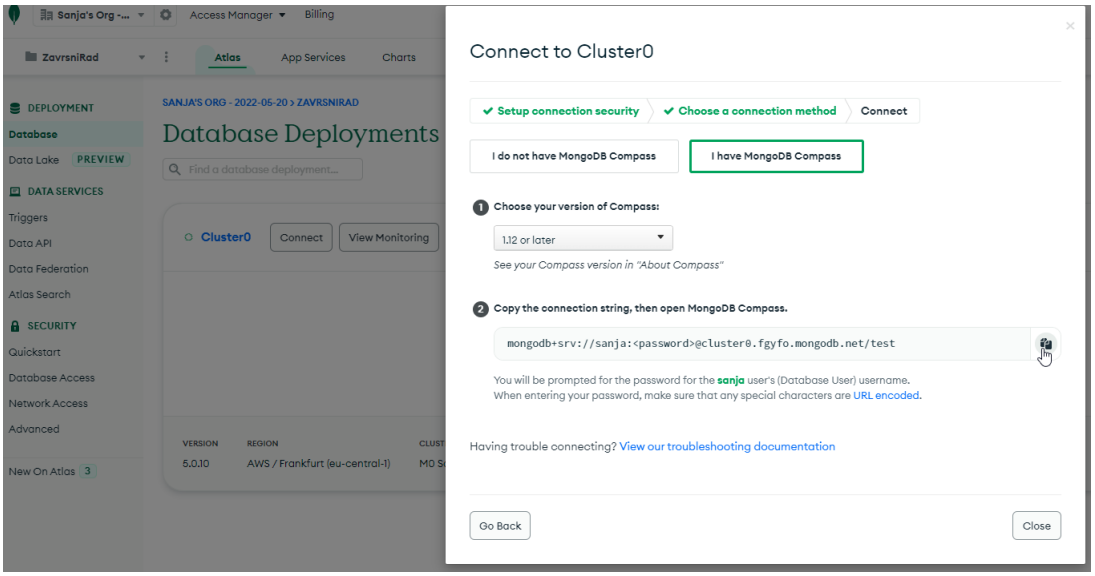
Nakon uspješno kreirane baze podataka, odlučila sam podesiti koje sve IP adrese imaju pristup bazi. Ta mogućnost je odlična kako bi se zaštitili podaci unutar baze podataka te sam za vrijeme izrade praktičnog djela stavila da samo moja IP adresa ima pristup bazi. Nakon što sam završila cijeli projekt sam promijenila pristup tako da sve IP adrese mogu pristupiti bazi podataka što se može vidjeti na slici 16. Moram napomenuti da je i dalje pristup na bazu reguliran, njoj se može samo pristupiti putem jedinstvenog konekcijskog stringa te naziva korisnika i lozinke.



Slika 16: Podešavanje IP adresa za pristup bazi podataka

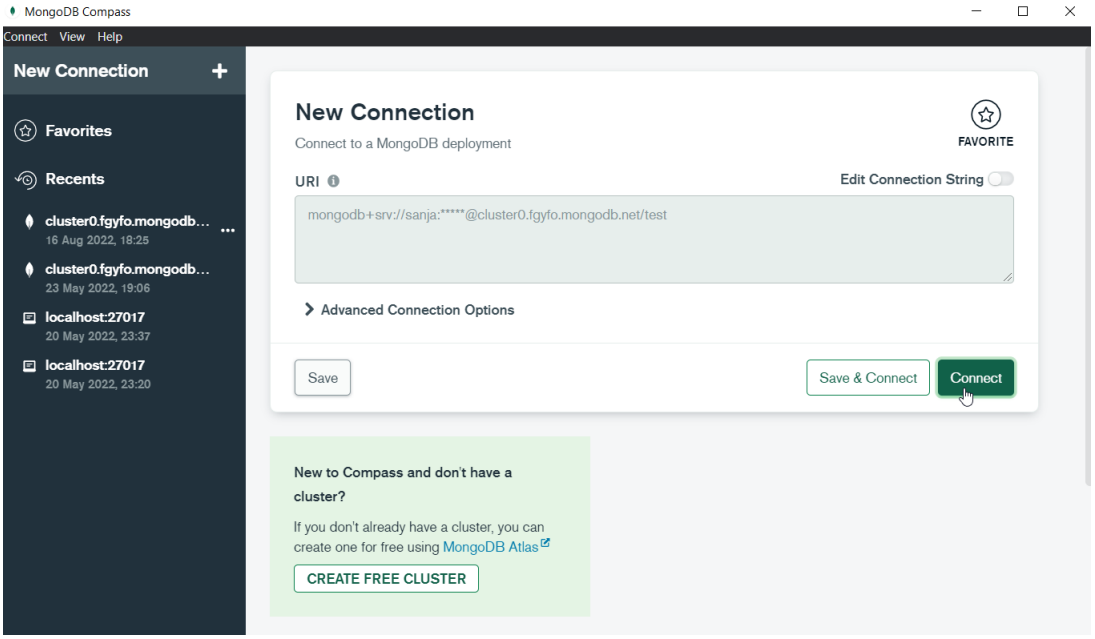
Kako bi izgenerirali string za povezivanje na bazu podataka potrebno se vratiti na početnu stranicu projekta te kliknuti "Connect" pored klastera koji smo napravili te nam MongoDB Atlas sam izgenerirati string za spajanje na bazu podataka koji sam koristila za povezivanje na bazu podataka putem MongoDB Compassa. Oblik stringa možete vidjeti na slici 17. String za

povezivanje na bazu podataka u sebi sadrži naziv korisnika koji ima pristup bazi podataka te lozinku koju je potrebno ručno napisati.



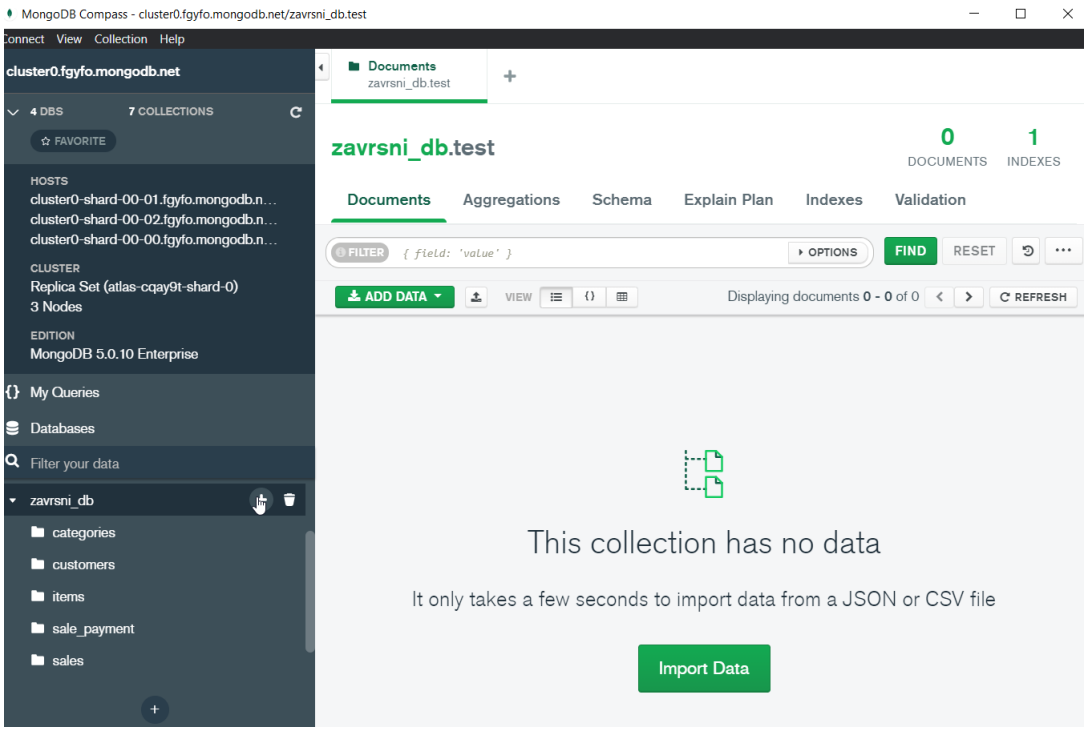
Slika 17: Izgenerirani string za povezivanje na bazu podataka

MongoDB Compass grafičko korisničko sučelje izgleda kao na slici 18. U njega je potrebno zalijepiti string za povezivanje na bazu podataka te kliknuti "Connect". Nakon uspješnog povezivanja može se sada pratiti sve promjene na bazi putem MongoDB Compassa ili putem MongoDB Cloud Atlasa jer baza nije napravljena lokalno.



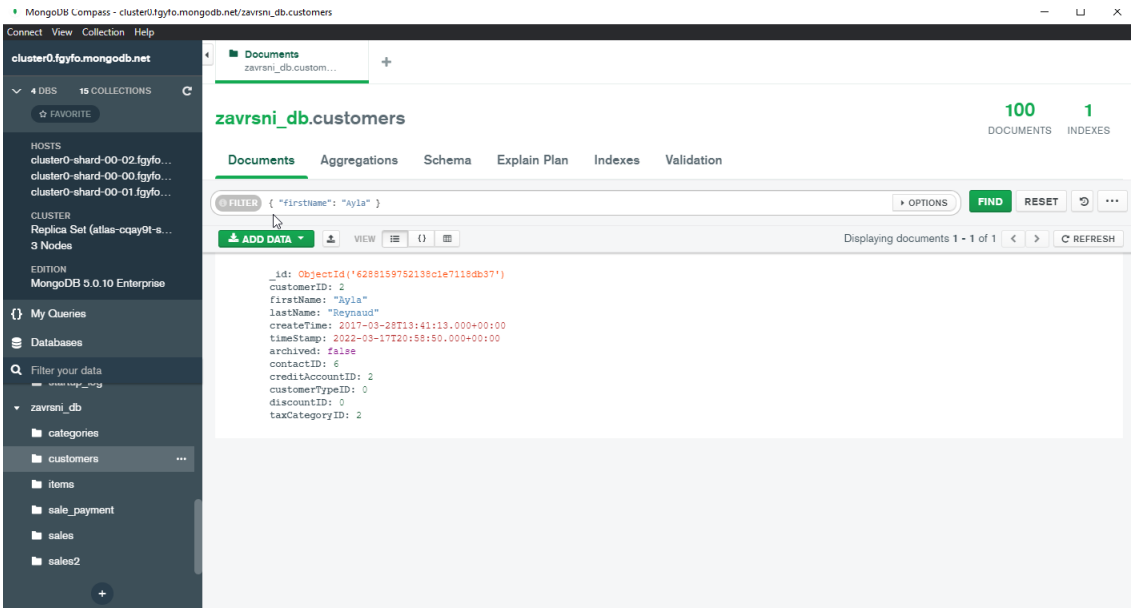
Slika 18: MongoDB Compass grafičko korisničko sučelje

Da bi se kreirale kolekcije unutar baze podataka, potrebno je kliknuti na plus pored bazu podataka koja se nalazi u lijevom kutu. Nazive kolekcija sam koristila identične kao one koje sam dobila u testnoj bazi podataka koju sam izvezla. MongoDB Compass nudi mogućnost uvoza podataka putem klika na gumb "Import Data". Cijeli postupak možete vidjeti na slici 19.



Slika 19: Kreiranje kolekcija

Nakon uspješnog uvoza podataka sam napravila par upita na svakoj kolekciji. Na slici 20 je napravljen upit koji filtrira cijelu kolekciju za kupce po `firstName` za ime Ayla. Već tu možemo vidjeti da je MongoDB automatski svakom dokumentu unutar kolekcije pridružio svoj identifikator.



Slika 20: Uvoz podataka unutar kolekcije

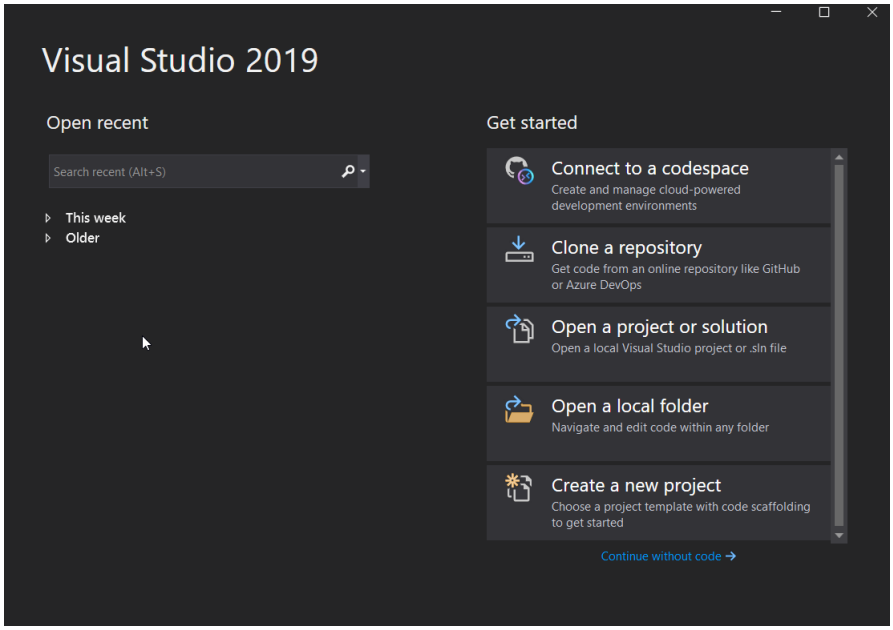
Nakon uspješno kreirane baze podataka i unosa podataka u nju, potrebno je kreirati aplikaciju.

6.2. Kreiranje aplikacije

U ovom poglavlju će biti detaljno objašnjen postupak kreiranja aplikacije i opisane sve funkcionalnosti aplikacije. U nastavku će detaljnije biti objašnjen postupak kreiranja projekta te dizajn aplikacije.

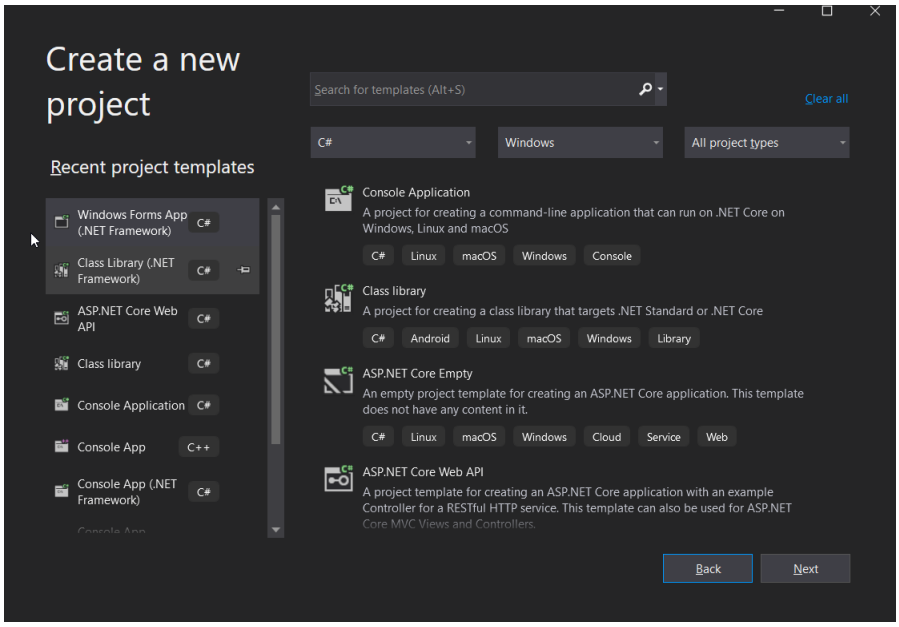
6.2.1. Kreiranje projekta i dizajn aplikacije

Kao što sam već prije spomenula, aplikaciju sam odlučila napraviti u Visual Studiju. Prvo je potrebno otvoriti Visual Studio. Zatim je potrebno kliknuti "Choose a new project" u desnom donjem kutu kao što je prikazano na slici 21.



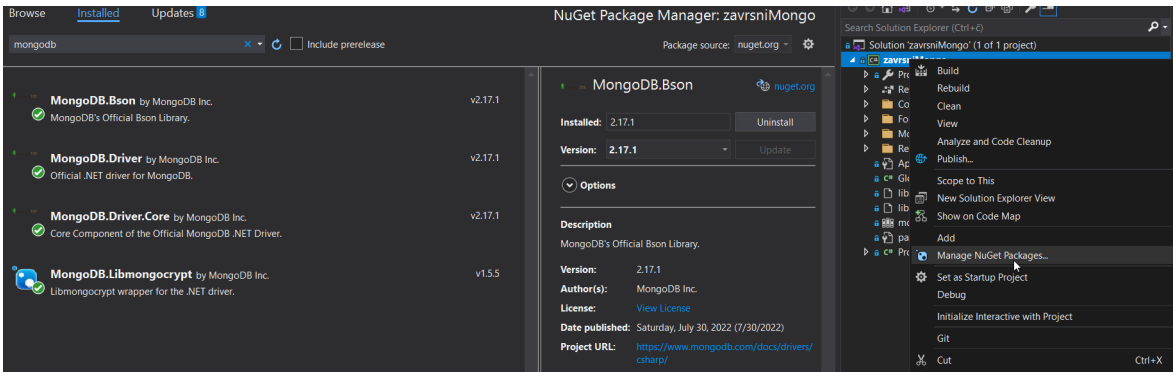
Slika 21: Visual Studio

Nakon toga je potrebno izabrati predložak za projekt. Na slici 22 se mogu vidjeti samo neki od dostupnih predložaka, a ja sam odlučila za ovaj projekt koristiti Windows Forme. Zatim je potrebno kliknuti "Next" te projektu dodati ime i odlučiti gdje će se on spremati lokalno.



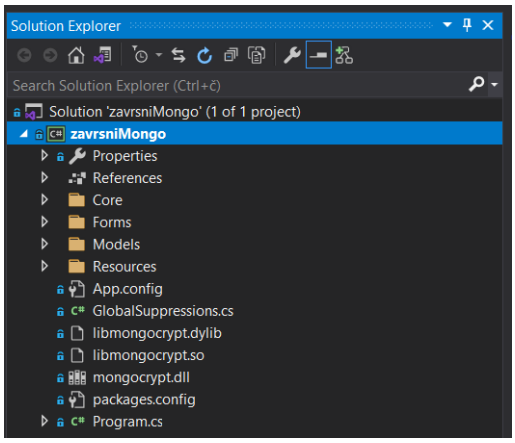
Slika 22: Kreiranje novog projekta u Visual Studio

Nakon što je uspješno kreiran projekt, potrebno je instalirati MongoDB NuGet pakete. To se radi tako da se desnim klikom klikne na projekt "Manage NuGet Packages". Tamo se u tražilicu instaliraju potrebno MongoDB paketi (MongoDB.Bson i MongoDB.Driver). Postupak i instalirani NuGet paketi se mogu vidjeti na slici 23.



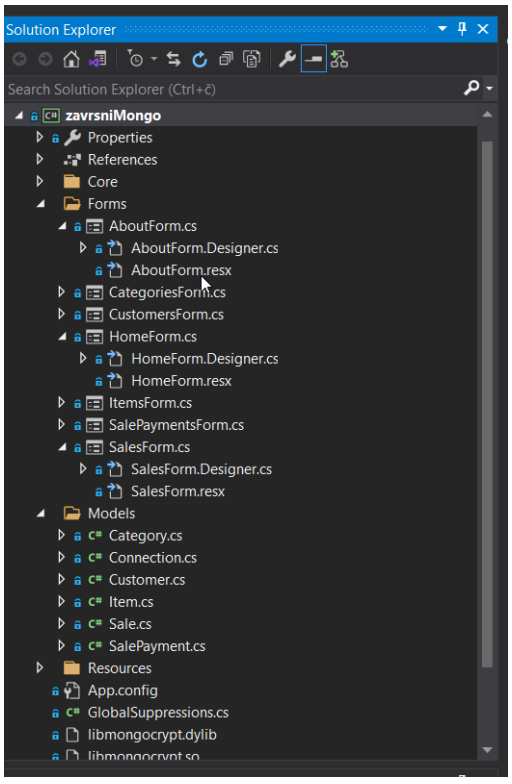
Slika 23: Instalirati NuGet paketi

Radi lakšeg snalaženja i bolje organizacije, kreirala sam dvije mape pod nazivom `Forms` i `Models`. Mapa `Resources` se automatski kreira nakon što se dodaju slike u projekt. Organizaciju projekta možete vidjeti na slici 24.



Slika 24: Organizacija projekta

U mapi `Forms` kreirala sam sve potrebne forme u kojima ću prikazati podatke iz baze podataka. To su forme `CategoriesForm`, `CustomerForm`, `ItemsForm`, `SalesForm` i `SalePaymentForm`. Nazivi forma se podudaraju s nazivima kolekcija u bazi podataka. Dodatne forme su `HomeForm` koje predstavlja formu koja se otvara odmah nakon što se pokrene aplikacija te `AboutForm` koja predstavlja formu u kojoj se nalazi krati opis aplikacije. Što se tiče mape `Models` u njoj sam kreirala sve modele koji se nazivom podudaraju s nazivom kolekcija u bazi te dodatan model pod nazivom `Connection` koji će sadržavati podatke za povezivanje na bazu podataka. Konačni Solution izgleda kao na slici 25.



Slika 25: Konačni Solution projekta

Nakon uspješno napravljenih modela, potrebno je dodati svojstva koja se podudaraju s poljima iz kolekcije u bazi. Pokazat ću kako sam to napravila za model `Category`. Ujedno sam

dodala i konstruktor koji sadrži sva svojstva osim `Id`.

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;
using System;

namespace zavrsniMongo.Models
{
    public class Category
    {
        public Category(int categoryId, string name, int nodeDepth,
           string fullPathName, int leftNode, int rightNode, int
           parentID, DateTime createTime, DateTime timeStamp)
        {
            CategoryId = categoryId;
            Name = name;
            NodeDepth = nodeDepth;
            FullPathName = fullPathName;
            LeftNode = leftNode;
            RightNode = rightNode;
            ParentID = parentID;
            CreateTime = createTime;
            TimeStamp = timeStamp;
        }

        [BsonId]
        public ObjectId Id { get; set; }
        [BsonElement("categoryID")]
        public int CategoryId { get; set; }
        [BsonElement("name")]
        public string Name { get; set; }
        [BsonElement("nodeDepth")]
        public int NodeDepth { get; set; }
        [BsonElement("fullPathName")]
        public string FullPathName { get; set; }
        [BsonElement("leftNode")]
        public int LeftNode { get; set; }
        [BsonElement("rightNode")]
        public int RightNode { get; set; }
        [BsonElement("parentID")]
        public int ParentID { get; set; }
        [BsonElement("createTime")]
        public DateTime CreateTime { get; set; }
        [BsonElement("timeStamp")]
```

```
        public DateTime TimeStamp { get; set; }
    }
}
```

Isječak kôda 3: Implementacija modela `Category`

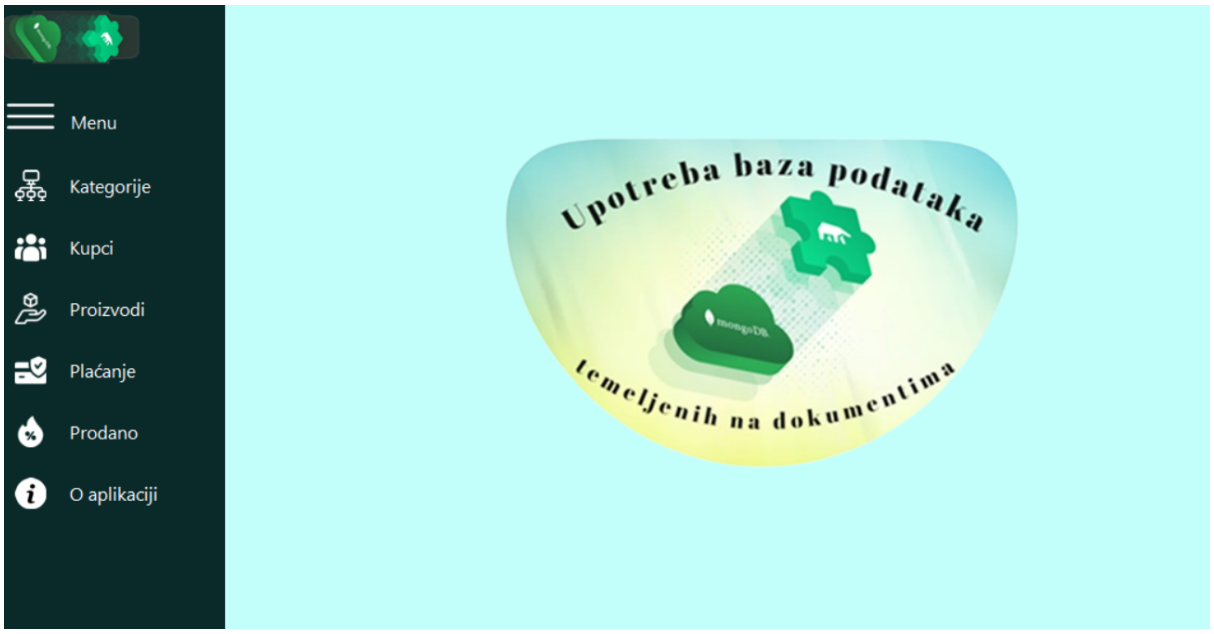
Na isti način su napravljeni i ostali modeli, osim modela `Connection`. U model `Connection` sam dodala string za povezivanje na bazu podataka.

```
using MongoDB.Driver;

namespace zavrsniMongo
{
    public class Connection
    {
        public static string Mongodbconnection =
           "mongodb+srv://sanja:sanja@cluster0." +
            "fgyfo.mongodb.net/?retryWrites=true&w=majority";
        public static MongoClient client = new
           MongoClient(Mongodbconnection);
        public static IMongoDatabase db =
           client.GetDatabase("zavrsni_db");
    }
}
```

Isječak kôda 4: Implementacija modela `Connection`

Što se tiče dizajna aplikacije, aplikacija izgleda kao na slici 26. Odlučila sam se na zelene boje. Što se tiče izbornika, njega sam odlučila dodati na lijevu stranu aplikacije.



Slika 26: Dizajn aplikacije

Način na koji sam napravila izbornik jest tako da sam dodala dva panela koja sam nazvala `menuBar` i `panelNewForm`. `menuBar` mi predstavlja izbornik, a `panelNewForm` je panel koji se nalazi na ostatku forme i koji će prikazivati formu na koju korisnik klikne. Dodala sam još i `menuBarTimer` koji služi da bi se veličina izbornika mogla smanjiti i povećati klikom na ikonicu koja predstavlja meni (ikonica s tri crte u gornjem lijevom kutu). Da bi se mogla podesiti veličina izbornika, bilo je potrebno postaviti minimalnu i maksimalnu širinu `menuBar`. Metoda `openNewForm` je metoda u kojoj sam napravila logiku kako dodati novu formu na `panelNewForm`. Ona se okine svaki puta kada se klikne jedna od ikona na izborniku koja otvara napravljene modele. Programski kod za `HomeForm` izgleda ovako:

```
using System;
using System.Windows.Forms;

namespace zavrsniMongo.Forms
{
    public partial class HomeForm : Form
    {
        bool menuBarExpand = true;
        public HomeForm()
        {
            InitializeComponent();
        }
        private void menuBarTimer_Tick(object sender, EventArgs e)
        {
            if (menuBarExpand)
            {
                menuBar.Width -= 10;
```

```
            if (menuBar.Width == menuBar.MinimumSize.Width)
            {
                menuBarExpand = false;
                menuBarTimer.Stop();
            }
        }
        else
        {
            menuBar.Width += 10;
            if (menuBar.Width == menuBar.MaximumSize.Width)
            {
                menuBarExpand = true;
                menuBarTimer.Stop();
            }
        }
    }

    private Form activeForm = null;
    private void openNewForm(Form newForm)
    {
        if (activeForm != null)
        {
            activeForm.Close();
        }
        activeForm = newForm;
        newForm.TopLevel = false;
        newForm.FormBorderStyle = FormBorderStyle.None;
        newForm.Dock = DockStyle.Fill;
        panelNewForm.Controls.Add(newForm);
        panelNewForm.Tag = newForm;
        newForm.BringToFront();
        newForm.Show();
    }

    private void categoriesButton_Click(object sender, EventArgs
       e)
    {
        openNewForm(new CategoriesForm());
    }

    private void customerButton_Click(object sender, EventArgs e)
    {
        openNewForm(new CustomersForm());
```

```
        }

        private void menuButton_Click(object sender, EventArgs e)
        {
            menuBarTimer.Start();
        }

        private void itemButton_Click(object sender, EventArgs e)
        {
            openNewForm(new ItemsForm());
        }

        private void salePaymentButton_Click(object sender,
           EventArgs e)
        {
            openNewForm(new SalePaymentsForm());
        }

        private void saleButton_Click(object sender, EventArgs e)
        {
            openNewForm(new SalesForm());
        }

        private void aboutButton_Click(object sender, EventArgs e)
        {
            openNewForm(new AboutForm());
        }
    }
}
```

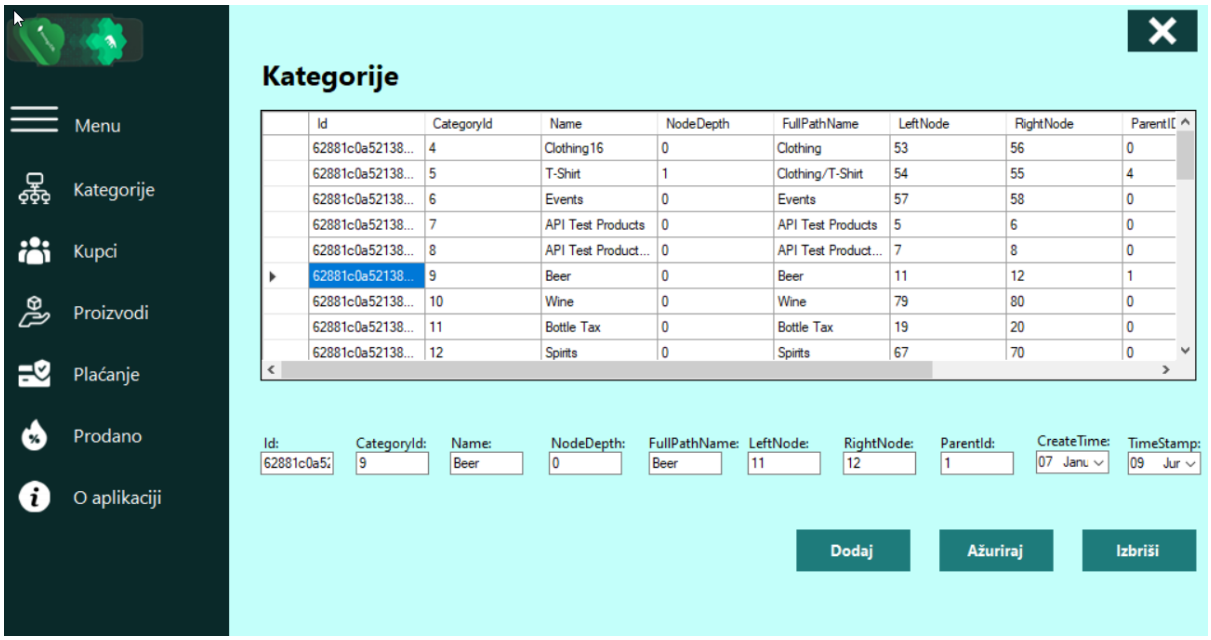
Isječak kôda 5: Implementacija `HomeForm`

U sljedećem poglavlju će detaljnije biti opisana forma koja prikazuje kategorije proizvoda.

6.2.2. Implementacija forme kategorija

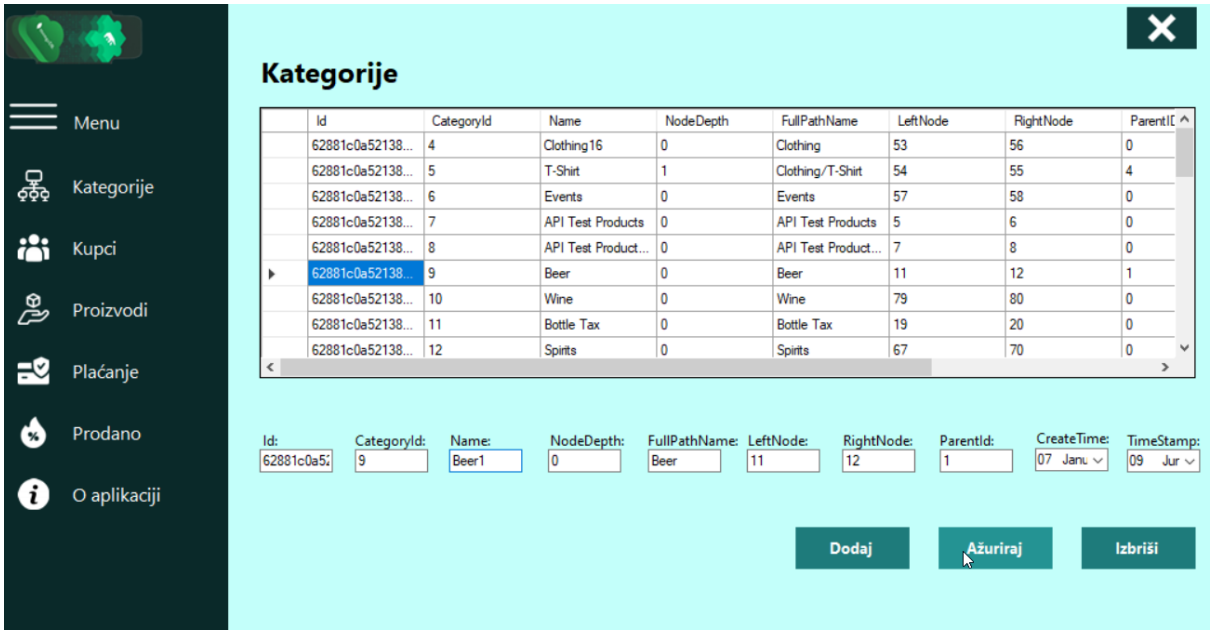
Forma `CategoriesForm` prikazuje sve kategorije proizvoda iz kolekcije te nudi mogućnost dodavanja, ažuriranja ili brisanja podataka iz kolekcije.

Klikom na `Kategorije` u izborniku, prikazuje se lista svih kategorija te nuditi mogućnost dodavanja, ažuriranja i brisanja kategorija. Nakon klika na bilo koji od navedenih gumbova, lista se iznova osvježava. Također, klikom na bilo koje od navedenih redovi u prikazu (što predstavljaju dokumente u kolekciji), tekstualni okviri ispod prikaza se automatski popunjavaju.

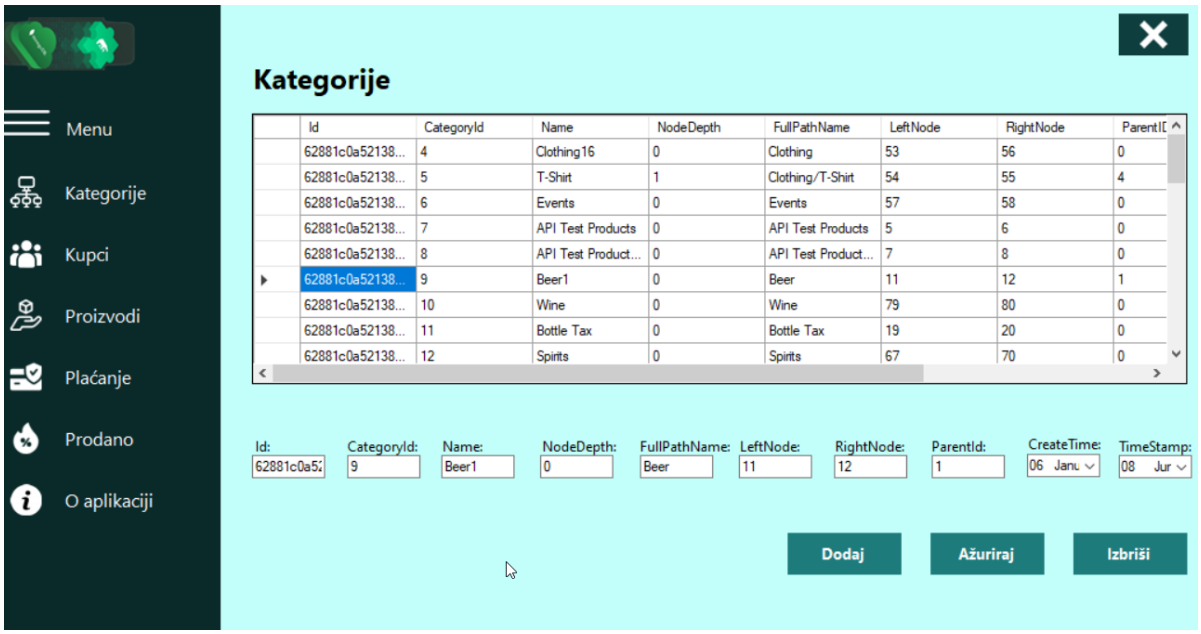


Slika 27: Prikaz forme o kategorijama

Želimo promijeniti naziv iz Beer u Beer1 te nakon što to promijenimo kliknemo na gumb Ažuriraj. Nakon klika na gumb, prikaz se ažurira te sada umjesto Beer piše Beer1. Postupak se može vidjeti na slikama 28 i 29.



Slika 28: Forma o kategorijama prije ažuriranja



Slika 29: Forma o kategorijama nakon ažuriranja

U svakoj formi koja prikazuje modele, dodano je posebno svojstvo `collection` koje koristi model `Connection` kako bi se moglo uspješno povezati na bazu podataka te na određenu kolekciju koja je u tom trenutku potrebna. Svi podaci se prikazuju pozivanjem metode `ReadAllDocuments()` koja se izvršava prilikom inicijalizacije forme. U ovom slučaju je još dodano da se Textboxevi automatski popunjavaju s prvim retkom iz kolekcije.

```
static readonly IMongoCollection collection=
   Connection.db.GetCollection("categories");

        public void ReadAllDocuments()
        {
            List list = collection.AsQueryable().ToList();
            categoriesDataGridView.DataSource = list;

            idTextBox.Text = categoriesDataGridView.Rows[0].Cells[0]
                .Value.ToString();
            categoryIdTextBox.Text = categoriesDataGridView.Rows[0]
                .Cells[1].Value.ToString();
            nameTextBox.Text = categoriesDataGridView.Rows[0]
                .Cells[2].Value.ToString();
            nodeDepthTextBox.Text = categoriesDataGridView.Rows[0]
                .Cells[3].Value.ToString();
            fullPathNameTextBox.Text = categoriesDataGridView.Rows[0]
                .Cells[4].Value.ToString();
            leftNodeTextBox.Text = categoriesDataGridView.Rows[0]
                .Cells[5].Value.ToString();
            rightNodeTextBox.Text = categoriesDataGridView.Rows[0]
```

```
                .Cells[6].Value.ToString();
            parentIdTextBox.Text = categoriesDataGridView.Rows[0]
                .Cells[7].Value.ToString();
            createTimeDateTimePicker.Text = categoriesDataGridView
                .Rows[0].Cells[8].Value.ToString();
            timeStampDateTimePicker.Text = categoriesDataGridView
                .Rows[0].Cells[9].Value.ToString();
        }

        public CategoriesForm()
        {
            InitializeComponent();
            ReadAllDocuments();
        }
```

Isječak kôda 6: Implementacija modela `Connection`

Također ono što je svim formama zajedničko jest gumb u gornjem desnom kutu koji ima znak X. Klikom na taj gumb zatvara se otvorena forma i korisnik se vraća na početnu stranicu aplikacije.

```
private void closeButton_Click(object sender, EventArgs e)
        {
            this.Close();
        }
```

Isječak kôda 7: Implementacija gumba za zatvaranje forme (`closeButton`)

Logika koja stoji iza gumba Ažuriraj jest da je prvo potrebno dohvatiti tekst koji je korisnik napisao u tekstualnim okvirima te nakon toga ažurirati kolekciju u bazi. Da bi se kolekcija ažurirala potrebno je na koristiti ispravne formate, a ako se upiše krivi format okinut će se `FormatException` i prikazat će se tekstualna poruka.

```
private void updateButton_Click(object sender, EventArgs e)
{
 try
 {
    int categoryId = int.Parse(categoryIdTextBox.Text);
    int leftNode = int.Parse(leftNodeTextBox.Text);
    int rightNode =int.Parse(rightNodeTextBox.Text);
    int parentId = int.Parse(parentIdTextBox.Text);

    if(nameTextBox.Text is string && nodeDepthTextBox.Text is string
       &&
       fullPathNameTextBox.Text is string)
    {
```

```
            var updateDef = Builders.Update.Set("categoryID",
                categoryId).Set("name",
                nameTextBox.Text).Set("nodeDepth", nodeDepthTextBox.Text)
                 .Set("fullPathName",
                    fullPathNameTextBox.Text).Set("leftNode",
                    leftNodeTextBox.Text)
                 .Set("rightNode", rightNodeTextBox.Text).Set("parentID",
                    parentIdTextBox.Text)
                 .Set("createTime",
                    DateTime.Parse(createTimeDateTimePicker.Value.Date
                          .ToString()))
                 .Set("timeStamp",
                    DateTime.Parse(timeStampDateTimePicker.Value.Date
                          .ToString()));
            collection.UpdateOne(category => category.Id ==
                ObjectId.Parse(idTextBox.Text) && category.CategoryId ==
                int.Parse(categoryIdTextBox.Text), updateDef);
            ReadAllDocuments();
        }
    }
    catch(FormatException)
    {
        MessageBox.Show("Upisano u neispravnom formatu!");
    }
}
```

Isječak kôda 8: Implementacija gumba za ažuriranje dokumenta u kolekciji (`updateButton`)

Što se tiče gumba za dodavanje nove kategorije on je vrlo sličan. U njemu je dodana bool varijabla koja provjerava postoji li vrijednost `categoryId` unutar kolekcije, ako postoji tada će se ona uvećati sve dok ne dođe do vrijednosti koja se ne nalazi u kolekciji.

```
private void insertButton_Click(object sender, EventArgs e)
{
    try
    {
        int categoryId = int.Parse(categoryIdTextBox.Text);
        int leftNode = int.Parse(leftNodeTextBox.Text);
        int rightNode = int.Parse(rightNodeTextBox.Text);
        int parentId = int.Parse(parentIdTextBox.Text);
        if (nameTextBox.Text is string && nodeDepthTextBox.Text is
            string &&
             fullPathNameTextBox.Text is string)
        {
             bool exists = collection.Find(_ => _.CategoryId ==
```

```
                int.Parse(categoryIdTextBox.Text)).Any();
            do
            {
                categoryId++;
                exists = collection.Find(_ => _.CategoryId ==
                    categoryId).Any();
            } while (exists);


            Category category = new Category(categoryId,
                nameTextBox.Text, int.Parse(nodeDepthTextBox.Text),
                fullPathNameTextBox.Text,
            int.Parse(leftNodeTextBox.Text),
                int.Parse(rightNodeTextBox.Text),
                int.Parse(parentIdTextBox.Text),
            DateTime.Parse(createTimeDateTimePicker.Value.Date
            .ToString()),
                DateTime.Parse(timeStampDateTimePicker.Value.Date
            .ToString()));
            collection.InsertOne(category);
            ReadAllDocuments();
        }

    }
    catch(FormatException)
    {
        MessageBox.Show("Upisano u neispravnom formatu!");
    }
}
```

Isječak kôda 9: Implementacija gumba za unos novog dokumenta u kolekciju (`insertButton`)

Posljednji gumb je gumb za brisanje, potrebno je samo pronaći željeni dokument po svom specifičnom identifikatoru te ga potom izbrisati i ažurirati cijeli prikaz.

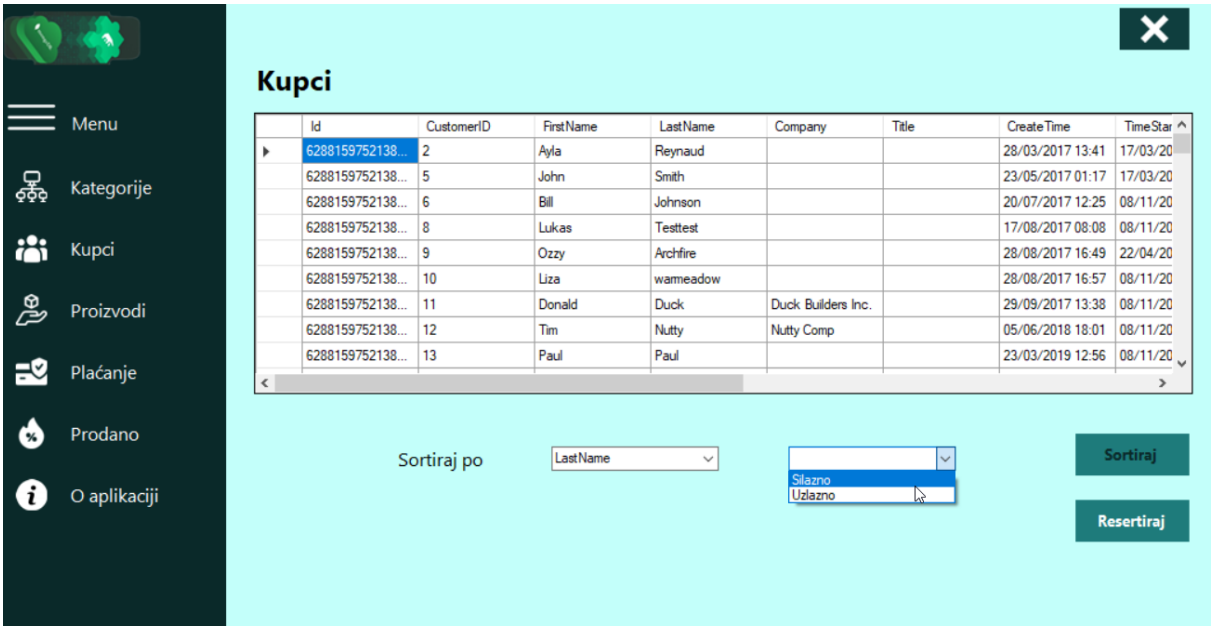
```
private void deleteButton_Click(object sender, EventArgs e)
{
    collection.DeleteOne(category => category.Id ==
        ObjectId.Parse(idTextBox.Text));
    ReadAllDocuments();
}
```

Isječak kôda 10: Implementacija gumba koji briše dokument iz kolekcije (`deleteButton`)

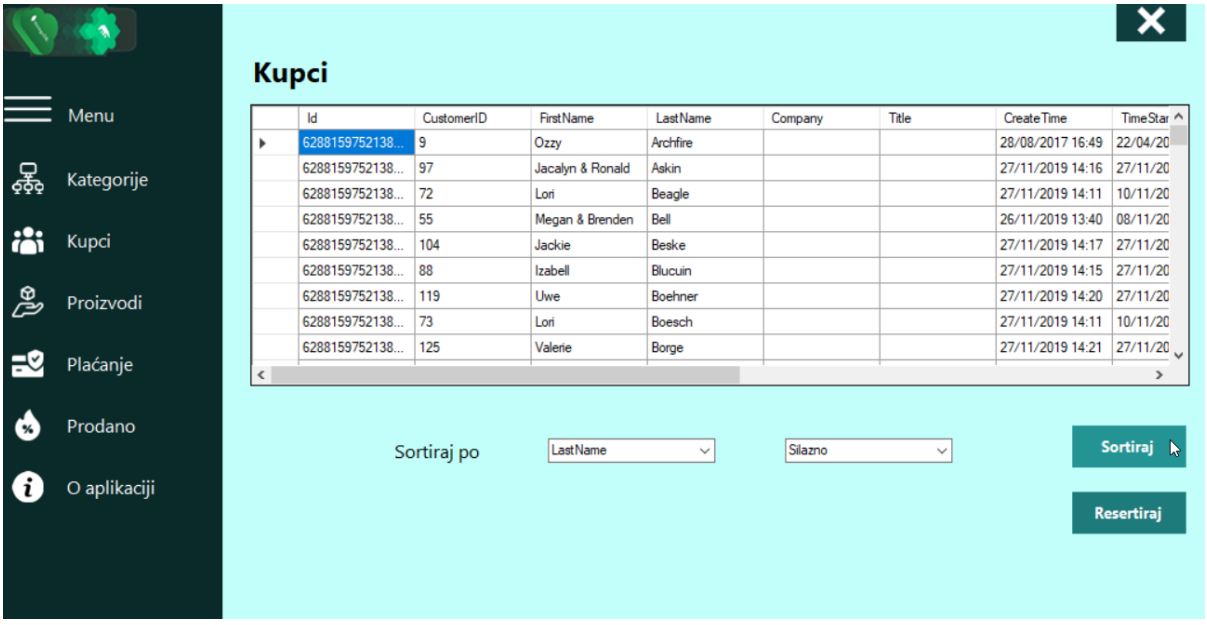
Sljedeći gumb u izborniku jest `Kupci` koji će u nastavku biti detaljnije objašnjen.

6.2.3. Implementacija forme kupaca

Klikom na gumb `Kupci` otvara se forma `CustomersForm` te se prikazuju svi dokumenti iz kolekcije `customers`. Na toj formi nalazi se gumbovi za sortiranje i resetiranje forme. Postavljeno je da se gumb sortiraj ne može kliknuti sve dok se u padajućem izborniku ne odluči po čemu će se sortirati te želi li se sortirati uzlazno ili silazno. Nakon što se to odabere može se kliknuti gumb sortiraj koji sortira cijelu listu. Na slikama 30 i 31 se može vidjeti kako se to točno radi.



Slika 30: Prikaz forme o kupcima prije sortiranja



Slika 31: Prikaz forme o kupcima nakon sortiranja silazno po prezimenu

Logika koja stoji iza toga da se ne može kliknuti gumb `Sortiraj` je u tome da je potrebno provjeriti indekse od oba ComboBoxa te pogledati jesu li selektirani indeksi veći i

jednaki od nule jer ukoliko ComboBox nije označen taka će njegov indeks biti -1.

```
private void sortComboBox_SelectedIndexChanged(object sender,
    EventArgs e)
{
     if(sortComboBox.SelectedIndex >= 0 &&
        sortTypeComboBox.SelectedIndex >= 0)
     {
         sortButton.Enabled = true;
     }
     else
     {
         sortButton.Enabled = false;
     }
}
private void sortTypeComboBox_SelectedIndexChanged(object sender,
    EventArgs e)
{
     if (sortTypeComboBox.SelectedIndex >= 0 &&
        sortComboBox.SelectedIndex >= 0)
     {
         sortButton.Enabled = true;
     }
     else
     {
         sortButton.Enabled = false;
     }
}
```

Isječak kôda 11: Implementacija `sortComboBox` i `sortTypeCombobox`

Što se tiče logike koja je napravljena prilikom klika na gumb `Sortiraj`, potrebno je provjeriti dvije stvari: atribut po kojemu se sortira (`sortType`) i način na koji se želi sortirati (uzlazno ili silazno, `sortValue`).

```
private void sortButton_Click(object sender, EventArgs e)
{
     var sortType = sortTypeComboBox.SelectedItem.ToString();
     var sortValue = sortComboBox.SelectedItem.ToString();
     if (sortValue == "Silazno")
     {

         var filterDefinition = Builders.Filter.Empty;
         var sortDefinition = Builders.Sort.Ascending(sortType);
         var query =
            collection.Find(filterDefinition).Sort(sortDefinition)
```

```
                .ToList();
        customersDataGridView.DataSource = query;
    }
    else
    {
        var filterDefinition1 = Builders.Filter.Empty;
        var filterDefinition = Builders.Sort.Descending(sortType);
        var query =
           collection.Find(filterDefinition1).Sort(filterDefinition)
                .ToList();
        customersDataGridView.DataSource = query;
    }
}
```

Isječak kôda 12: Implementacija gumba za sortiranje (`sortButton`)

Što se tiče gumba `Resetiraj`, njegova logika je vrlo jednostavna, nakon što se klikne gumb, očiste se ComboBoxovi (postave se na indeks -1) te se okine metoda koja dohvaća sve dokumente iz kolekcije.

```
private void resetButton_Click(object sender, EventArgs e)
{
    ReadAllDocuments();
    sortComboBox.SelectedIndex = -1;
    sortTypeComboBox.SelectedIndex = -1;
}
```

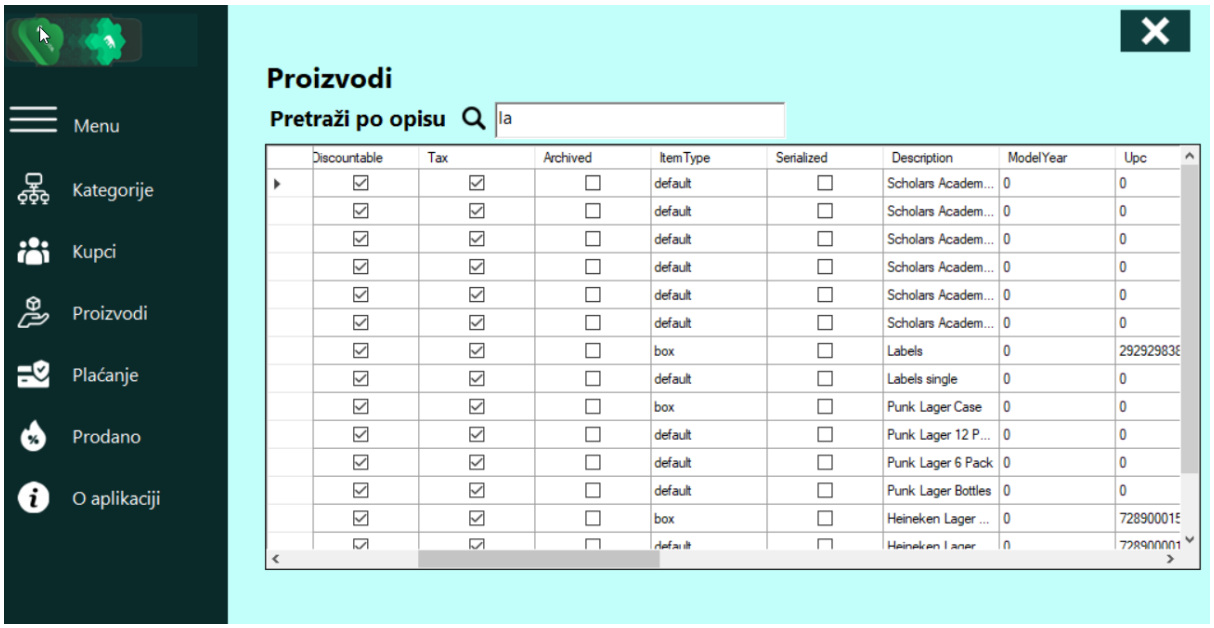
Isječak kôda 13: Implementacija gumba za resetiranje (`resetButton`) prikaza unutar forme o kupcima

Sljedeći gumb u izborniku jest `Proizvodi` koji prikazuje formu `ItemsForm` čija implementacija će biti objašnjena u nastavku.

6.2.4. Implementacija forme proizvoda

Klikom na gumb `Proizvodi` prikazuju se svi proizvodi koji su dohvaćeni iz kolekcije. Logika dohvaćanja podataka iz baze je ista pa ju neću ponovno objašnjavati. Ovdje se nudi mogućnost pretražiti proizvode po opisu (`Description`). Tako u primjeru na slikama 32 i 33 možemo vidjeti da se dohvaćaju svi proizvodi koji u opisu sadrže "la".

Slika 32: Prikaz forme o proizvodima



Slika 33: Prikaz forme o proizvodima nakon pretrage po opisu

Što se tiče logike kako se pretražuje cijela kolekcija, potrebno je bilo koristiti Regex. Postavljeno je da se u kolekciji pretražuje po `Description` oni znakovi koji su upisani u `searchRichTextBox`. Postavljeno je da se pretraživani znakovi mogu podudarati s onima iz baze bez obzira na velika i mala slova (engl. *case insensitive*).

```
private void statusRichTextBox_TextChanged(object sender, EventArgs
    e)
{
    var filter = Builders.Filter.Regex("description", new
        MongoDB.Bson.BsonRegularExpression(searchRichTextBox.Text,
```

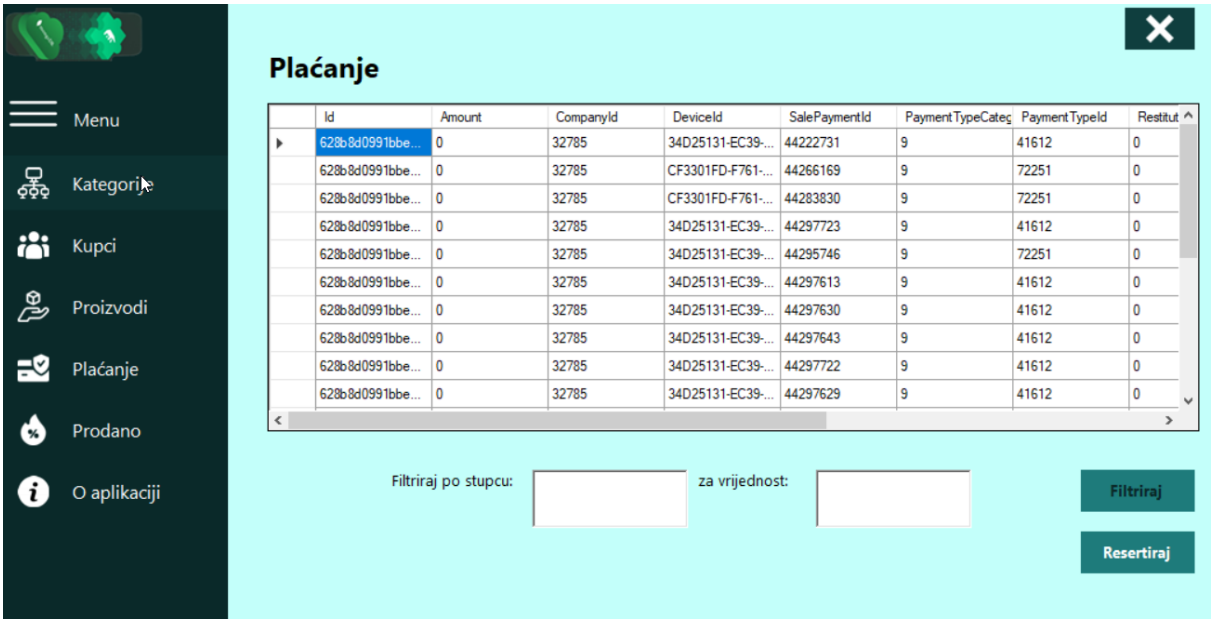
```
            "i"));
        var query = collection.Find(filter).ToList();
        itemsDataGridView.DataSource = query;
}
```

Isječak kôda 14: Implementacija tekstualnog okvira za pretraživanje po opisu

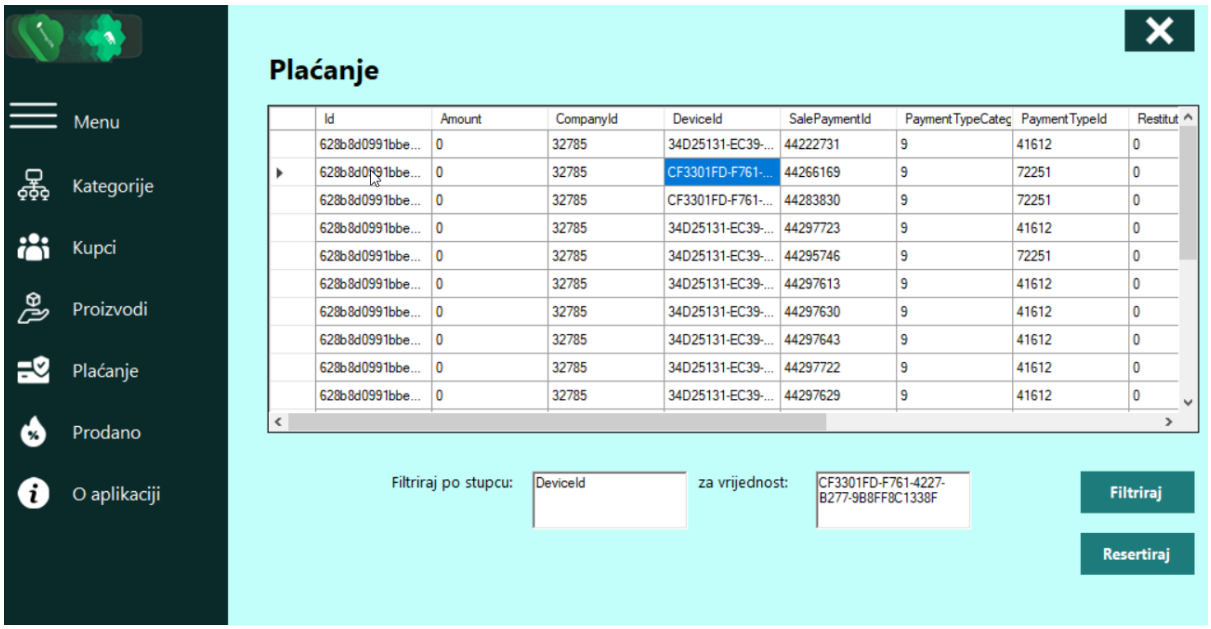
Sljedeća gumb u izborniku je gumb `Placanje` koji prikazuje kolekciju `sale_payment` na formi `SalesPaymentForm`. U nastavu će implementacija `SalesPaymentForm` biti objašnjena.

6.2.5. Implementacija forme plaćanja

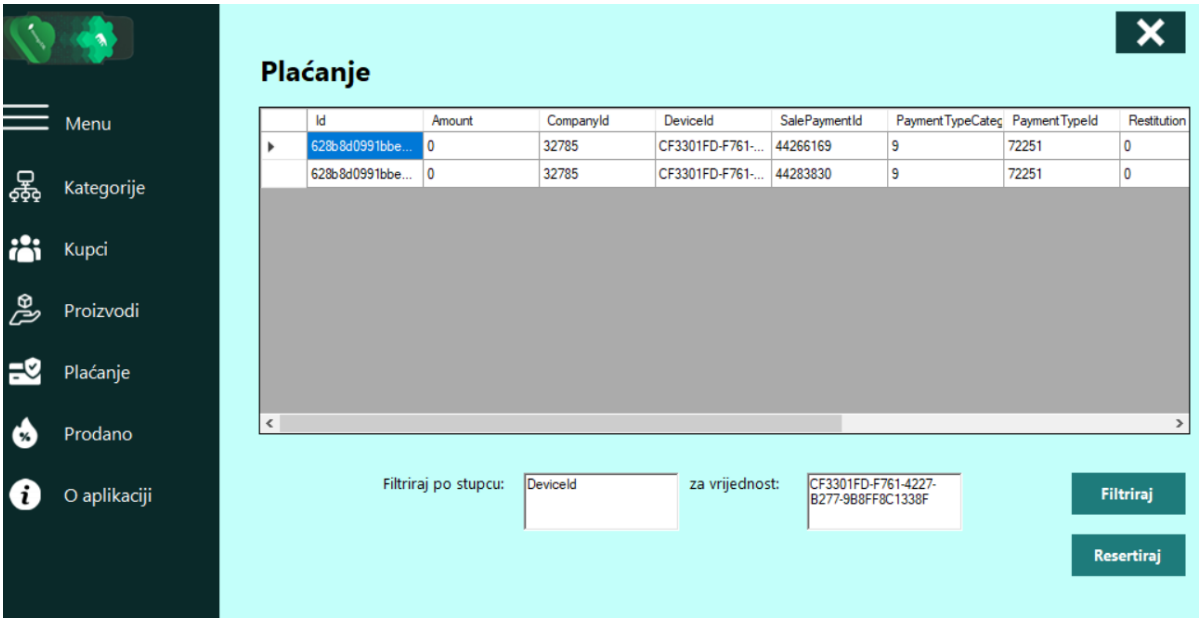
Prikazom forme `SalesPaymentForm` nudi se mogućnost filtriranja po odabranoj ćeliji. Na slici 34 možemo vidjeti kako forma izgleda prije nego što se išta klikne. Postavljeno je da se gumb za filtriranje ne može kliknuti sve dok su tekstualni okviri prazni. Na slici 35 se vidi da je označena ćelija te se automatski popune tekstualni okviri koji navode vrijednost ćeliji te stupac u kojoj se ćelija nalazi. Na slici 36 vidimo kako izgleda kada se filtrira po toj ćeliji.



Slika 34: Prikaz forme o plaćanju



Slika 35: Prikaz forme o plaćanju nakon odabira ćelije



Slika 36: Prikaz forme o plaćanju nakon filtriranja po ćeliji

Programski kôd za selektiranje ćelije izgleda ovako (nemoguće je označiti nazive stupaca):

```
private void salePaymentsDataGridView_CellClick(object sender,
    DataGridViewCellEventArgs e)
{
    try
    {

        queryTextBox.Text = salePaymentsDataGridView.Rows[e.RowIndex]
            .Cells[e.ColumnIndex].Value.ToString();
```

```
            var currentCell = salePaymentsDataGridView.Rows[e.RowIndex]
                .Cells[e.ColumnIndex];
            filterTextBox.Text = currentCell.OwningColumn.Name;
        }
        catch(FormatException)
        {
            MessageBox.Show("Molim oznacite celije koje nisu nazivi
               stupaca");
        }
}
```

Isječak kôda 15: Implementacija za klik ćelije u DataGridView-u

Logika za gumb `Filtriraj` je dosta jednostavna. Provjerava se jesu li tekstualni okviri (`queryTextBox` i `filterTextBox`) prazni, ako jesu tada se postavi da je gumb nedostupan sve dok se tekstualni okviri ne popune.

```
private void filterTextBox_TextChanged(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(filterTextBox.Text) &&
       !String.IsNullOrEmpty(queryTextBox.Text))
    {
        queryButton.Enabled = true;
    }
    else
    {
        queryButton.Enabled = false;
    }
}

private void queryTextBox_TextChanged(object sender, EventArgs e)
{
    if (!String.IsNullOrEmpty(queryTextBox.Text) &&
       !String.IsNullOrEmpty(filterTextBox.Text))
    {
        queryButton.Enabled = true;
    }
    else
    {
        queryButton.Enabled = false;
    }
}
```

Isječak kôda 16: Implementacija tekstualnih okvira za filtriranje

Kada se klikne gumb `Filtriraj`, prvo se provjerava je li unesen tekst u `filterTextBox`

(koji predstavlja filtriranje po nazivu stupca) jednak `Id` ili `Saleid` jer su to jedina polja u kolekcijama tipa integer, ostala su tipa string. Nakon toga se kolekcija filtrira po unesenoj vrijednosti u `queryTextBoxu` (koji predstavlja vrijednost po kojoj filtriramo).

```
private void queryButton_Click(object sender, EventArgs e)
{
    if (filterTextBox.Text == "Type" || filterTextBox.Text ==
       "DeviceId" || filterTextBox.Text == "StatusId")
    {
        var filterDefinition =
           Builders.Filter.Eq(filterTextBox.Text, queryTextBox.Text);
        var query = collection.Find(filterDefinition).ToList();
        salePaymentsDataGridView.DataSource = query;
    }
    else
    {
        if(filterTextBox.Text == "Id" || filterTextBox.Text ==
           "Saleid")
        {
            var filterDefinition =
               Builders.Filter.Eq(filterTextBox.Text,
               ObjectId.Parse(queryTextBox.Text));
            var query = collection.Find(filterDefinition).ToList();
            salePaymentsDataGridView.DataSource = query;
        }
        else
        {
            var filterDefinition =
               Builders.Filter.Eq(filterTextBox.Text,
               int.Parse(queryTextBox.Text));
            var query = collection.Find(filterDefinition).ToList();
            salePaymentsDataGridView.DataSource = query;
        }
    }
}
```

Isječak kôda 17: Implementacija gumba za filtriranje unutar forme za plaćanje

Kada se klikne na gumb `Resetiranje` tada se okine `ReadAllDocuments()` metoda koja prikazuje sve dokumente u kolekcije te se tekstualni okviri postavljaju kao da su prazni.

```
private void resetButton_Click(object sender, EventArgs e)
{
    ReadAllDocuments();
    queryTextBox.Text = String.Empty;
    filterTextBox.Text = String.Empty;
```

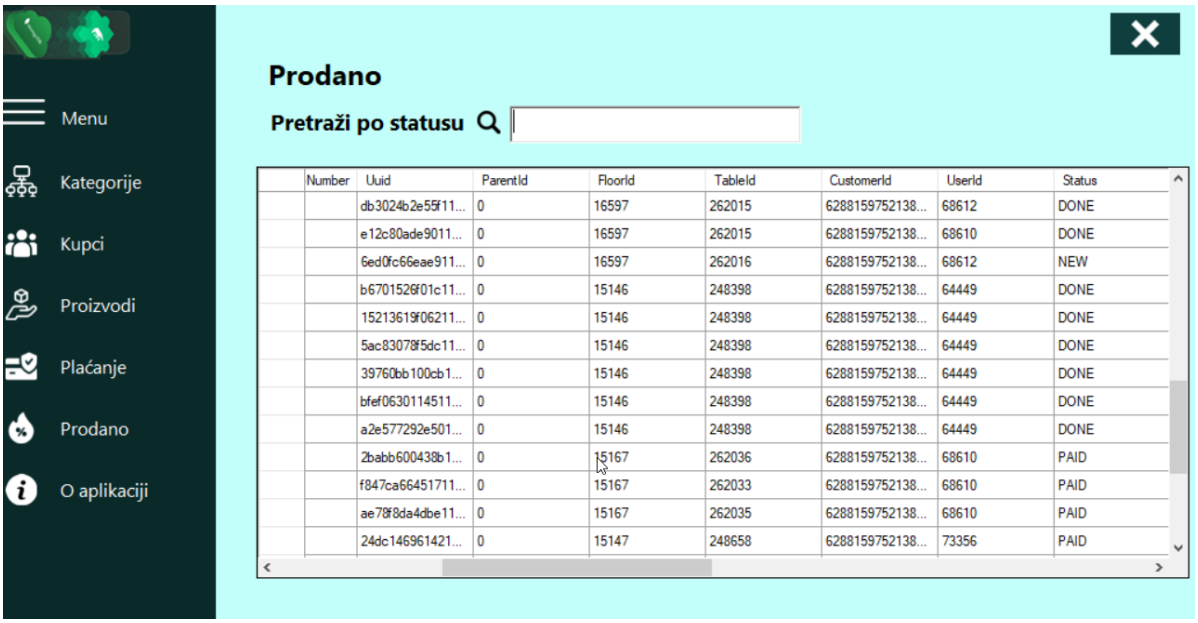
```
}
```

Isječak kôda 18: Implementacija gumba za resetiranje unutar forme za plaćanje

Sljedeći gumb u izborniku je `Prodano` koji otvara formu `SalesForm` čija implementacija je objašnjena u nastavku.

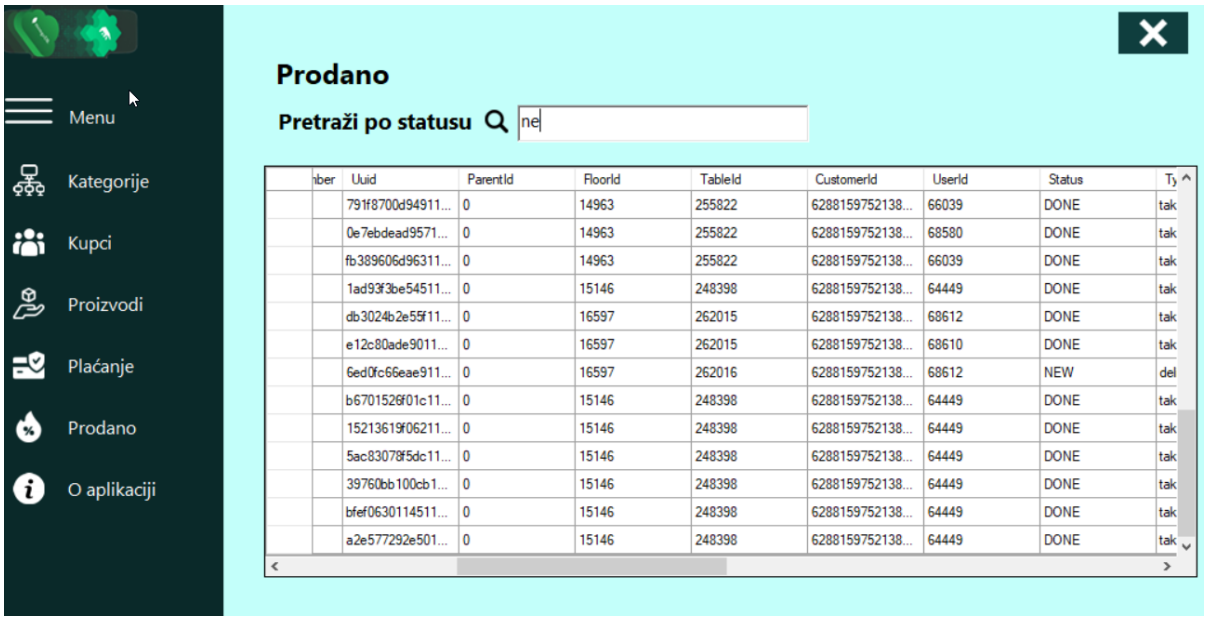
6.2.6. Implementacija forme prodano

Klikom na gumb `Prodano` prikazuje se `SalesForm` koja prikazuje sve dokumente iz kolekcije `sale`. Na ovoj formi moguće je pretražiti sve dokumente po status. Postavljeno je da je moguće pretraživanje bez obzira na velika i mala slova (engl. *case insensitive*). Tako na slici 37 možemo vidjeti kako izgleda prikaz kolekcije prije pretraživanje, a na slici 38 nakon što se kolekcija pretraži po status prema znakovima "ne".



Number	Uuid	ParentId	FloorId	TableId	CustomerId	UserId	Status
	db3024b2e55f11...	0	16597	262015	6288159752138...	68612	DONE
	e12c80ade9011...	0	16597	262015	6288159752138...	68610	DONE
	6ed0fc66eae911...	0	16597	262016	6288159752138...	68612	NEW
	b6701526f01c11...	0	15146	248398	6288159752138...	64449	DONE
	15213619f06211...	0	15146	248398	6288159752138...	64449	DONE
	5ac83078f5dc11...	0	15146	248398	6288159752138...	64449	DONE
	39760bb100cb1...	0	15146	248398	6288159752138...	64449	DONE
	bfef0630114511...	0	15146	248398	6288159752138...	64449	DONE
	a2e577292e501...	0	15146	248398	6288159752138...	64449	DONE
	2babb600438b1...	0	15167	262036	6288159752138...	68610	PAID
	f847ca66451711...	0	15167	262033	6288159752138...	68610	PAID
	ae78f8da4dbe11...	0	15167	262035	6288159752138...	68610	PAID
	24dc146961421...	0	15147	248658	6288159752138...	73356	PAID

Slika 37: Prikaz forme prodano



Slika 38: Prikaz forme prodano nakon pretrage po statusu

Programski kôd na koji način je pretraživanje po statusu napravljeno može se vidjeti ispod:

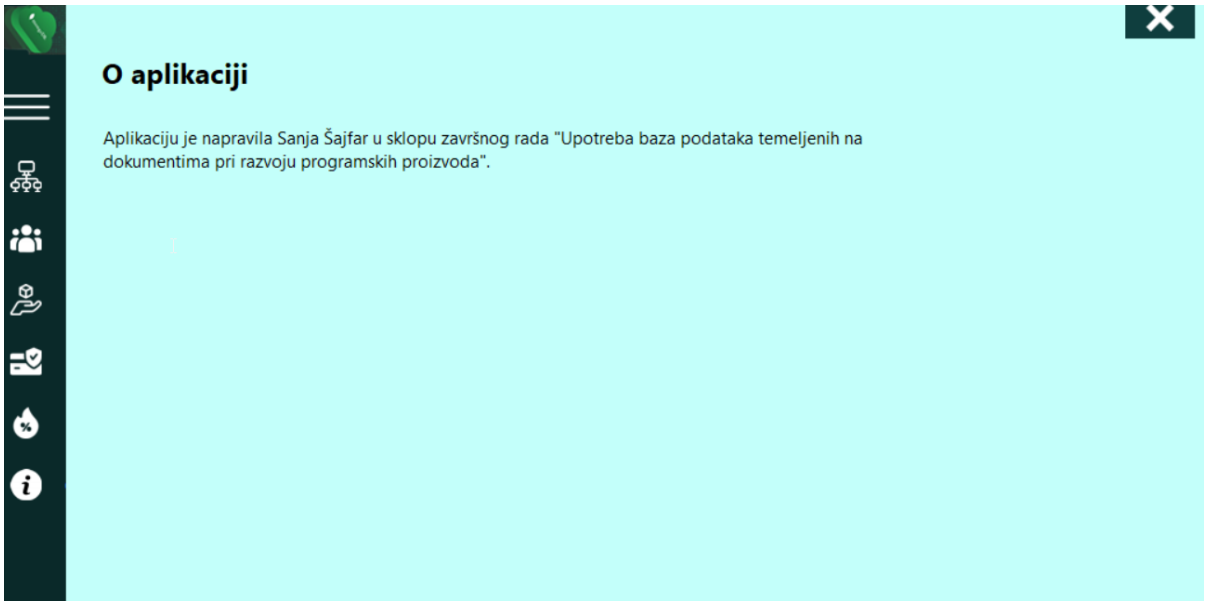
```
private void statusRichTextBox_TextChanged(object sender, EventArgs
    e)
        {
            var filter = Builders.Filter.Regex("status", new
               MongoDB.Bson.BsonRegularExpression(
                statusRichTextBox.Text, "i"));
            var query = collection.Find(filter).ToList();
            salesDataGridView.DataSource = query;
        }
```

Isječak kôda 19: Implementacija gumba za pretraživanje po statusu u kolekciji

Sljedeći gumb u izborniku je `O aplikaciji` koji sadrži osnovne informacije o aplikaciji.

6.2.7. Prikaz forme o aplikaciji

Posljednji gumb u izborniku je `O Aplikaciji`. Klikom na gumb otvara se `AboutForm`. To se može vidjeti na slici 39 te se na toj slici ujedno može vidjeti kako izgleda izbornik kada se on odluči smanjiti.



Slika 39: Prikaz forme o aplikaciji

7. Zaključak

U današnje vrijeme se sve češće izrađuju aplikacije koje rade s velikom količinom kompleksnih ili nestrukturiranih podataka čije spremanje nije moguće u sustave za relacijske baze podataka, tu u pomoć uskaču dokument-baze podataka. Dokument-baze podataka nude mogućnost rada s različitom vrstom podataka gdje pritom svaki dokument može izgledati drugačije i imati različit broj svojstva. Također nude horizontalnu skalabilnost, imaju bolje performanse u odnosu na relacijske baze, nemaju relacijsku shemu i dokumenti se lagano mapiraju u objekte što ubrzava proces stvaranja aplikacija.

Cilj ovog završnog rada jest bio objasniti što su baze podataka i sustavi za upravljanje bazama podataka te je naglasak stavljen na nerelacijske baze podataka. Objašnjene su vrste nerelacijskih baza podataka i napravljena je njihova usporedba s relacijskih baza podataka. Fokus je stavljen na dokument-baze podataka zbog njihove široke primjene, popularnosti i mnogih prednosti koje nude. Za kraj sam izradila bazu podataka u MongoDB i implementirala aplikaciju u C#.

Glavni alati za izradu baze podataka bio je MongoDB Compass koji je ujedno i službeno grafičko korisničko sučelje (engl. *GUI*), a za implementaciju glavni alat je bio Visual Studio koji je i službeno razvojno okruženje (engl. IDE – integrated development environment) za C#. Programskom kôdu za aplikaciju može se pristupiti na sljedećem linku `https://github.com/sanja1999/ZavrsniRad`.

Na kraju se može zaključiti da se isplati naučiti podvrste nerelacijskih baza podataka te naučiti koristiti MongoDB kao bazu podataka orijentiranoj prema dokumentima. Baze podataka orijentirane prema dokumentima pri razvoju programskih proizvoda u programskom jeziku C# predstavljaju izazov, ali i nužno osvježenje i promjenu u odnosu na korištenje relacijskih baza podataka s kojima sam isključivo do sada radila.

Popis literature

[1] F. Cady, „Databases,” siječanj 2017., str. 203–215. DOI: 10.1002/9781119092919.ch14.

[2] E. Derclaye, „What is a Database?” *The Journal of World Intellectual Property*, sv. 5, str. 981–1011, studeni 2005. DOI: 10.1111/j.1747-1796.2002.tb00189.x.

[3] T. Jäkel, „The Role Concept for Relational Database Management Systems,” studeni 2013., ISBN: 978-3-319-14138-1. DOI: 10.1007/978-3-319-14139-8_29.

[4] Y. Silva, I. Almeida i M. Queiroz, „SQL: From Traditional Databases to Big Data,” veljača 2016., str. 413–418. DOI: 10.1145/2839509.2844560.

[5] H. Vyawahare, „Brief Review on SQL and NoSQL,” srpanj 2018.

[6] A. Nayak, A. Poriya i D. Poojary, „Article: Type of nosql databases and its comparison with relational databases,” *International Journal of Applied Information Systems*, sv. 5, str. 16–19, siječanj 2013.

[7] M. Dave, „SQL and NoSQL Databases,” *International Journal of Advanced Research in Computer Science and Software Engineering*, kolovoz 2012.

[8] D. Abadi, P. Boncz i S. Harizopoulos, „Column oriented Database Systems.,” *PVLDB*, sv. 2, str. 1664–1665, kolovoz 2009. DOI: 10.14778/1687553.1687625.

[9] M. Macak, M. Stovcik i B. Buhnova, „The Suitability of Graph Databases for Big Data Analysis: A Benchmark,” siječanj 2020., str. 213–220. DOI: 10.5220/0009350902130220.

[10] M. Besta, E. Peter, R. Gerstenberger, M. Fischer, M. Podstawski, C. Barthels, G. Alonso i T. Hoefler, „Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries,” listopad 2019.

[11] V. Morgante. (2020.). „What is a graph database?” Adresa: https://towardsdatascience.com/what-is-a-graph-database-249cd7fdf24d (pogledano 4. 8. 2021.).

[12] (2020.). „Neo4j Cyber Manual,” adresa: https://neo4j.com/docs/cypher-manual// (pogledano 18. 8. 2022.).

[13] (2018.). „OrientDB Manual,” adresa: http://orientdb.com/docs/2.2.x (pogledano 18. 8. 2022.).

[14] R. Mason, „NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database,” srpanj 2015. DOI: 10.28945/2245.

[15] Y. Hiyane, A. Benmakhlouf i A. Marzouk, „Storing Data in A Document-oriented Database and Implemented from A Structured Nesting Logical Model," *International Journal of Database Management Systems*, sv. 12, str. 17–23, travanj 2020. DOI: 10.5121/ijdms.2020.12202.

[16] L. Lobel. (2015.). „Relational Databases vs. NoSQL Document Databases," adresa: https://lennilobel.wordpress.com/2015/06/01/relational-databases-vs-nosql-document-databases/ (pogledano 14. 8. 2021.).

[17] V. Jain i A. Upadhyay, „MongoDB and NoSQL Databases," *International Journal of Computer Applications*, sv. 167, str. 16–20, lipanj 2017. DOI: 10.5120/ijca2017914385.

[18] D. Vohra, „Using MongoDB," siječanj 2016., str. 57–80, ISBN: 978-1-4842-1829-7. DOI: 10.1007/978-1-4842-1830-3_5.

[19] A. A. Zeeshan. (2016.). „Why To Use C# And When To Prefer Other Languages," adresa: https://www.c-sharpcorner.com/article/why-to-use-C-Sharp-and-when-to-prefer-other-languages/ (pogledano 15. 8. 2021.).

[20] M. Watson. (2020.). „What is C# used for?" Adresa: https://stackify.com/what-is-c-used-for/ (pogledano 15. 8. 2021.).

[21] (2022.). „Web Languages supported by Visual Studio," adresa: https://visualstudio.microsoft.com/vs/features/web/languages/ (pogledano 15. 8. 2021.).

[22] (2022.). „MongoDB Cloud Services," adresa: https://www.mongodb.com/cloud (pogledano 15. 8. 2021.).

Popis slika

Popis tablica

Popis isječaka kôdova