Usporedba NoSQL baza podataka u React aplikaciji

Dujić, Nikolina

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA NOSQL BAZA PODATAKA U REACT APLIKACIJI

Nikolina Dujić

Split, rujan 2022.

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

ZAVRŠNI RAD

USPOREDBA NOSQL BAZA PODATAKA U REACT APLIKACIJI

Nikolina Dujić

Split, rujan 2022.

Temeljna dokumentacijska kartica

Završni rad

Sveučilište u Splitu

Prirodoslovno-matematički fakultet

Odjel za informatiku

Ruđera Boškovića 33, 21000 Split, Hrvatska

USPOREDBA NOSQL BAZA PODATAKA U REACT APLIKACIJI

Nikolina Dujić

SAŽETAK

Rad se bavi usporedbom NoSQL baza podataka u React aplikaciji. Baza podataka je alat za prikupljanje i organiziranje određenih informacija ili podataka. Za razliku od SQL baza podataka, izum NoSQL baza omogućio je rukovanje velikom količinom podataka uz veliku brzinu. Također, NoSQL baze dizajnirane su za distribuirane pohrane podataka. To je ono što čini NoSQL idealnim odabirom za velike podatke, web aplikacije, online trgovinu, društvene mreže itd. U ovom radu naglasak je upravo na ovoj vrsti baza podataka od kojih su implementirane i detaljnije uspoređene MongoDB i Redis.

Ključne riječi: Baze podataka, SQL, NoSQL, MongoDB, Redis, React, Web aplikacija, izdavanje računa.

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 62 stranica, 69 grafičkih prikaza, 1 tablica i 25 literaturnih navoda.

Izvornik je na hrvatskom jeziku.

Mentor: **Dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Ocjenjivači: **Dr.sc. Divna Krpan**, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Monika Mladenović, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Dr.sc. Goran Zaharija, docent Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad prihvaćen: Rujan, 2022.

Basic documentation card

Thesis

University of Split

Faculty of Science

Department of Computer Science

Ruđera Boškovića 33, 21000 Split, Croatia

COMPARISON OF NOSQL DATABASES IN REACT APPLICATION

Nikolina Dujić

ABSTRACT

This paper is concentrated on a comparison of NoSQL databases in a React application. A database is a tool for collecting and organizing certian information or data. Unlike SQL databases, the invention of NoSQL databases made it possible to handle large volumes of data at high speed. Also, NoSQL databases are designed for distributed data stores. That made NoSQL an ideal choice for big data, web applications, online stores, social media, etc. In this thesis, the emphasis is on exactly this type of database, of which Mongo and Redis are implemented and detailedly compared.

Key Words: Databases, SQL, NoSQL, MongoDB, Redis, React, Web application, issuing invoices.

Thesis deposited in library of Faculty of science, University of Split.

Thesis consist of: 62 pages, 69 figures, 1 tables and 25 references

Original language: Croatian

Supervisor: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split

Reviewers: **Divna Krpan**, **Ph.D.** Assistant Professor of Faculty of Science, University of Split,

Monika Mladenović, Ph.D. Assistant Professor of Faculty of Science, University of Split,

Goran Zaharija, Ph.D. Assistant Professor of Faculty of Science, University of Split

Thesis accepted: September, 2022.

ZAHVALA

Prije svega, zahvalila bih se profesorici i metorici Divni Krpan na vodstvu, pomoći i korisnim savjetima tijekom izrade ovog završnog rada. Hvala Vam i za preneseno znanje kroz brojne kolegije koje ste predavali.

Također, zahvalila bih se svojim roditeljima i bratu koji su mi uvijek bili prva i najveća podrška na ovom putovanju od tri godine. Hvala vam za svaki savjet, za svaku kritiku i za sve ono što ste mi omogućili do sada jer bez vas ne bih bila tu gdje jesam.

Uz to, zahvalila bih se i svim svojim prijateljima i prijateljicama koji su bili tu za mene, kao i svim kolegama i kolegicama koji su uvijek bili spremni pomoći kada je to bilo potrebno.

Za kraj, zahvalila bih se i svim profesorima i profesoricama s kojima sam se susrela tijekom studiranja. Hvala Vam za svaku pomoć, sva prenesena znanja i iskustva.

Sadržaj

Uvod

Još od samih početaka ljudi su razmjenjivali i koristili informacije, točnije podatke čija se količina konstantno povećavala te se nastavila povećavati i danas. Podatci su se upotrebljavali i spremali na razne načine kako bi kasnije njihovo korišenje bilo olakšano. U početcima su ih ljudi zapisivali na papir ili koristeći umjetnost, s vremenom se uvelo i zapisivanje u digitalnom obliku i tako dalje. Kako je tehnologija svako malo napredovala, tako smo dobili i bolja rješenja za efikasan rad s podatcima. Danas je ovo sve pojednostavljeno radi izuma baza podataka. Baza podataka predstavlja nekakav alat kojim se prikupljaju i olakšano organiziraju određeni podatci. Baze podataka su ujedno i tema ovog završnog rada.

Prvo obilježavamo pojavu tradicionalnih SQL (engl. *Structured Query Language*) baza podataka. One su se pojavile još 1970-ih godina i pružaju mnogo mogućnosti u radu s podatcima. Dugo se smatralo da su one optimalne za rad s podatcima, no količina podataka koja se koristila u digitalnom obliku se konstantno povećavala i to je zahtijevalo kvalitetnija rješenja u radu s podatcima.

Taj problem je omogućio razvoj NoSQL (engl. *Not only SQL*) baza podataka. Prvi susret s NoSQL-om zabilježen je 1998. godine. Uvođenjem NoSQL-a su se riješili nedostatci rada s SQL-om, na način da su odbacili određena pravila SQL-a kako bi se omogućio kvalitetan rad s podatcima.

U nastavku rada ćemo se detaljnije upoznati s razlikama ove dvije vrste baza podataka. Također, u ovom radu sam se prvenstveno osvrnula na NoSQL baze podataka, njihovu funkcionalnost, strukturu, te razlike između određenih NoSQL baza od kojih sam dvije implementirala u svojoj *React* aplikaciji za izdavanje računa.

1. Baza podataka

Baza podataka je alat za prikupljanje i organiziranje određenih informacija ili podataka. Takvi podatci su uglavnom pohranjeni u digitalnom obliku na osobnom računalu. Ideja koja se provela korištenjem baza jest da aplikacije ne moraju stvarati datoteke na disku. Odnosno, sve aplikacije koriste zajedničku kolekciju podataka. Dodatna prednost koja se razvila izumom baza jest i to da aplikacija ne treba pristupati podatcima sa diska izravno, već se za to brine softver *sustav za upravljanje bazom podataka* ili DBMS (engl. *Database Management System*).

1.1. Povijest baza podataka

Prvi susret s bazama podataka zabilježen je sredinom 60-ih. U to vrijeme se omogućio koncept baze podataka pojavom medija za pohranu s izravnim pristupom kao što su magnetski diskovi. Raniji sustavi oslanjali su se na sekvencijalno pohranjivanje podataka na magnetsku vrpcu. Naknadni razvoj baza podataka se podijelio u tri razdoblja na temelju podatkovnog modela: navigacijsko, relacijsko i postrelacijsko razdoblje.

U navigacijskom razdoblju su se razvila dva glavna modela – hijerarhijski i *CODASYL* model (mrežni model). Kod njih su se koristili pokazivači kako bi se pratili odnosi od jednog do drugog zapisa.

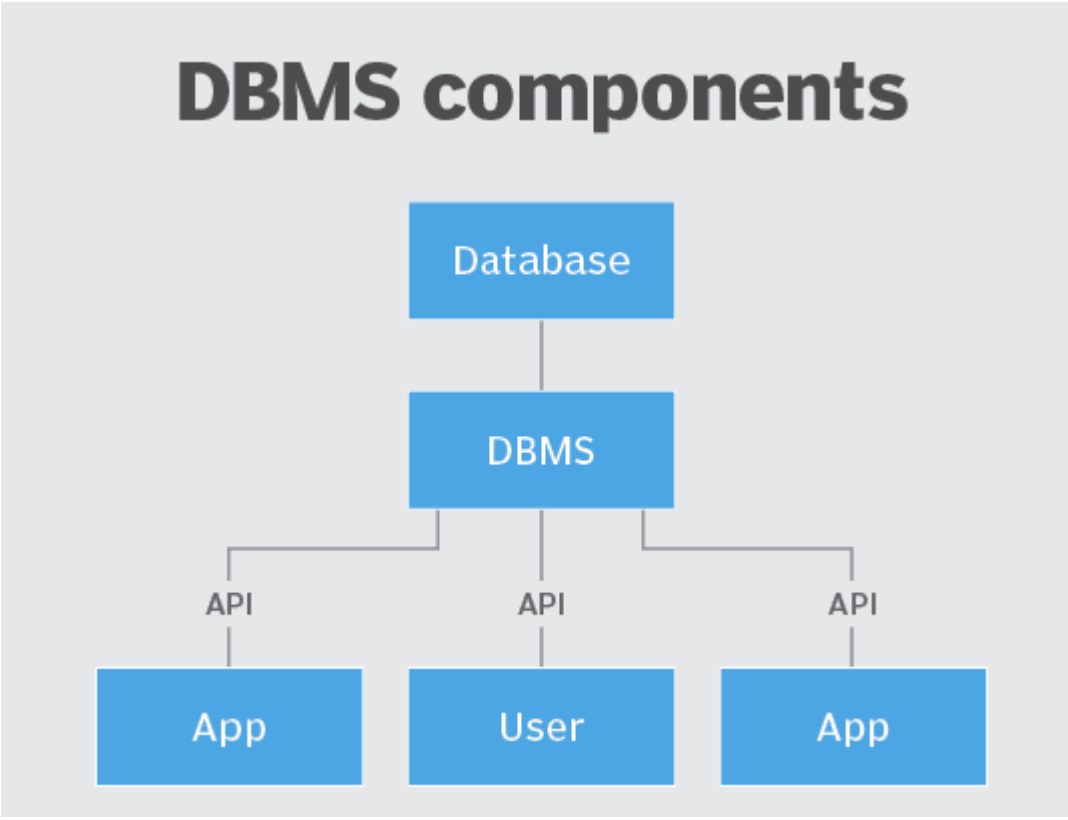
Edgar F. Codd je 1970. godine predložio tzv. relacijski model. Ovaj model je odstupio od ove tradicije i inzistirao je na tome da aplikacije trebaju tražiti podatke prema sadržaju, a ne prema poveznicama. Zatim je 80-ih IBM razvio prvi prototip relacijske baze podataka kojeg je standardizirao ANSI. Do 90-ih su dominirale u svim aplikacijama za obradu podataka, a od 2018. i dalje dominiraju: *Oracle, MySQL* i *IBM db2*.

90-ih se uvode baze orijentirane prema objektu. To je donijelo uspjeh u područjima gdje se upravljalo kompleksnim podatcima npr. prostorne baze podataka, inženjerski podatci, multimedijski podatci.

Sljedeća generacija takozvanih post-relacijskih baza podataka razvila se u 2000-ima i postala je poznata kao NoSQL baza podataka. Ovim bazama se uvela brza pohrana o kojoj ćemo detaljnije pričati u nastavku.

1.2. Sustav za upravljanje bazom podataka (DBMS)

DBMS je softver koji služi za kreiranje i upravljanje bazama podataka. On omogućuje krajnjim korisnicima stvaranje, čitanje, ažuriranje, brisanje i zaštitu podataka u bazi. Ovo je zapravo najčešći tip platforme koji se koristi za upravljanje podatcima. DBMS služi kao sučelje između baze podataka i korisnika ili aplikacijskog programa, na način da osigurava da se podatci dosljedno organiziraju i da su dostupni [1].



Slika 1 DBMS. Preuzeto iz [1]

Neki od zadataka koje DBMS obavlja:

1. Upravljanje pohranom podataka
2. Upravljanje sigurnošću
3. Kontroliranje pristupa za više korisnika
4. Upravljanje integritetom podataka
5. Upravljanje sigurnosnom kopijom i oporavkom
6. Korištenje jezika za pristup bazi podataka – DDL (engl. *Data Definition Language*) i DML (engl. *Data Manipulation Language*) naredbe

1.3. Izrada baze podataka

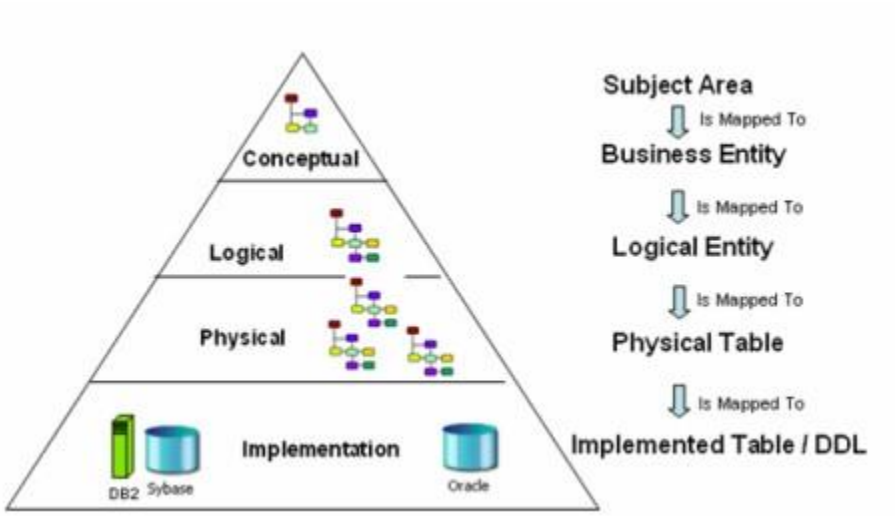
Izrada baze podataka je zapravo proces kreiranja modela za podatke koji se pohranjuju. Model je konceptualni prikaz objekata, veza između tih objekata i pravila. Modeliranje nam pomaže vizualizirati podatke i provoditi poslovna pravila. Razlikujemo dvije tehnike modeliranja podataka [2]:

- UML (engl. *Unified Modeling Language*) je standardni skup dijagrama za modeliranje bilo koje faze razvoja nekog softverskog sustava.
- E-R model (engl. *Entity-relationship model*) je vrsta dijagrama koja prati entitetske odnose. Ovaj model se temelji na tri glavne komponente:
    1. Entitet – bilo koja stvar, objekt, osoba, pojava ili koncept
    2. Atribut – neko svojstvo entiteta
    3. Veze – odnosi između entiteta

1.3.1. Vrste modela

Postoje tri različite vrste modela – konceptualni model, logički model i fizički model. Svaki od njih ima svoju svrhu koju ćemo ukratko definirati u nastavku.

- Konceptualni model – jednostavni model koji se razvija prikupljanjem klijentskih zahtjeva i prijedloga. Svrha kreiranja ovog modela jest da u kasnijem modeliranju imamo što manje promjena. Prikaz entiteta, njihovih atributa i odnosa.
- Logički model – nadogradnja konceptualnog modela. Svrha je povezati entitete, odnosno postaviti odgovarajuće veze između entiteta. Implementacija ovog modela u potpunosti olakšava posao fizičkog modela.
- Fizički model – nadogradnja prethodnog modela na način da se dodaju primarni ključevi, sekundarni ključevi, ograničenja i slično. Ovaj model predstavlja završnu fazu modela i naša baza je spremna za kreiranje [2].



Slika 2 Prikaz procesa implementacije modela baze podataka. Preuzeto iz [3]

1.3.2. Kardinalnost veze

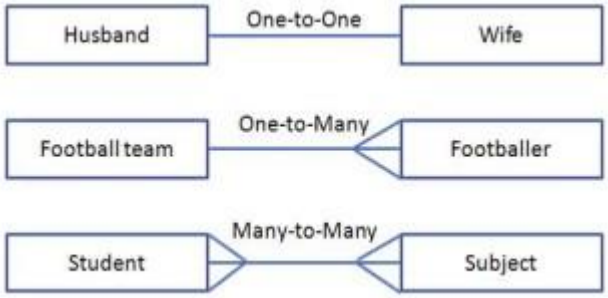
Kardinalnost veze je odnos između podataka u dvije tablice (dva entiteta) baze podataka. Ovaj koncept je ključan za već spomenuti logički model u kojem je implementiran. Veza može biti:

- Jedan (engl. *One*) – oznaka je 1
- Više (engl. *Many*) – oznaka je N

Također, razlikujemo četiri vrste veza:

- Jedan-na-jedan (engl. *One-to-One*) – oznaka 1:1

- Jedan-na-više (engl. *One-to-Many*) – oznaka 1:N
- Više-na-jedan (engl. *Many-to-One*) – oznaka N:1
- Više-na-više (engl. *Many-to-Many*) – oznaka N:N



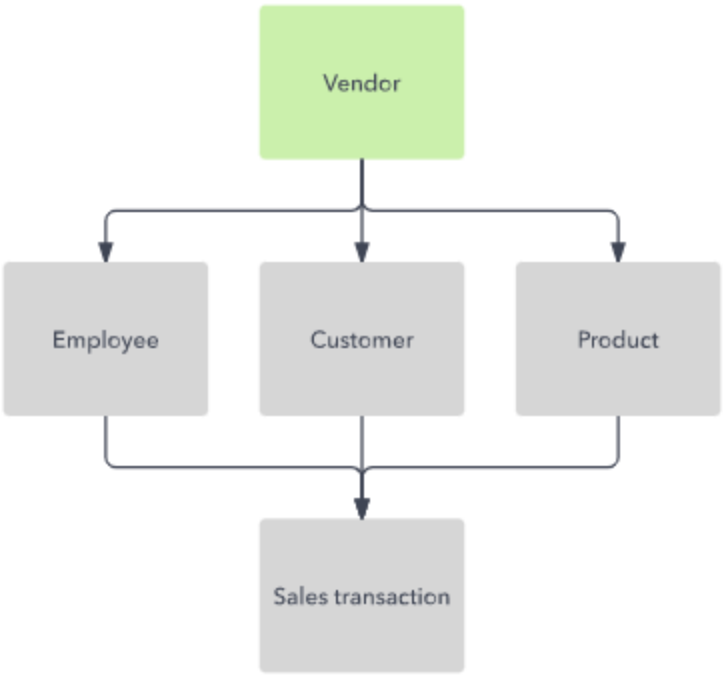
Slika 3 Primjeri kardinalnosti. Preuzeto iz [4]

1.4. Modeli baze podataka

Modeli baze podataka prikazuju logičku strukturu baze, uključujući odnose i ograničenja koja određuju kako se podatci mogu pohraniti i način na koji im se može pristupiti. Imamo različite vrste modela baze podataka. Neki od najčešćih su: mrežni model, hijerarhijski model, relacijski model, objektno-orijentirani model, model dokumenta, ER model i drugi. U nastavku ćemo detaljnije opisati nekoliko njih [5].

1.4.1. Mrežni model

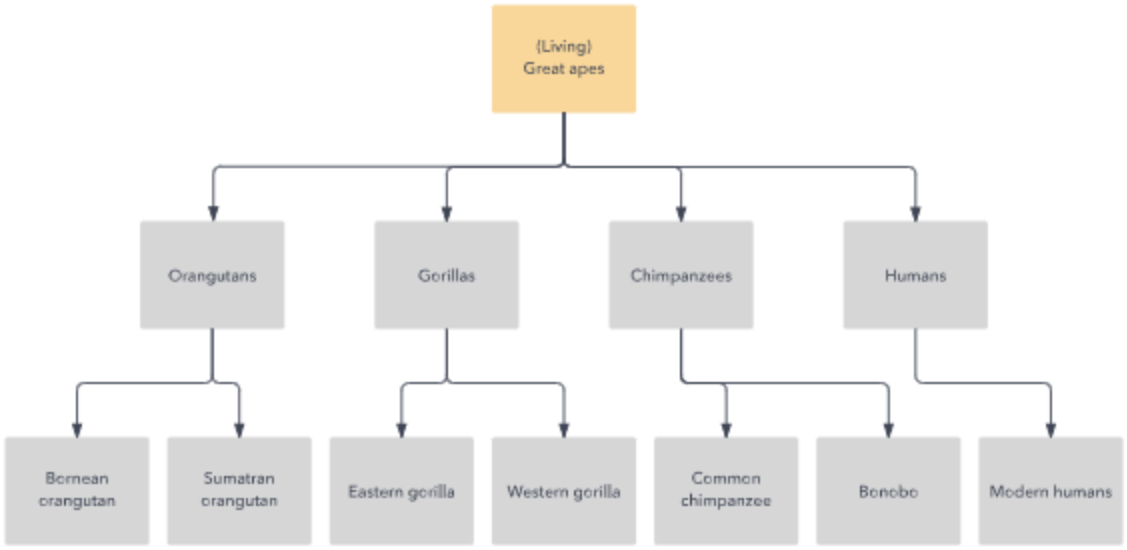
Prvi ćemo istaknuti mrežni model koji je bio najpopularniji 70-ih godina nakon što se razvio na temelju ideja Charlesa Bachman-a. Kod ovog modela baza je mreža koja sadrži čvorove i usmjerene lukove. Čvorovi su tipovi zapisa i oni mogu biti organizirani hijerarhijski. Lukovi su zapravo odnosi tj. veze među zapisima. Za ovaj model je karakteristično da on dopušta odnose više-prema-više između zapisa. Točnije, više roditelja ima više djece, ali i više djece ima više roditelja [5].



Slika 4 Primjer mrežnog modela. Preuzeto iz [5]

1.4.2. Hijerarhijski model

Hijerarhijski model je jedan od slučajeva mrežnog. Podatci su kao što i samo ime govori organizirani hijerarhijski poput stabla. Stablo sadrži skup čvorova i veza „nadređeni-podređeni“ između čvorova. Kao i kod mrežnog modela, čvorovi su tipovi zapisa, a spomenuta veza povezuje te zapise. Za razliku od mrežnog, za ovaj model je karakteristično da jedan roditelj ima više djece, a dijete ima samo jednog roditelja [5].

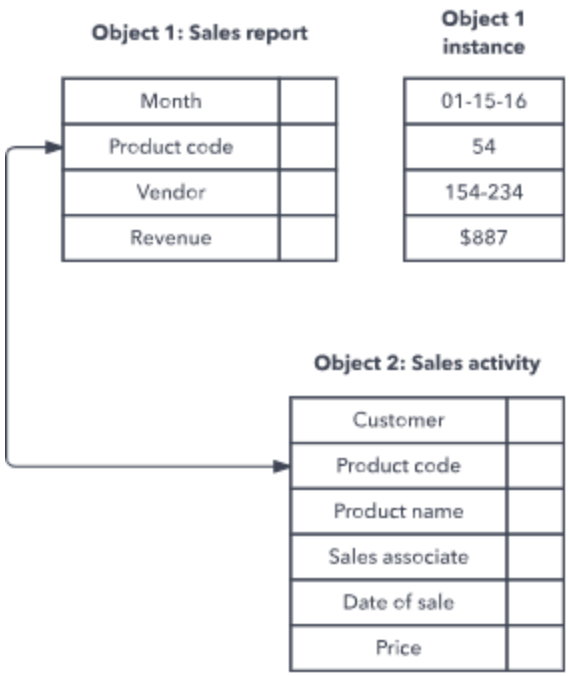


Slika 5 Primjer hijerarhijskog modela. Preuzeto iz [5]

Dakle, možemo zaključiti da je jedina razlika između hijerarhijskog i mrežnog modela u njihovim odnosima „roditelj-dijete“.

1.4.3. Objektno-orijentirani model

Ovaj model definira bazu podataka kao zbirku objekata ili softverskih elemenata koji se mogu ponovno koristiti. Objekti sadrže atribute, odnosno nekakve informacije i metodu koja upravlja podatcima. Objektno-orijentirani model je najpoznatiji model iz post-relacijskog razdoblja baza podataka [5].

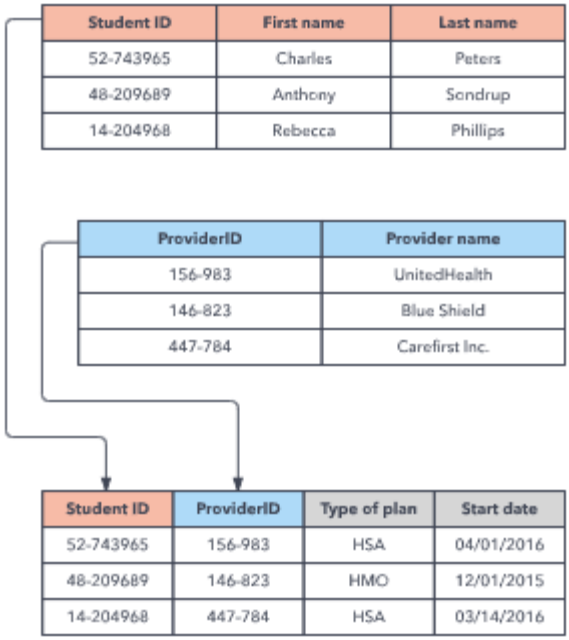


Slika 6 Primjer objektno-orijentiranog modela. Preuzeto iz [5]

Mrežni i hijerarhijski modeli su se primjenjivali još u samim početcima 60-ih i 70-ih godina, dok se danas gotovo ne koriste. Od 80-ih do danas dominira relacijski model. Ono što je bitno naglasiti jest to da se još uvijek nije prešlo na korištenje objektno-orijentiranog modela. Stoga, današnje baze najčešće uspoređujemo s relacijskim modelom kojeg ću opisati u nastavku.

1.4.4. Relacijski model

Relacijski model je najčešće korišten model. On razvrstava podatke u tablice koja se sastoji od stupaca i redaka. Svaki stupac je atribut entiteta (npr. kućna adresa, poštanski broj, spol, godine itd.) Svaki redak je određena instanca entiteta (npr. jedan zaposlenik). Također, ono što je bitno izdvojiti jest da model koristi i odnose (veze) između svih tablica. Tu imamo već prethodno spomenute veze jedan-na-jedan, jedan-na-više i više-na-više [5].



Student ID	First name	Last name
52-743965	Charles	Peters
48-209689	Anthony	Sondrup
14-204968	Rebecca	Phillips

ProviderID	Provider name
156-983	UnitedHealth
146-823	Blue Shield
447-784	Carefirst Inc.

Student ID	ProviderID	Type of plan	Start date
52-743965	156-983	HSA	04/01/2016
48-209689	146-823	HMO	12/01/2015
14-204968	447-784	HSA	03/14/2016

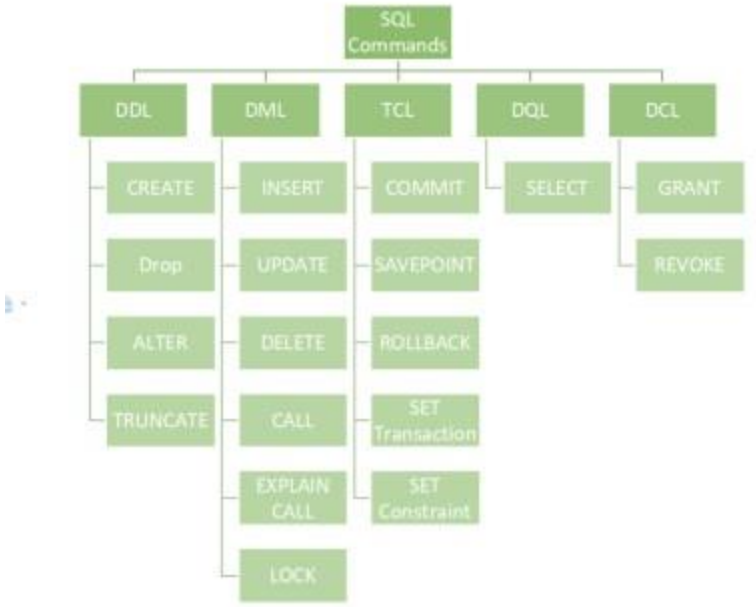
Slika 7 Primjer relacijskog modela. Preuzeto iz [5]

1.5. SQL

SQL (engl. *Structured Query Language*) je računalni jezik baze podataka koji je dizajniran za dohvaćanje i upravljanje podatcima u relacijskoj bazi podataka. Dakle, koristeći SQL možemo pohranjivati, dohvaćati i upravljati podatcima pohranjenim u relacijskoj bazi podataka. Svi *sustavi upravljanja relacijskim bazama* ili RDBMS-ovi (engl. *Relational Database Management System*) poput MySQL, MS Access, Oracle, Postgres i SQL Server koriste SQL kao svoj standardni jezik baze podataka [6].

SQL jezici omogućuju korisnicima čitanje, modificiranje, brisanje, kao i pohranjivanje podataka u sustav baze podataka. Razlikujemo pet različitih vrsta takvih jezika koji se koriste u SQL upitima, a to su:

- DQL ili *jezik za postavljanje upita* (engl. *Data Query Language*) omogućuje korisnicima pretraživanje po bazi podataka koristeći upite. Ključna naredba koja se koristi za postavljanje upita jest *SELECT*.
- DDL ili *jezik za definiranje podataka* (engl. *Data Description Language*) omogućuje korisnicima stvaranje, modificiranje i uništavanje sheme objekata baze podataka. Naredbe koje to omogućuju su *CREATE, ALTER, RENAME* i *DROP*.
- DML ili *jezik za manipulaciju podatcima* (engl. *Data Manipulation Language*) omogućuje korisnicima promjenu postojećih podataka u tablicama. Npr. ovu vrstu jezika možemo koristiti kada želimo pristupiti nekom zapisu, umetnuti novi zapis, ažurirati postojeći zapis ili pak izbrisati postojeće podatke iz tablice. To se provodi koristeći naredbe *SELECT, INSERT, UPDATE* i *DELETE.*
- TCL ili *jezik za kontrolu transakcija* (engl. *Transaction Control Language*) omogućuje korisnicima održavanje SQL operacija unutar baze podataka. Također, spremaju promjene napravljene DML naredbama. Neke od korištenih naredbi su *COMMIT* i *ROLLBACK*.
- DCL ili *jezik za kontrolu podataka* (engl. *Data Control Language*) omogućuje korisnicima upravljanje pravima i dozvolama vezanim za podatke u bazi, a naredbe koje se koriste za to su *GRANT* i *REVOKE* [7].



Slika 8 Prikaz naredbi SQL jezika. Preuzeto iz [7]

SQL ima definirane tipove podataka, koji se dijele na tekstualne, numeričke, tipove podataka za vrijeme i datum, binarne i logičke tipove podataka.

Kao što smo već u uvodu spomenuli SQL baze su dominirale do 90-ih godina. No, uspon interneta je promjenio razvoj aplikacija. Količina i struktura podataka, opseg aplikacija i način na koji su se aplikacije razvijale dramatično su se promijenile. Zbog tih promjena došlo je do otkrića NoSQL tehnologije o kojoj ćemo detaljnije pričati u nastavku.

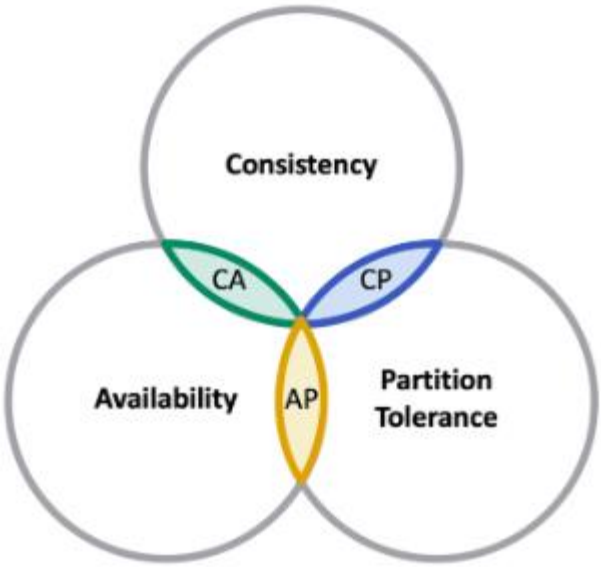
2. NoSQL

Relacijske baze se koriste i danas i njihovo korištenje neće tako skoro nestati, ali često se javlja potreba za korištenjem NoSQL-a. Pojam NoSQL je poznat kao *ne samo SQL* ili *ne relacijska baza*. 1998. godine obilježavamo prvu pojavu ovog pojma kojeg je koristio Carlo Strozzi. NoSQL su netabelarne baze podataka i pohranjuju podatke drukčije od SQL baza podataka. Koristimo ih gotovo u svakoj industriji – bilo to za pohranjivanje nekih ozbiljnijih podataka ili pak zabavnijih i neozbiljnih. Ove baze dolaze u različitim vrstama na temelju njihovog modela podataka. Neke od glavnih vrsta su dokument, ključ-vrijednost, stupac i graf baze podataka. Ove vrste baza ćemo detaljnije obraditi u nastavku [6].

NoSQL baze izvrsno odgovaraju mnogim modernim aplikacijama poput mobilnih, web i igara koje zahtijevaju visokoučinkovite, skalabilne, fleksibilne i vrlo funkcionalne baze podataka za pružanje izvrsnog korisničkog iskustva.

CAP teorem (engl. ***C****onsistency,* ***A****vailability and* ***P****artition tolerance theorem*) znanstvenika Erica Brewera smatra da se u svakom distribuiranom sustavu u isto vrijeme mogu osigurati samo dva od tri jamstva: konzistentnost (dosljednost), dostupnost i toleranciju particije.

- Konzistentnost – svi čvorovi u mreži mogu vidjeti iste podatke u isto vrijeme.
- Dostupnost – svaki zahtjev mora dobiti odgovor o tome je li bio uspješan ili neuspješan. Što je veći broj korisnika u sustavu, to je konzistentnost bolja.
- Tolerancija particija – jamstvo da sustav nastavlja s radom unatoč nekakvog gubitka podataka ili pak pada čitavog sustava [8].



Slika 9 CAP teorem. Preuzeto iz [9]

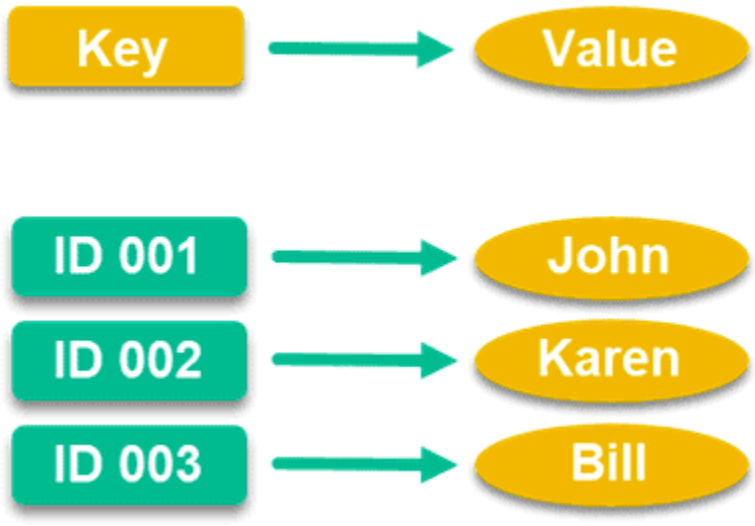
3. Vrste NoSQL baza podataka

NoSQL baze podataka svrstajemo u četiri različite kategorije od kojih razlikujemo ključ-vrijednost, dokument, graf i stupac baze podataka. Neki projekt može imati više vrsta NoSQL baza zbog toga što svaka baza može biti adekvatna za riješavanje nekog slučaja.

U nastavku ćemo detaljnije objasniti četiri spomenute kategorije NoSQL baza podataka.

3.1. Ključ-vrijednost baze podataka

Ključ-vrijednost baze podataka se smatraju najjednostavnijom vrstom nerelacijskih baza. Kod njih su podatci pohranjeni kao asocijativni niz (hash ili riječnik) u kojem su podatci pohranjeni u obliku ključ-vrijednost. Svaki pojedinačni ključ se povezuje samo sa jednom vrijednošću u kolekciji. Pretraživanje se kod ovih baza podataka izvršava po ključu, a ne po vrijednosti. No, tip vrijednosti može biti string, lista, set i slično [10]. Za razliku od tradicionalnih relacijskih, ove baze podataka ne zahtijevaju unaprijed definiranu strukturu. Također, nude veću fleksibilnost pri pohranjivanju podataka i imaju bržu izvedbu [11]. Primjeri ovakvih vrsta baza su: Redis, Amazon DynamoDB, Riak, Couchbase itd.



Slika 10 Ključ-vrijednost nerelacijska baza podataka. Preuzeto iz [11]

3.1.1. Redis

Redis ili *Remote Dictionary Server* je jedan od najpopularnijih primjera ključ-vrijednost baza podataka i ona svoje podatke pohranjuje u glavnu memoriju. Korištenjem memorije on potiže izvrsnu brzinu i skalabilnost, ali je problem što kapacitet RAM-a može biti ograničen. Redis pruža strukture podataka poput nizova, *hashova*, popisa, *bitmape* i slično. Korištenjem Redisa možemo izvoditi razne operacije poput dodavanja nizu, dodavanja elemenata u listu, povećavanja vrijednosti *hash-u* i tako dalje [12].

Redis Cloud je upravljana usluga smještena u oblaku (engl. *Cloud*) koja omogućuje jednostavno i brzo pokretanje podataka naših aplikacija. Odnosno, Redis podatci se pokreću na visoko dostupan i skalabilan način s izvrsnim performansama. Korištenjem Redis Cloud dodatka Heroku, možemo jednostavno koristiti ovu opciju [13].

Ovo je jedna od baza podataka koju sam koristila za implementaciju svog projekta. U nastavku ćemo vidjeti način na koji je implementirana, te ju usporediti s MongoDB dokument bazom koju sam također implementirala u svom projektu.

3.1.2. Amazon DynamoDB

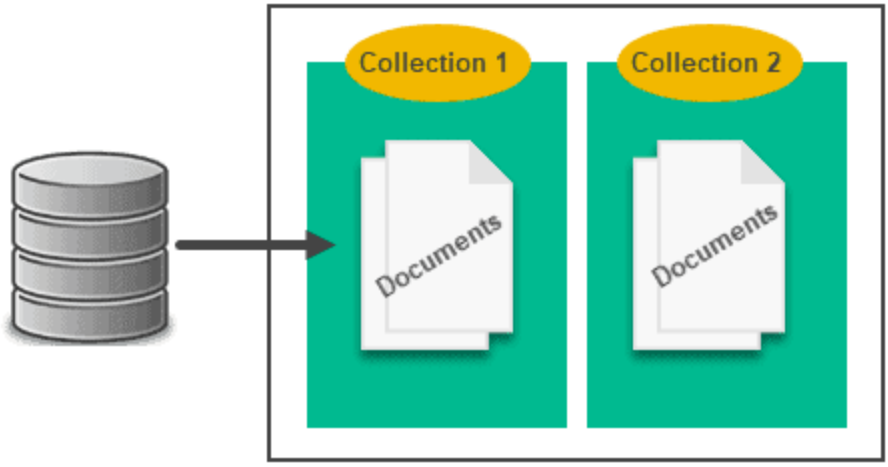
Amazon DynamoDB je jedna od NoSQL baza podataka koju nudi *Amazon.com* koji je dio AWS-a ili *Amazon Web Services* [14]. *Amazon Web Services* predstavljaju web servise koji korisnicima omogućuju brzu pohranu preko interneta [15]. Osim DynamoDB, AWS nudi različite baze podataka poput AmazonRDS, Amazon Aurora i druge. DynamoDB je uistinu brza usluga i pruža pohranu podataka za velike web stranice. Podatkovni model ove vrste baza podataka se sastoji od parova ključ-vrijednost koji su pohranjeni u velikoj nerelacijskoj tablici. No, nije podržan od strane već prije spomenutog RDBMS-a za povezivanje tablica preko stranih ključeva. Ono što je korisno kod ovih baza jest to što se podatkovni model može sastojati i od pohrane pomoću dokumenta koristeći JSON-a (engl. *JavaScript Object Notation*) [16].

Redis	Amazon DynamoDB
Brza, pouzdana i open-source usluga	Pozdana, skalabilna i robusna usluga
Ključ-vrijednost baza podataka, Nudi modele sekundarnih baza poput spremišta dokumenata, DBMS s grafikonima **[17]**	Ključ-vrijednost i dokument baza podataka
„*In-Memory* baza podataka“	„NoSQL baza podataka kao *service*“ [18]
BSD licencirana	Nudi ju *Amazon.com* koji je dio AWS-a

Tablica 1 Usporedba Redis i Amazon DynamoDB baza podataka

3.2. Dokument baze podataka

Dokument baze podataka se sastoje od skupova ključ-vrijednost koji su pohranjeni u dokumentu. Svaki dokument je osnovna jedinica podataka i njih također možemo grupirati u kolekciju (bazu) na temelju njihovih funkcionalnosti. Objektni model možemo prenijeti izravno u dokument pomoću nekoliko različitih formata. Najčešće korišteni su JSON (engl. *Javascript Object Notation*), XML (engl. *EXtensible Markup Language*) i BSON (engl. *Binary JSON*) [11].



Slika 11 Dokument nerelacijska baza podataka. Preuzeto iz [11]

Primjer jednostavnog dokumenta u JSON formatu koji sadrži tri para ključ-vrijednost bi izgledao ovako:

```
{
„ID“ : „1“,
„Ime“ : „Pero“
„Godine“ : „23“
}
```

Kod 1 Primjer jednostavnog dokumenta u JSON formatu

Baze podataka orijentirane na dokumente poslijednjih godina su doživjele ogroman uspon. Zahvaljujući svojoj fleksibilnoj shemi, upotrebljavaju se na platformama za e-trgovinu, analitiku, kao i sustavima za upravljanje sadržajem. Jedne od poznatijih dokument baza su MongoDB, CouchDB, Firebase Realtime itd [19].

3.2.1. MongoDB

MongoDB je jedna od najčešće korištenih dokument baza podataka. Kod ove baze svi podatci su spremljeni u obliku dokumenata, a unutar dokumenata imamo ključ-vrijednost polja. Vrijednost

polja može biti bilo kojeg BSON (engl. *Binary JavaScript Object Notation*) tipa podataka kao što su double, string, bool itd. Dakle, MongoDB pohranjuje, obrađuje i izvršava podatke u binarnom obliku i koristi jednostavan prikaz dokumenata u JSON formatu za ulaz i izlaz podataka. Skup dokumenata čini kolekciju. Dokumenti predstavljaju „cijeli“ podatak, stoga je čitanje olakšano. Vrijednosti polja mogu uključivati druge dokumente, nizove ili nizove dokumenata [20].

```
{
_id: ObjectId(„8776d30492k1p2177j221“),
ime : „Pero“,
godine: 24,
OIB: 123455678,
Hobiji: [„košarka“, „plivanje“, „slikanje“]
}
```

Kod 2 Primjer zapisa nekog dokumenta

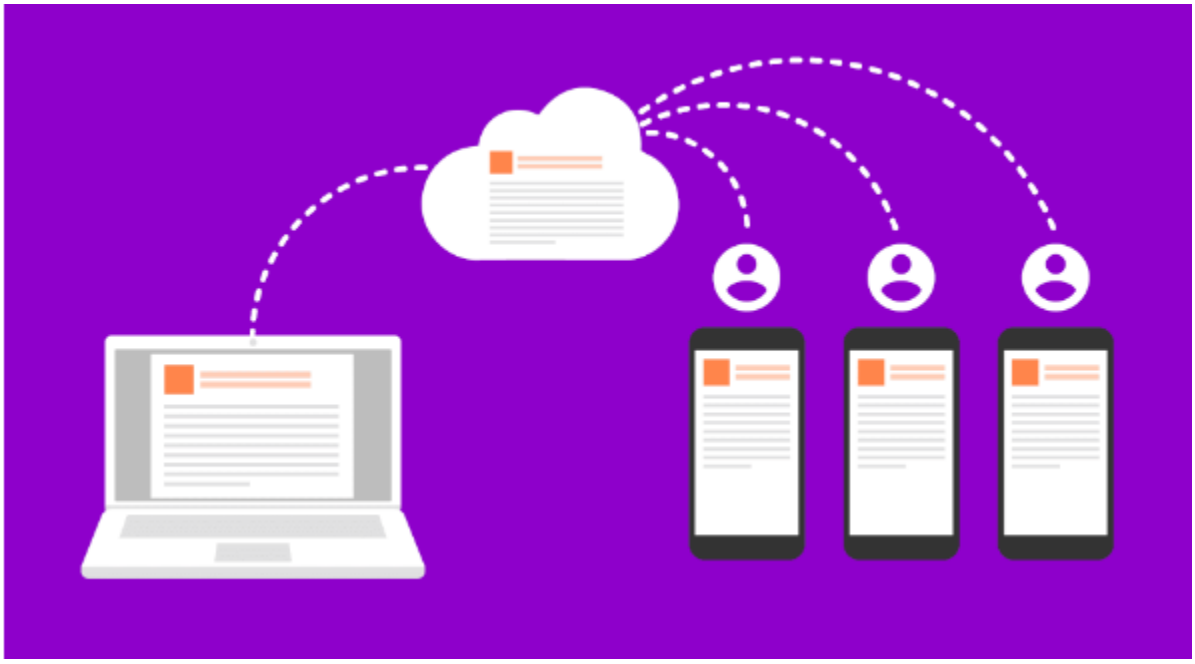
Umjesto korištenja tablica i redaka kao kod relacijskih, arhitektura MonogDB-a se sastoji od zbirki i dokumenata. Zbog toga se ovaj alat koristi u aplikacijama koje pohranjuju veliku količinu podataka.

MongoDB Inc je razvio MongoDB bazu podataka. Osim toga, prateći ostale pružatelje NoSQL baza podataka pokrenuo je i bazu podataka u oblaku pod nazivom MongoDB Atlas 2016. Atlas radi na AWS-u, *Microsoft Azureu* i *Google Cloud platformi*. Kasnije su izdali platformu pod nazivom Stitch za razvoj aplikacija na MongoDB Atlasu s planovima da se proširi i na lokalnu upotrebu [21].

3.2.2. Firebase Realtime

Firebase Realtime je još jedna dokument baza podataka koja u oblaku pohranjuje podatke kao JSON. Umjesto klasičnih HTTP zahtjeva, Firebase koristi sinkronizaciju podataka, odnosno svaki put kada se podatci promijene bilo koji uređaj koji je povezan na bazu ažurira podatke unutar milisekundi. Dosta aplikacija prestane reagirati dok su izvan mreže, no Firebase pruža izvrsnu izvanmrežnu podršku jer čuva unutarnju predmemoriju svih podataka. Kad nema internetske mreže, aplikacija koristi upravo te podatke iz predmemorije i omogućuje aplikacijama da i dalje reagiraju. Kada se ponovno povežemo na Internet, baza reagira i sinkronizira promjene lokalnih

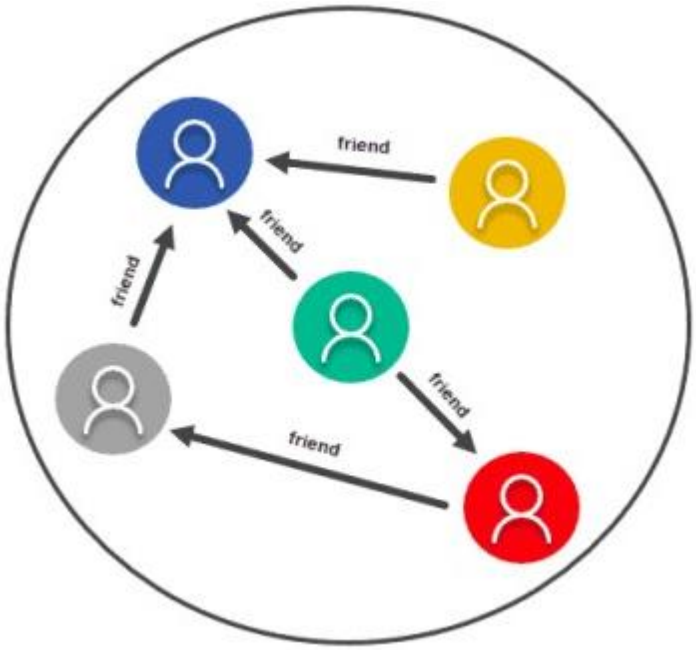
podataka s udaljenim ažuriranim podatcima koji su se promjenili dok je korisnik bio izvan mreže. Firebase nije potpuno besplatna. Postoje odreženi planovi cijena. Ukoliko želite da vaša aplikacija bude skalirana, morate platiti broj veza, korištenje diska i korištenje mreže. Kao što je već spomenuto, podatci se spremaju u JSON formatu, no cijela baza je jedno veliko JSON stablo s više čvorova [22].



Slika 12 Realtime Firebase [23]

3.3. Graf baze podataka

Graf baze podataka koriste fleksibilan grafički prikaz za upravljanje podatcima. Grafikoni se sastoje od dva elementa – čvorova i rubova. Čvorovi služe za pohranu entiteta, dok rubovi služe za pohranu odnosa između entiteta. Ovakav odnos između entiteta nam omogućuje da izravno povezujemo podatke u pohrani i u mnogim slučajevima je njihovo dohvaćanje moguće samo s jednom operacijom. Budući da je ova vrsta pohrane podataka uistinu specifična, to nije često korištena vrsta NoSQL baza podataka. No, postoje određeni slučajevi kada se koristi i u kojima je isključivo grafički prikaz idealno rješenje. Na primjer, društvene mreže često koriste grafikone za pohranu informacija o tome kako su njihovi korisnici povezani (opcija zajedničkih prijatelja na Facebook-u). Najčešće graf baze podataka su RedisGraph, Neo4j, OrientDB i druge [11].



Slika 13 Graf nerelacijska baza podataka. Preuzeto iz [11]

3.4. Stupac baze podataka

Stupac ili stupčane baze podataka su zadnja vrsta NoSQL baza podataka na koju ćemo se osvrnuti. U njima se podatci pohranjuju i grupiraju u zasebno pohranjene stupce. Takvi stupci funkcioniraju slično tablicama u relacijskim bazama podataka. Međutim, za razliku od tradicionalnih baza podataka, ova vrsta NoSQL baza su dosta fleksibilnije. One nemaju unaprijed definirane ključeve niti nazive stupaca. Najznačajnija prednost baza podataka usmjerenih na stupce jest ta da možemo pohraniti velike količine podataka unutar jednog stupca. Ova značajka omogućuje smanjenje resursa diska i vremena potrebnog za dohvaćanje informacija s njega. Također, izvrsni su u situacijama kada moramo širiti podatke na više poslužitelja [11].

Na sljedećim slikama možemo uočiti razliku pohranjivanja relacijskih baza i stupac baza podataka.

brojRacuna	nazivRacuna	idKlijenta
75839	racun1	1
98271	racun2	1
32495	racun3	2
77990	racun4	2
15633	racun5	2

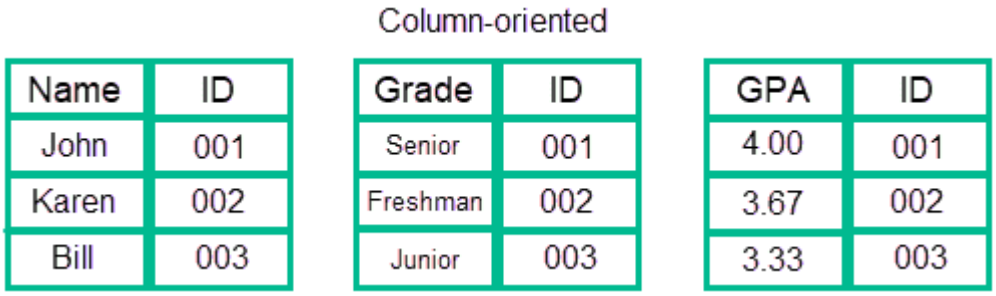
Slika 14 Primjer pohrane relacijske baze podataka

U stupac tipu NoSQL baze podataka ovi podaci će biti pohranjeni na sljedeći način:

```
(brojRacuna) 75839, 98271, 32495, 77990, 15633
(nazivRacuna) racun1, racun2, racun3, racun4, racun5
(idKlijenta) 1, 1, 2, 2, 2
```

Slika 15 Primjer pohrane stuac tipa NoSQL baze podataka

Neke od stupac baza podataka su Apache Cassandra, HBase, CosmoDB i druge.

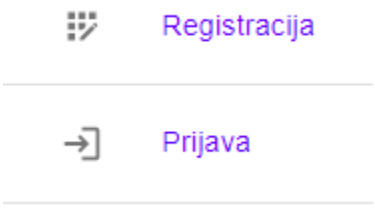


Slika 16 Primjer stupac nerelacijske baze podataka. Preuzeto iz [11]

4. Projektni zadatak – aplikacija za izdavanje računa

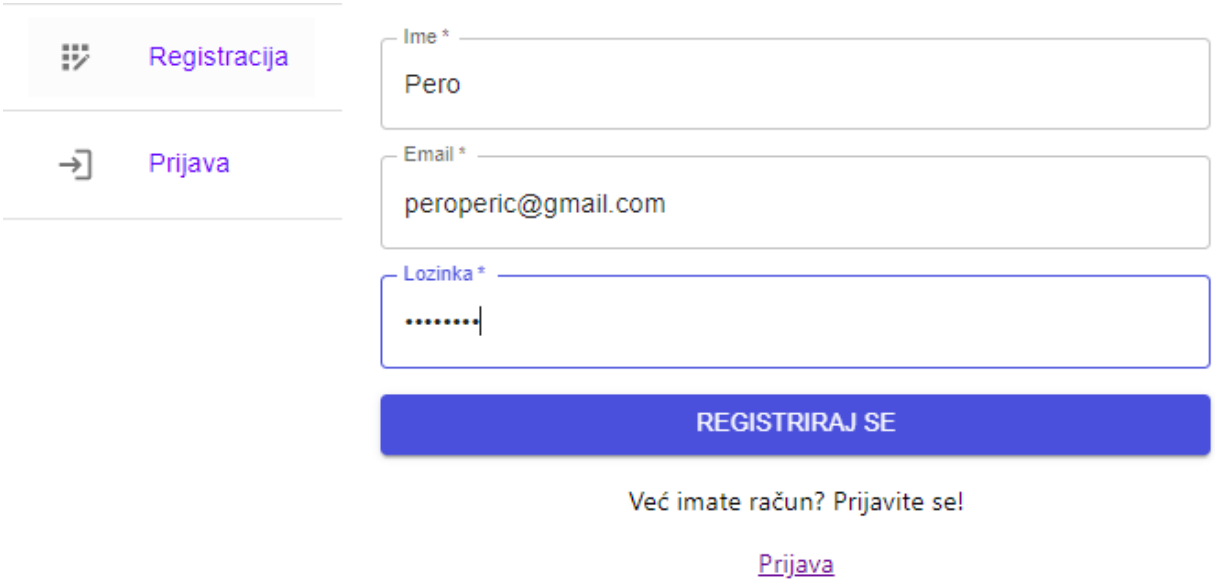
Za svoj projektni zadatak sam odabrala izradu aplikacije za izdavanje računa. Aplikacija funkcionira na način da se korisnici mogu registrirati i ulogirati na svoj račun kako bi sami mogli izdavati račune za svoje potrebe. U nastavku ću objasniti čitav postupak izdavanja računa nekog korisnika.

Za početak se korisniku otvara stranica na kojoj ima opciju registracije ili prijave. Odaberemo opciju „Registracija“ kako bismo dodali novi račun.



Slika 17 Izbornik za registraciju i prijavu

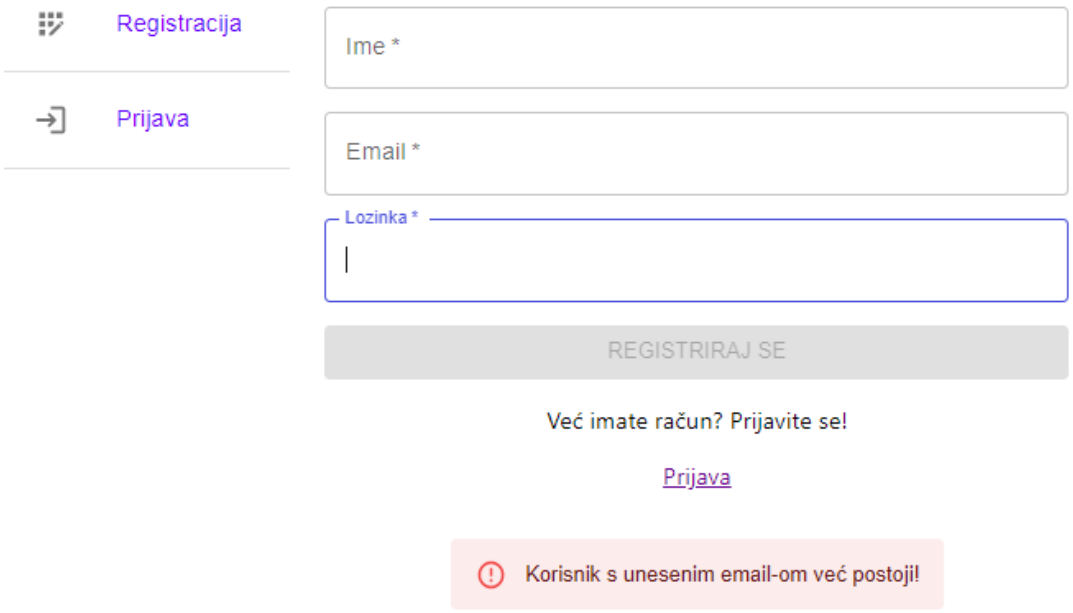
Odabirom opcije „Registracija“ otvara se sljedeći prozor u kojem unosimo ime, email i lozinku za naš primjer. U ovom primjeru ćemo kreirati račun s e-mailom „peroperic@gmail.com“.



Slika 18 Registriranje novog korisnika

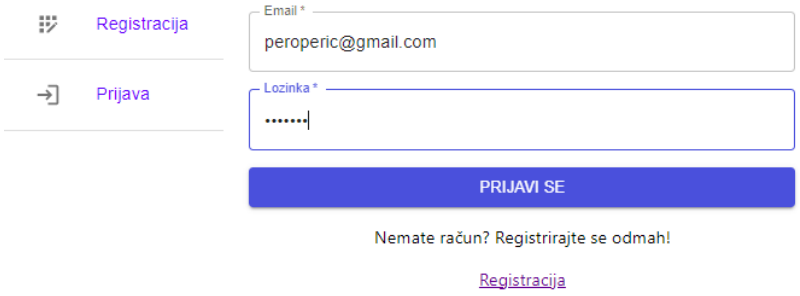
Klikom na botun „Registriraj se“ naš korisnik je uspješno spremljen u bazu i automatski je ulogiran na svoj račun, što mu omogućuje pristup prozoru sa svim opcijama kreiranja računa.

Ukoliko smo već prethodno imali kreiran račun s e-mailom „peroperic@gmail.com“, nije se potrebno ponovno registrirati. Odnosno, prilikom takvog postupka aplikacija bi vratila upozorenje da smo se pokušali registrirati s postojećim e-mailom. U tom slučaju potrebno je prijaviti se na već postojeći račun.



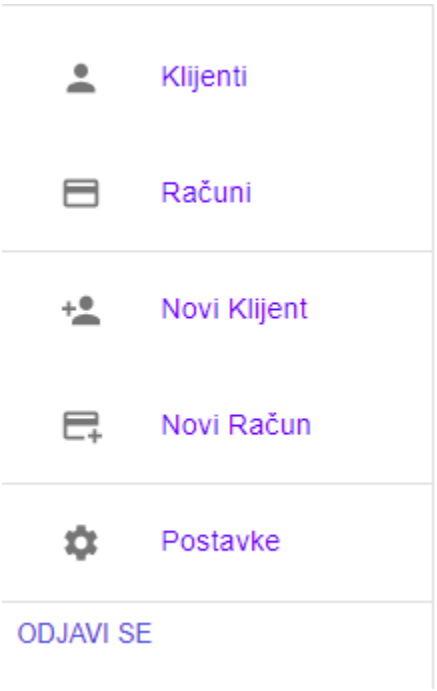
Slika 19 Prikaz greške prilikom registracije s postojećim korisnikom

U sljedećem koraku se prijavljujemo na naš postojeći račun.



Slika 20 Prikaz greške prilikom registracije s postojećim korisnikom

Nakon što smo kreirali naš račun ili se prijavili na već postojeći, otvara se stranica sa dostupnim sljedećim izbornikom:

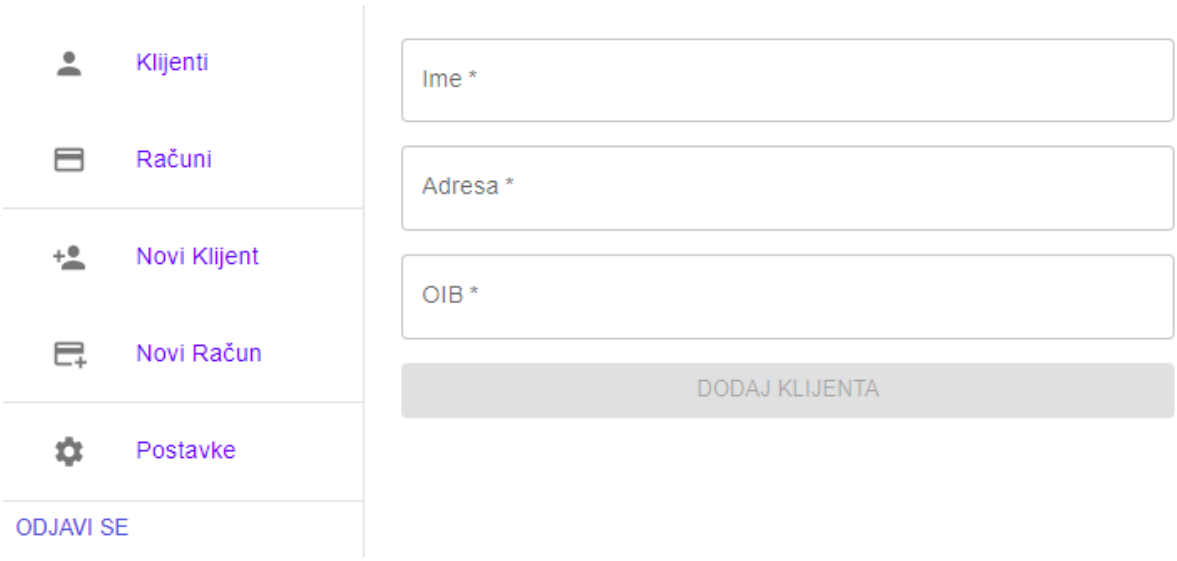


Slika 21 Prikaz izgleda izbornika nakon prijave korisnika

Klikom na opciju „Klijenti“ imamo pristup listi svih klijenata s kojima naš korisnik surađuje. Svaki korisnik mora ručno dodati klijente s kojima surađuje koristeći opciju „Novi klijenti“.

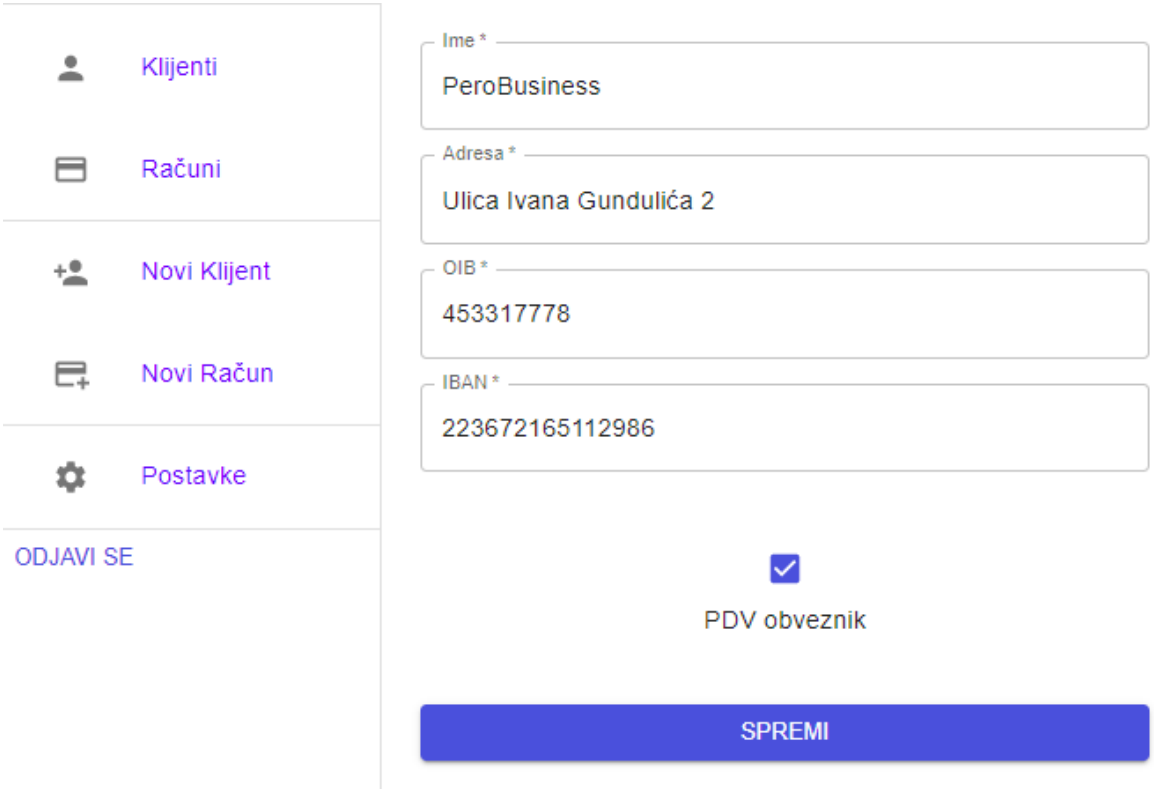
U slučaju da je naš korisnik završio suradnju s određenim klijentom, može ga u bilo kojem trenutku ukloniti iz liste klikom na opciju „Obriši“.

Slika 22 Prikaz liste svih klijenata nekog korisnika



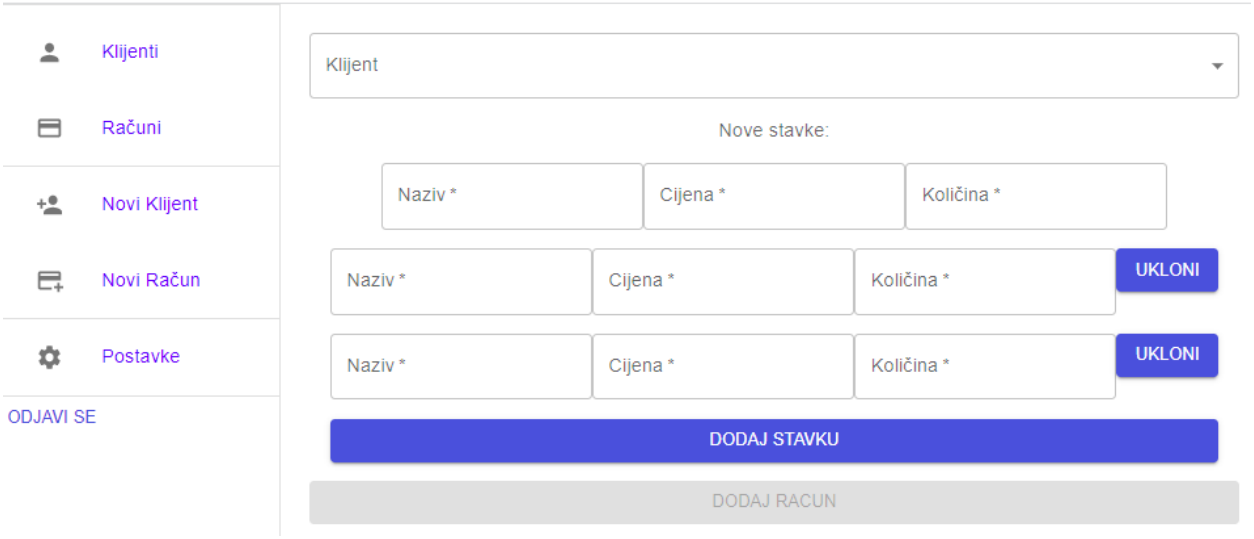
Slika 23 Mogućnost dodavanja novih klijenata

Korisnik „Pero“ ima pristup informacijama firme u opciji „Postavke“ koje može ažurirati u bilo kojem trenutku.



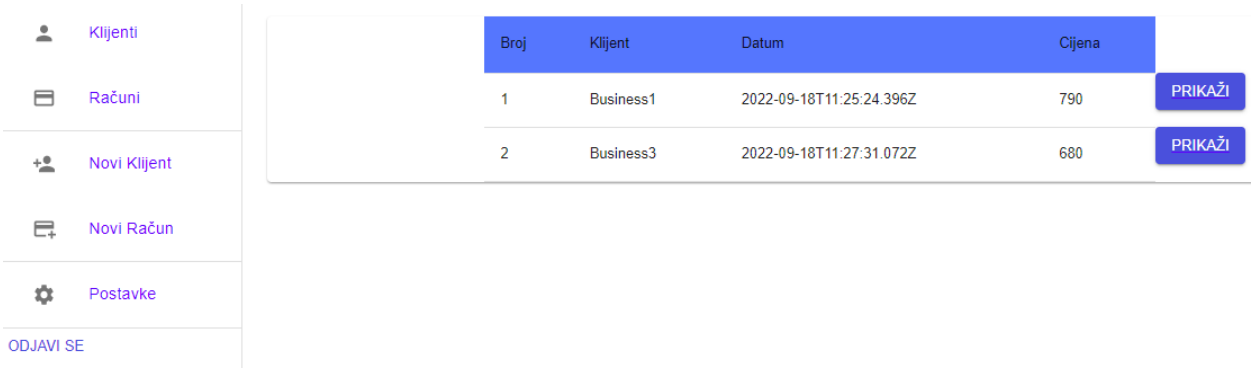
Slika 24 Ažuriranje podataka preko opcije „Postavke“

U sljedećem koraku, korisnik Pero ima mogućnost dodavanja računa klikom na opciju „Novi račun“. Korisniku se otvara prozor sa priložene slike u kojem odabire klijenta za kojeg izdaje račun (klikom na opciju klijent ponuđeni su svi klijenti s kojima Pero surađuje). Također, Pero može dodati jednu ili više stavki. Stavke se sastoje od polja naziv, cijena i količina koja su obavezna za unos.

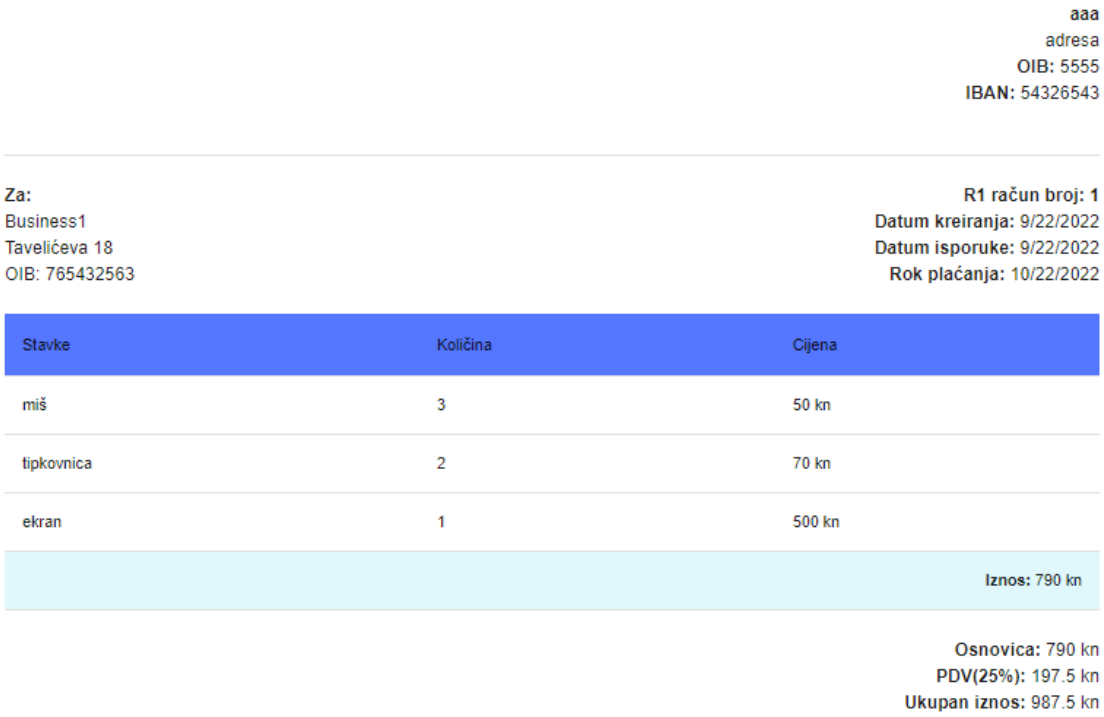


Slika 25 Mogućnost kreiranja novih računa

Korisnik Pero naravno ima i uvid u sve svoje kreirane račune koristeći opciju „Računi“. Klikom na opciju otvara se prozor s listom svih računa, te klikom na botun „Prikaži“ korisnik može pristupiti postavkama svakog pojedinačnog računa.

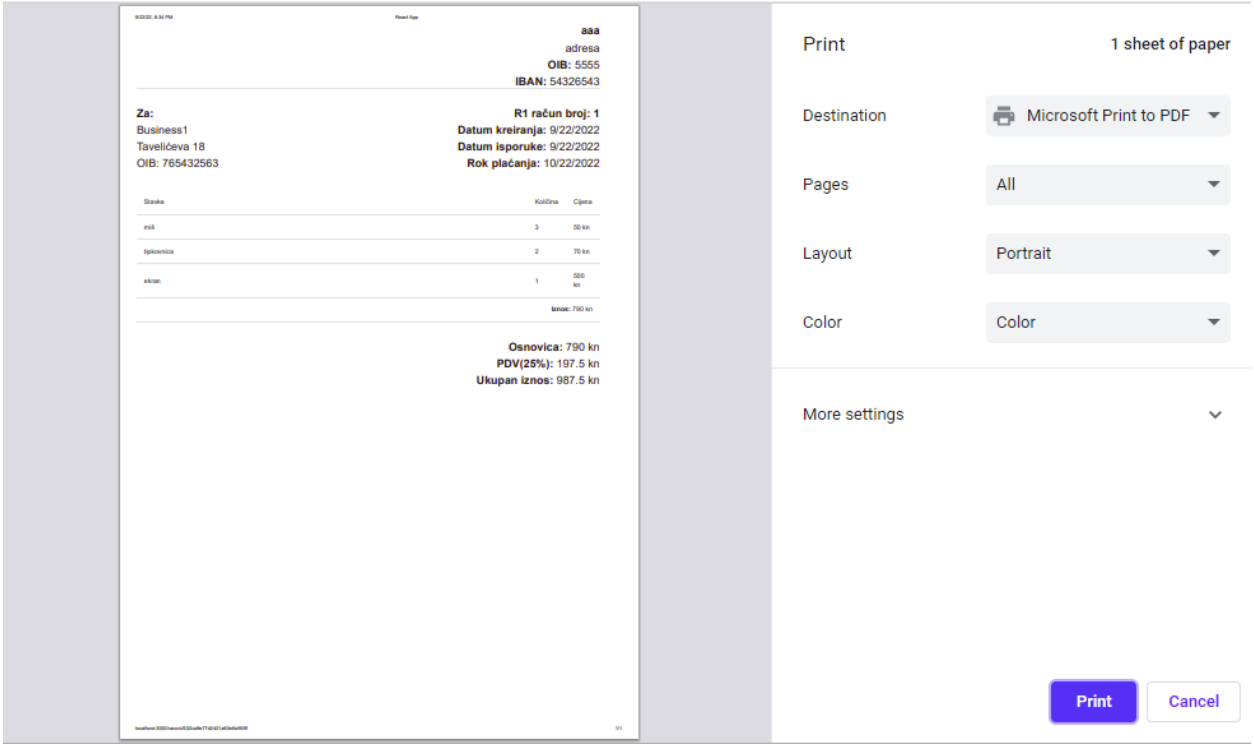


Slika 26 Prikaz liste svih računa nekog korisnika



Slika 27 Prikaz jednog od postojećih računa

Osim toga, korisnik uvijek može isprintati bilo koji od svojih postojećih računa. Prvo je potrebno odabrati određeni račun kao u prethodnom koraku, a nakon toga klikom na „CRTL + P“ korisniku se otvara prozor za printanje računa.



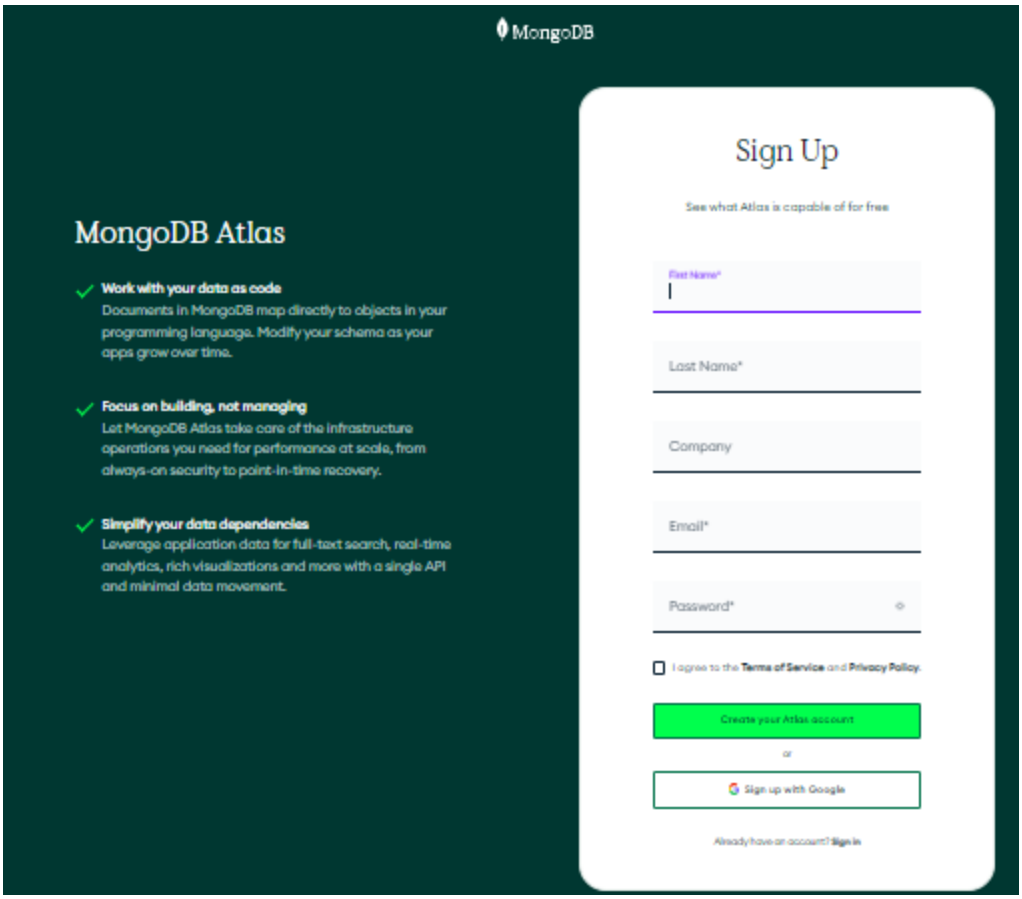
Slika 28 Prikaz procesa printanja računa

4.1. Implementacija MongoDB

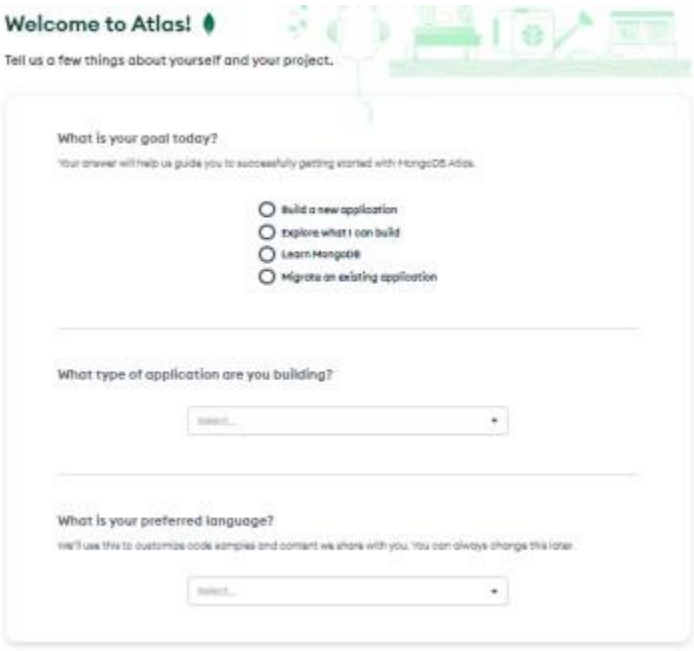
Nakon pregleda funkcionalnosti aplikacije, prelazimo na dio kodiranja. Za prvu implementaciju ovog projekta odabrala sam MongoDB bazu podataka. U jednom od prethodnih poglavlja smo se malo bolje upoznali s ovom vrstom baza, pa ćemo sada prijeći na samu implementaciju.

4.1.1. MongoDB Atlas

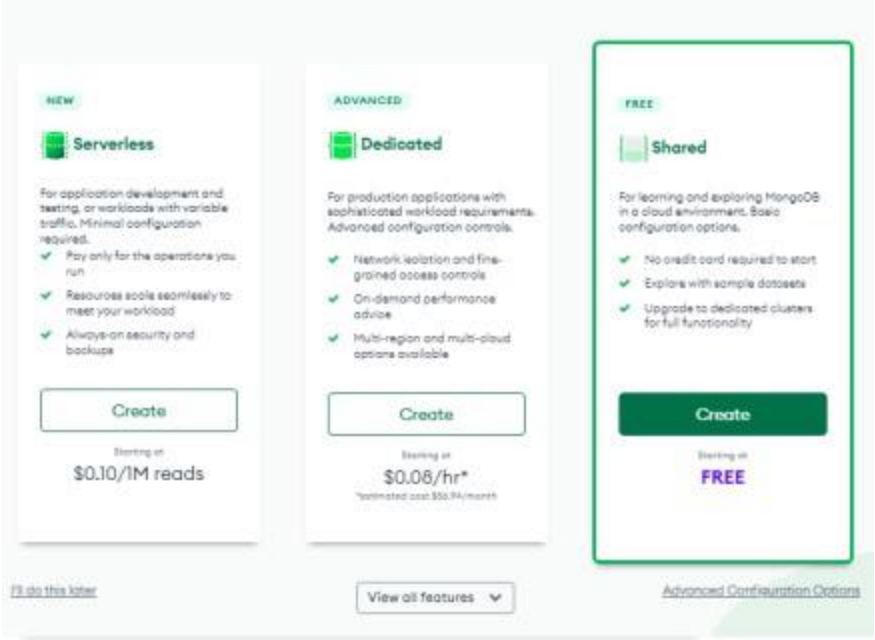
Za početak je potrebno kreirati korisnički račun na *cloud* uslugi MongoDB Atlas kojeg smo već spomenuli. Nakon kreiranja računa, potrebno je unijeti početne informacije, odabrati JavaScript kao željeni jezik i odabrati besplatnu opciju korištenja.



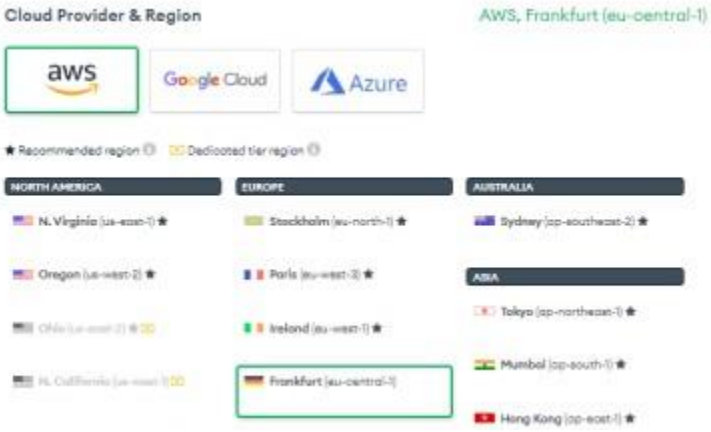
Slika 29 Izbornik za kreiranje korisničkog računa



Slika 30 Odabir potrebnih opcija

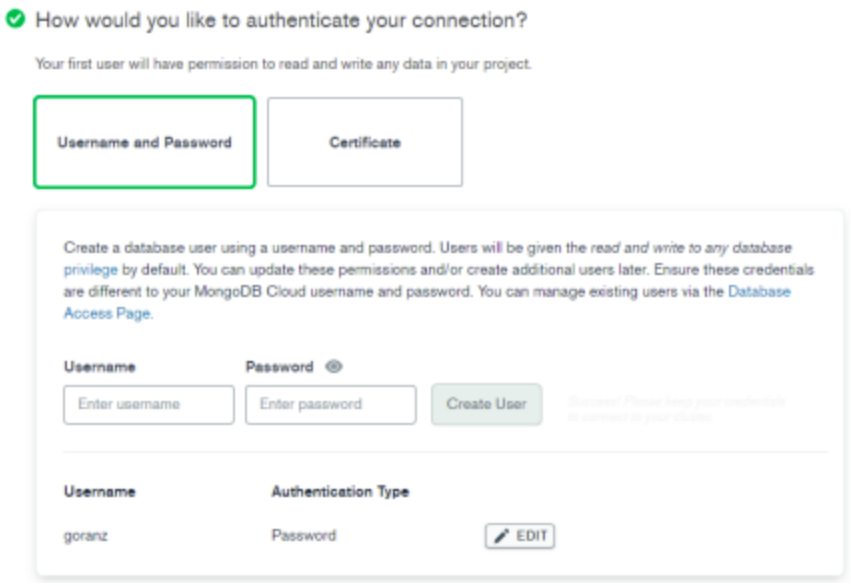


Slika 31 Odabir besplatne verzije MongoDB Atlas-a



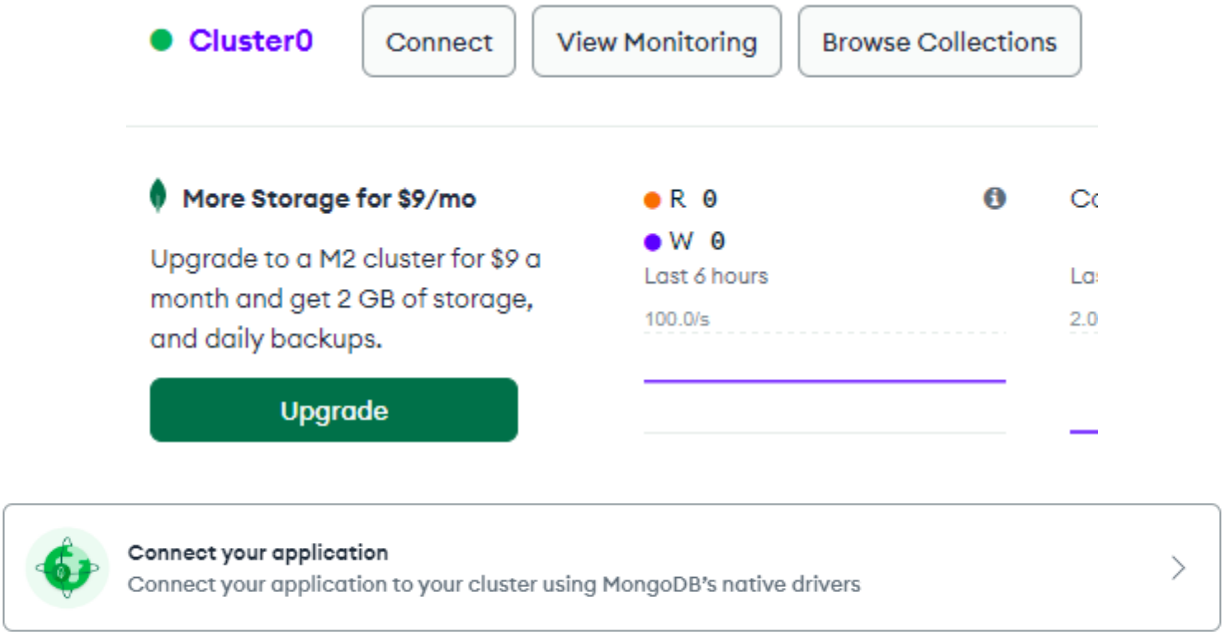
Slika 32 Odabir poslužitelja (AWS) i europske regije za lokaciju (Frankfurt)

Nakon svih ovih koraka, započinje izrada *Clustera* koja može potrajati nekoliko minuta. Zatim je potrebno stvoriti korisnika koji će imati pristup bazi, te dodati adrese sa kojih je dozvoljen pristup bazi (preko opcije *„Network Access“*).



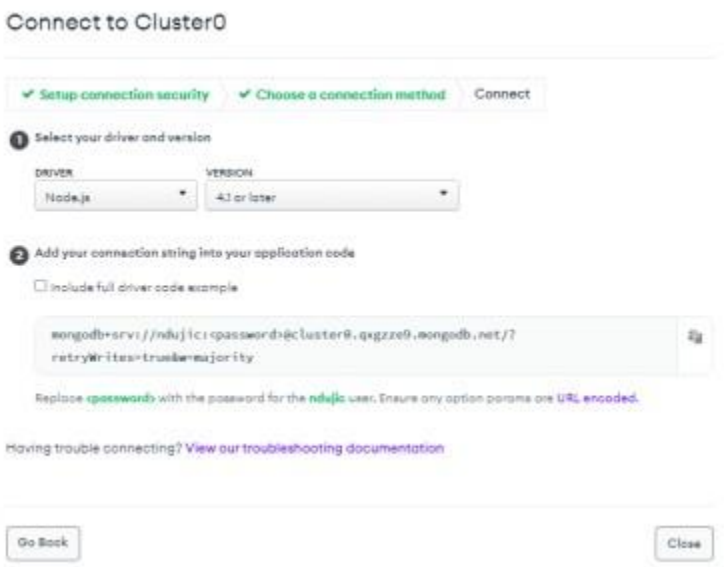
Slika 33 Kreiranje korisnika za pristupanje bazi

Nakon što se *Cluster* kreirao, potrebno je odabrati opciju *„Connect“*, a zatim *„Connect your application“*.



Slika 34 Opcije „*Connect*“ i „*Connect your application*“

Nakon odabranog, otvara se izbornik s dostupnim *connection string-om* preko kojeg bazu spajamo s aplikacijom.



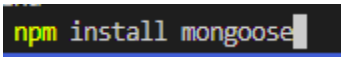
Slika 35 Prikaz *connection string-a*

4.1.2. Mongoose

Mongoose je ODM (engl. *Object Data Modeling*) *library* za MongoDB i Node.js. Upravlja odnosima među podatcima, a validacija se obavlja pomoću *scheme*. *Schema* definira koja se točno polja nalaze u svakom dokumentu i kojeg su oni tipa. Uz to, pomoću nje možemo postavljati validacijske uvjete i nekakve zadane vrijednosti. *Schema* može imati proizvoljan broj polja. S druge strane imamo model, koji se definira pomoću gotove *scheme*. Pomoću njega baratamo objektima zadanog tipa (CRUD operacije):

- Model se veže uz kolekciju dokumenata u Mongo bazi – svi dokumenti u toj kolekciji će pratiti *schemu* koju smo koristili za definiranje modela.

Prvo je potrebno u terminalu instalirati Mongoose *library* pomoću sljedeće naredbe:



Slika 36 Instalacija Mongoose *library-a*

U sljedećem koraku instaliramo novi paket „dotenv“ pomoću kojeg uvodimo *environment* varijable (štitimo određene podatke poput lozinke).

```
npm install dotenv --save-dev
```

Slika 37 Instalacija „dotenv“ paketa

Kreiramo novu mapu „.env“ u kojem definiramo naše *environment* varijable. Imena varijabli se u pravilu pišu velikim slovima.

projekt-ndujic > backend > .env

```
1  ATLAS_PASS=kakodane
2  ATLAS_USER=ndujic
3  PORT=3001
4  SECRET = "nekisigurnistringkojinesmiprocuritdofrontenda";
```

Slika 38 Dodavanje .env foldera i *environment* varijabli

Također, potrebno je u „.gitignore“ dodati sljedeći zapis:

projekt-ndujic > .gitignore

```
1  node_modules
2  .env
```

Slika 39 Modifikacija .gitignore

Naravno, kako bi ovo funkcioniralo potrebno je i modificirati skriptu za pokretanje poslužitelja.

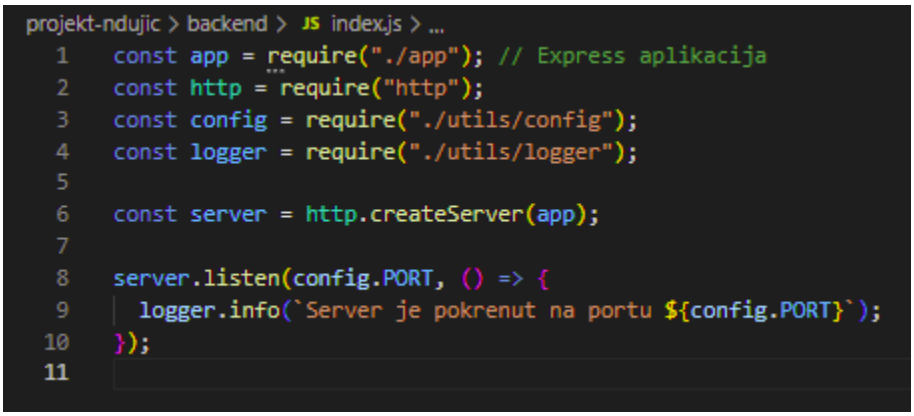
```
"scripts": {
  "start": "cross-env NODE_ENV=production node index.js",
  "test": "cross-env NODE_ENV=test jest --verbose --runInBand",
  "dev": "cross-env NODE_ENV=development nodemon -r dotenv/config index.js"
},
```

Slika 40 Modificiranje skripte za pokretanje

U „config.js“ definiramo sve potrebne podatke za spajanje na bazu, kao i *connection string*.

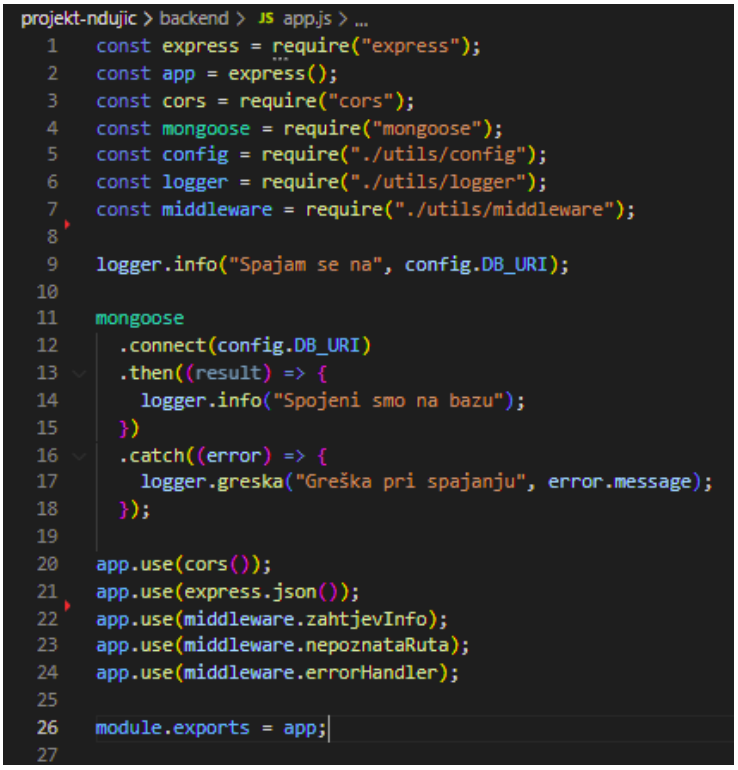
Slika 41 Definiranje podataka za spajanje na bazu podataka

Potrebno je kreirati http server na sljedeći način:



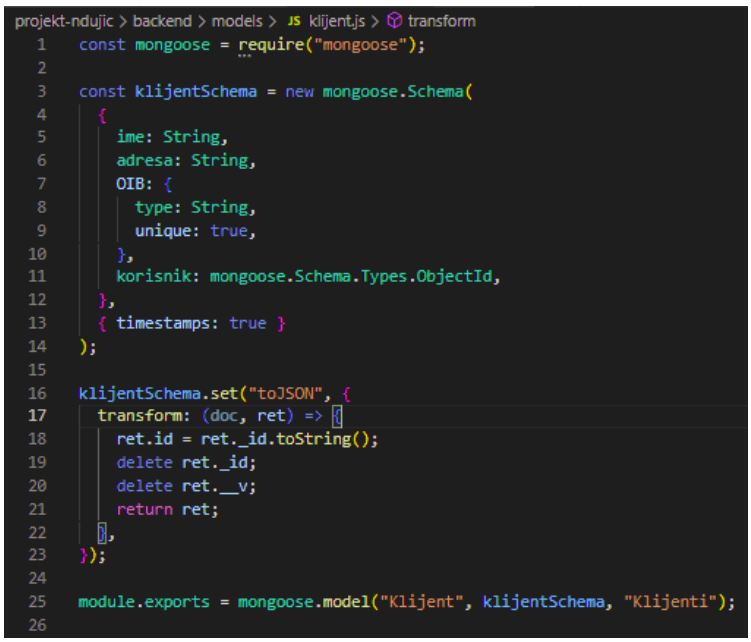
Slika 42 Kreiranje i pokretanje http servera

U „app.js“ učitavamo naš Mongoose i sve ostalo što je potrebno. Dodamo poziv metode *„connect“* kako bi se prilikom pokretanja poslužitelja spojili sa bazom. Koristimo *callback* funkcije *then* i *catch* za ispis poruke u terminalu prilikom spajanja.



Slika 43 Spajanje na bazu

U sljedećem koraku kreiramo novu mapu „models“ u kojoj ćemo definirati sve naše modele. Potrebno je kreirati modele za klijente, račune, korisnike i postavke.



Slika 44 Primjer modela klijenta

Naša baza podataka kao rezultat vraća niz objekata koji nisu u JSON formatu. Potrebno je pretvoriti te objekte u JSON, te ukloniti „_v“ polje koje nam nije potrebno. Također, zamijenili smo polje „_id“ s poljem id (želimo da nam svaki objekt ima svoj id).

```
klijentSchema.set("toJSON", {
  transform: (doc, ret) => {
    ret.id = ret._id.toString();
    delete ret._id;
    delete ret.__v;
    return ret;
  },
});
```

Slika 45 Modificiranje scheme klijenta

Kreiramo *GET* rutu kako bismo dohvatili podatke iz naše baze. Potrebno je *importati* model „Klijent“ nad kojim koristimo metodu „*find*“. Pomoću nje možemo dohvatiti sve podatke ili podatke koji ispunjavaju određeni uvjet koji sami postavimo.

```
projekt-ndujic > backend > controllers > JS klijenti.js > klijentiRouter.post("/") callback > [e] klije
1   const klijentiRouter = require("express").Router();
2   const Klijent = require("../models/klijent");
3
4   klijentiRouter.get("/", async (req, res) => {
5     try {
6       const klijenti = await Klijent.find({ korisnik: req.user.id });
7       res.json(klijenti);
8     } catch (err) {
9       res.status(400).json({ message: "Klijent ne postoji" });
10    }
11  });
12
```

Slika 46 Ruta za dohvat svih korisnika iz baze

Novog klijenta je potrebno stvoriti preko konstruktora modela. *POST* ruta za dodavanje novih klijenata bi trebala izgledati ovako:

```
13  klijentiRouter.post("/", async (req, res) => {
14    const sadrzaj = req.body;
15
16    const klijent = new Klijent({
17      ime: sadrzaj.ime,
18      adresa: sadrzaj.adresa,
19      OIB: sadrzaj.OIB,
20      korisnik: req.user.id,
21    });
22
23    const noviKlijent = await klijent.save();
24    res.json(noviKlijent);
25  });
```

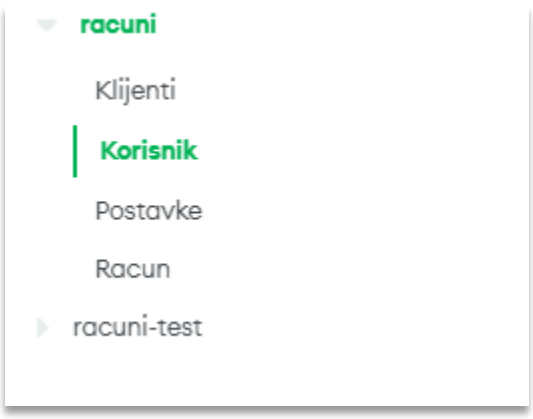
Slika 47 Ruta za dodavanje novih klijenata

Kako bismo izbrisali određenog klijenta iz baze definiramo *DELETE* rutu. Nad našim modelom „Klijent“ koristimo metodu *„findByIdAndRemove“* koja dohvaća određeni podatak preko id-a i uklanja ga iz baze.

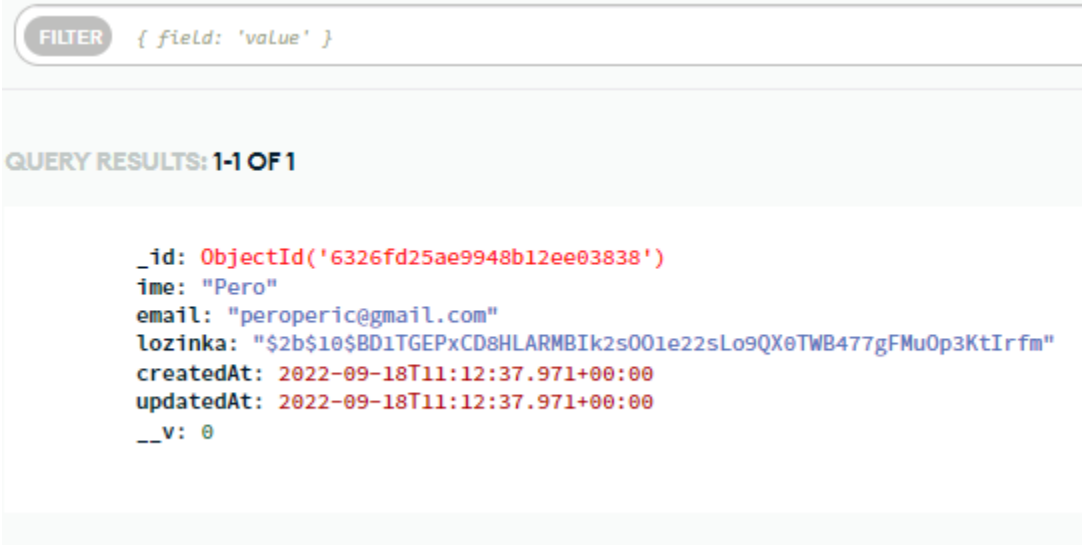
```
27  klijentiRouter.delete("/:id", (req, res) => {
28    Klijent.findByIdAndRemove(req.params.id)
29      .then((result) => {
30        res.status(204).end();
31      })
32      .catch((err) => next(err));
33  });
```

Slika 48 Ruta za brisanje klijenta iz baze

U prvom dijelu poglavlja smo objasnili kako aplikacija funkcionira, te smo kreirali korisnika Peru. Dakle, otvaranjem naše baze podataka (MongoDB) uočavamo kolekciju „Korisnik“ koja sadrži jedan dokument. U dokumentu su pohranjeni podatci o korisniku Peri. Stoga, možemo zaključiti da su rute za korisničku registraciju i prijavu, kao i sami model korisnika uspješno kreirali.



Slika 49 Baza podataka „racuni“ s kreiranim kolekcijama

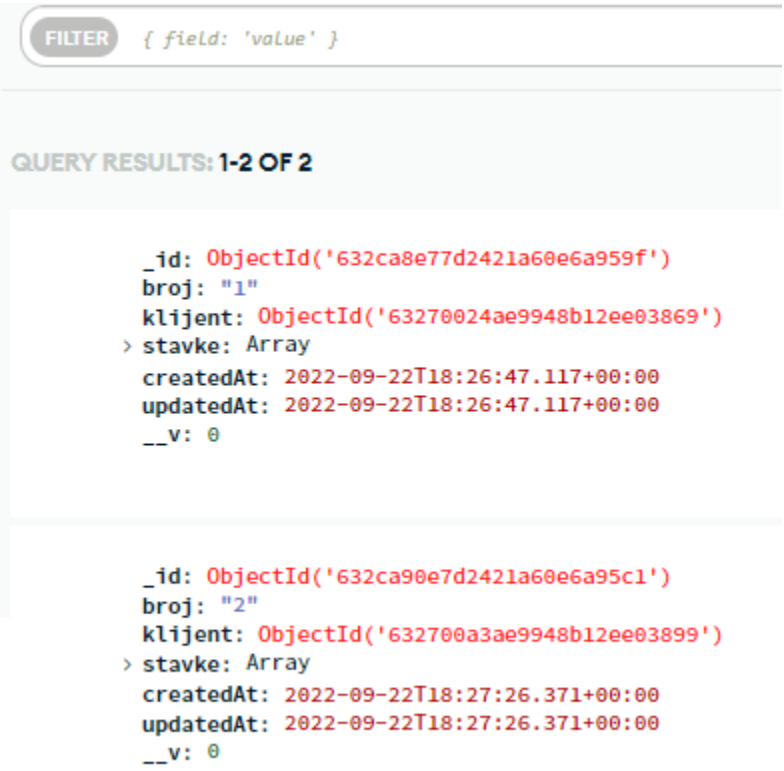


Slika 50 Dokument unutar kolekcije Korisnik

Primjećujemo i kolekciju „Klijenti“. Prisjetimo se, korisnik Pero je kreirao tri klijenta (Business1, Business2 i Business3). Možemo vidjeti kako je i ovaj dio uspješno prošao tako što su se sva tri klijenta spremila u kolekciju „Klijenti“.

Slika 51 Prikaz dodanih klijenata

Na kraju, još ćemo pogledati Perine kreirane račune. Vidimo da je i ovaj dio prošao bez ikakvih problema. Podatci su uspješno spremljeni u kolekciju „Racun“.



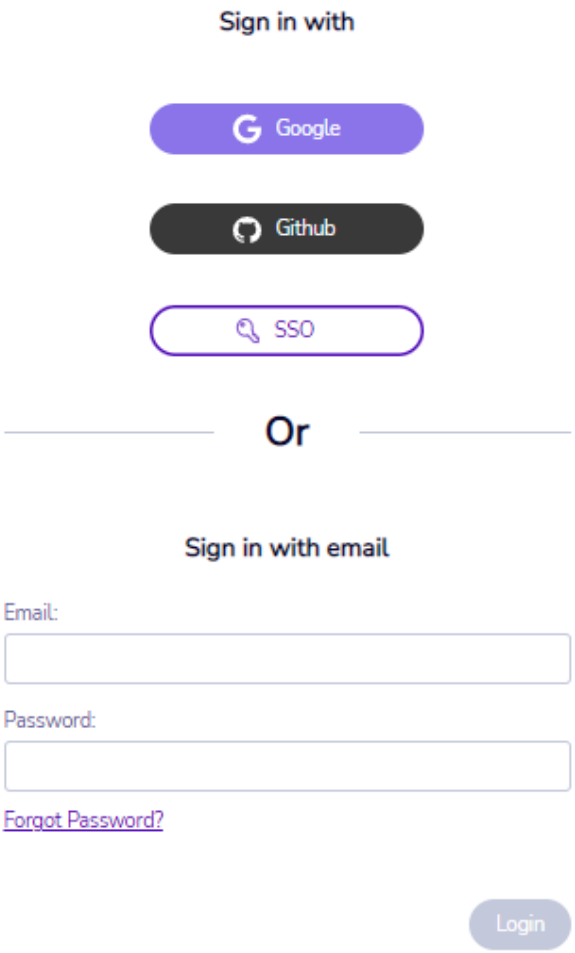
Slika 52 Prikaz kolekcije „Racun“

4.2. Implementacija Redis

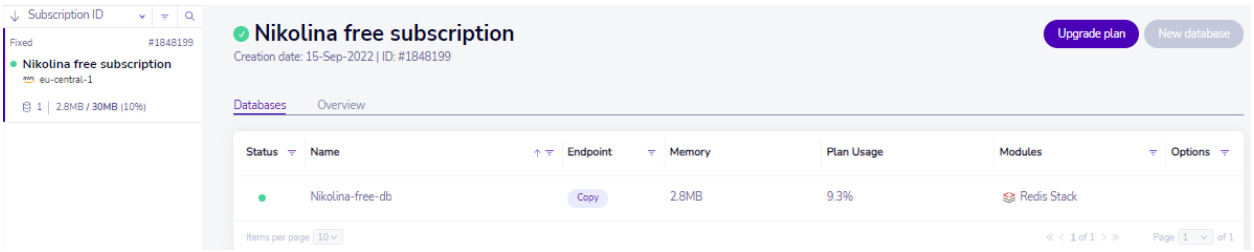
Nakon implementacija Monga, prelazimo na implementaciju Redisa. U jednom od prethodnih poglavlja smo se malo bolje upoznali s ovom vrstom baza, pa ćemo sada prijeći na samu implementaciju.

4.2.1. Redis Cloud

Redis Cloud je upravljana usluga smještena u oblaku (engl. *Cloud*) koja omogućuje jednostavno i brzo pokretanje podataka naših aplikacija. Dostupan je na AWS-u, Microsoft Azure-u i Google Cloud-u.

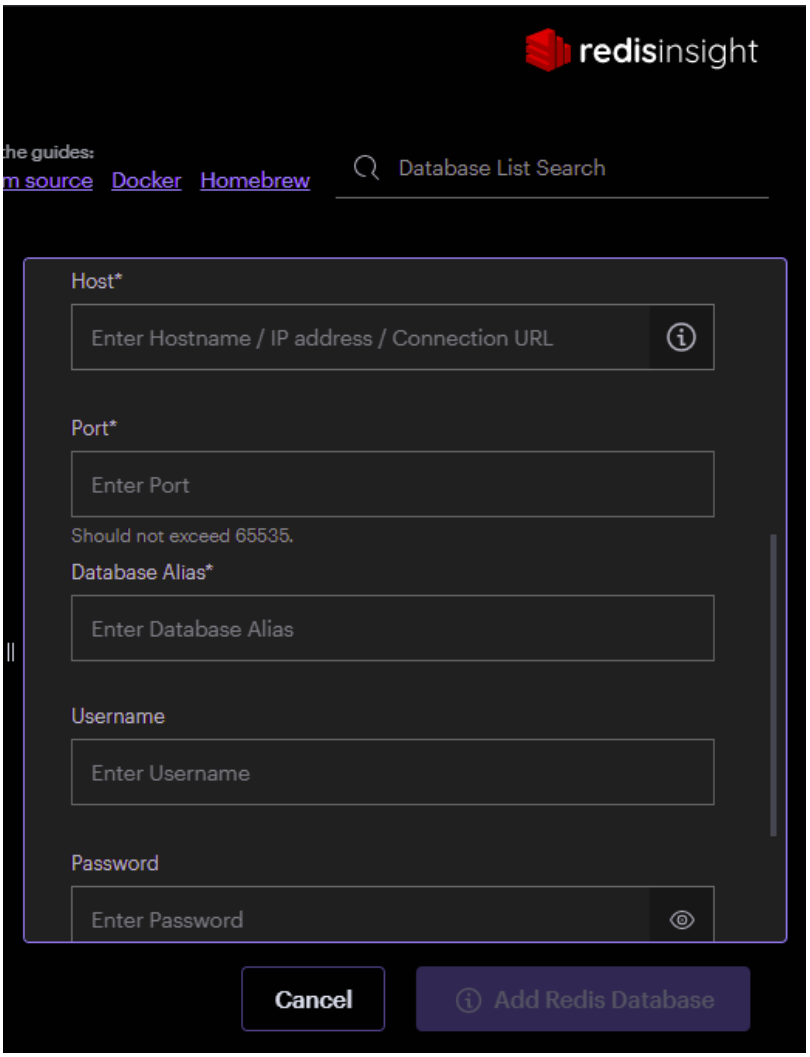


Slika 53 Prijava na Redis Cloud



Slika 54 Automatski kreirana Redis baza

U sljedećem koraku ćemo instalirati Redis Insight. To je alat kojeg povezujemo s Redis Cloud-om i on nam omogućuje vizualizaciju podataka te njihovo upravljanje. U Redis Insight-u imamo opciju za dodavanje naše baze podataka, te je potrebno kopirati određene podatke s *clouda* kako bismo se povezali.



Slika 55 Proces spajanja na bazu podataka

Nakon toga prelazimo na samu aplikaciju. Potrebno je instalirati Redis OM *library* pomoću kojeg se preslikavaju Redis tipovi podataka u JavaScript objekte. Instaliramo Redis-om pomoću sljedeće naredbe:

```
npm install redis-om --save
```

Slika 56 Naredba za instalaciju Redis-om library-a

U sljedećem koraku, postavljamo našeg klijenta. Klasa klijent nam omogućuje „razgovor“ s Redisom u ime Redis OM-a. U folderu „models“ kreiramo „client.js“ u koji dodajemo sljedeći kod:

projekt-ndujic > backend > models > JS client.js > ...

```
1  const { Client } = require("redis-om");
2
3  const url = process.env.REDIS_URL;
4
5  const getClient = new Client().open(url);
6  console.log(url);
7  module.exports = getClient;
8
```

Slika 57 Client.js

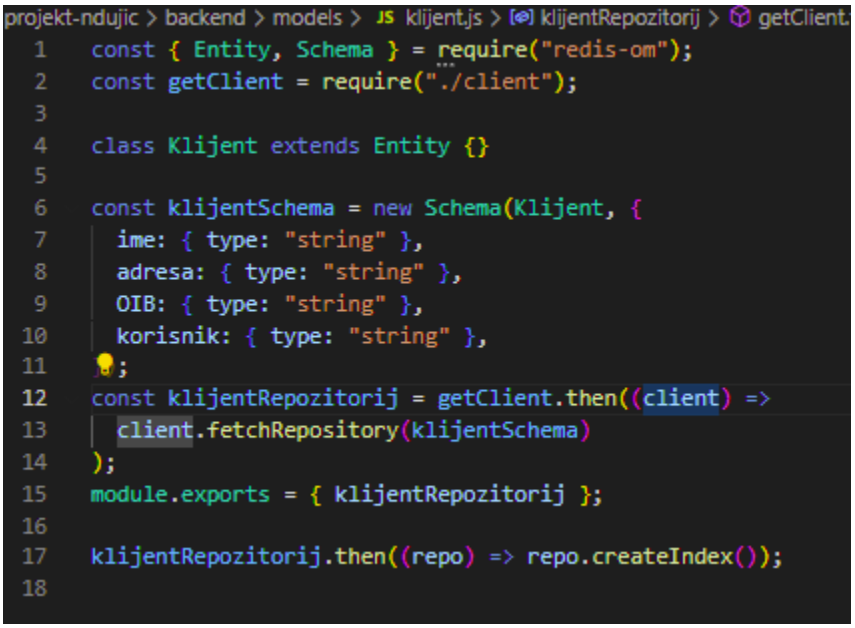
Naravno potrebno je REDIS_URL dodati u naš „.env“ file koji sada izgleda ovako:

projekt-ndujic > backend > .env

```
1  PORT=8085
2  SECRET = "nekisigurnistringkojinesmiprocuritdofrontenda";
3  REDIS_URL = "redis://default:6qU2mOEvArGmffObXr0JuI2GcrM6pw8x@redis-15182.c135.eu-central-1-1.ec2.cloud.redislabs.com:15182"
```

Slika 58 .env file

Nakon što smo kreirali klijenta koji je povezan s Redis-om, potrebno je definirati entitet i *schemu*. U *folderu* „models“ kreiramo *file* „klijent.js“. *Importamo* klasu „Entity“ i „Schema“ iz Redis OM-a. *Schema* mijenja klasu entiteta koju smo joj predali na način da dodajemo vrijednosti različitih tipova podataka. Svi ovi dijelovi su nam potrebni za stvaranje spremišta (repozitorija). Repozitorij je glavno sučelje u Redis OM-u. Daje nam metode za čitanje, pisanje i uklanjanje određenog entiteta. Također, kako bismo mogli pretraživati, bitno je izraditi indeks. To radimo na način da pozivamo „.createIndex()“. Na sljedećoj slici vidimo objašnjene korake:



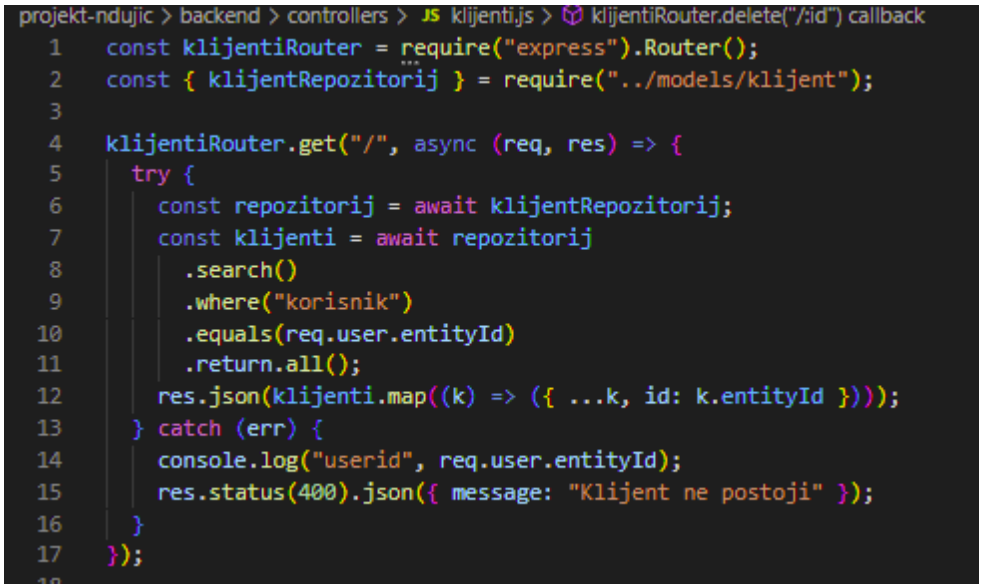
Slika 59 klijent.js file

Sve ove korake potrebno je napraviti i za korisnika, račune i postavke.

Kako bi aplikcija ispravno funkcionirala s novom implementacijom, potrebno je modificirati rute. GET ruta koja je prije izgledala ovako:

Slika 60 MongoDB GET ruta

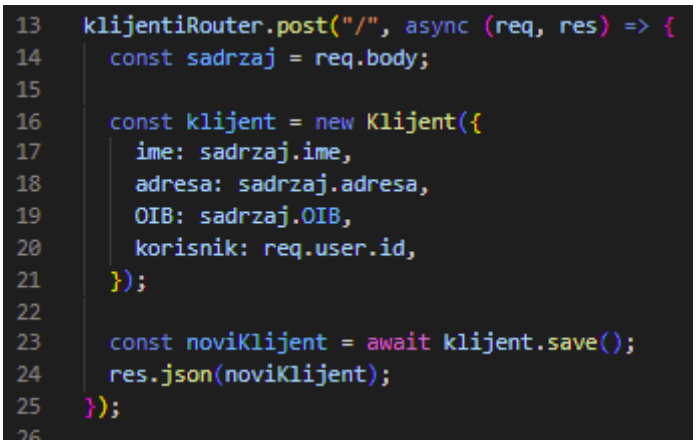
Sada naša GET ruta izgleda ovako:



Slika 61 Redis GET ruta

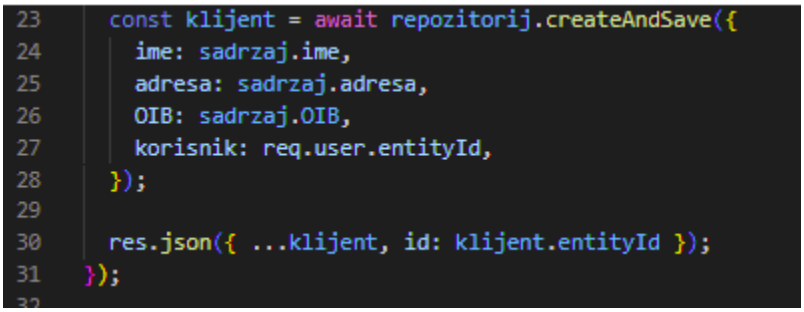
Prisjetimo se, kod Monga smo imali ugrađenu metodu „find()“ koja se koristila nad samim modelom (šaljemo joj objekt sa uvjetima pretrage). Ukoliko je objekt prazan, dohvaćaju se svi podatci koji odgovaraju tom modelu. S druge strane, kod Redisa imamo metodu „search()“, pomoću koje kreiramo upite. Ukoliko ne kreiramo upit, dohvaćamo sve podatke. U prethodnom primjeru smo definirali rutu u kojoj navodimo polje prema kojem želimo filtrirati i vrijednost kojoj ono mora biti jednako. Naziv polja u pozivu „where()“ je naziv polja navedenog u našoj schemi, a „equals()“ je vrijednost tog polja. Koristimo i „return.all()“ koji omogućuje vraćanje svih podataka koji su ispunili određeni uvjet.

Pogledat ćemo i POST rutu koja je kod Monga izgledala ovako:



Slika 62 MongoDB POST ruta

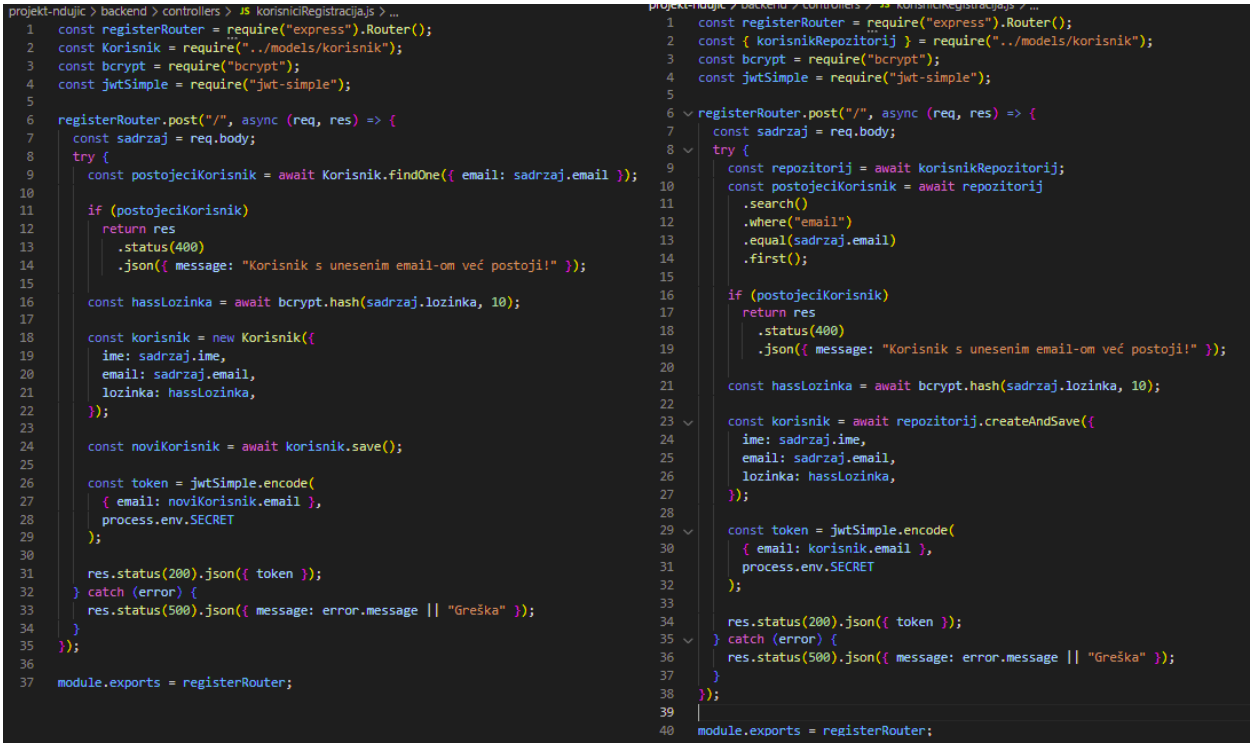
POST ruta implementirana kod Redisa izgleda ovako:



Slika 63 Redis POST ruta

Uočavamo da smo kod Monga prvo kreirali novog klijenta preko konstruktora modela, a zatim koristili ugrađenu metodu „save()“ koja sprema novog klijenta. S druge strane, Redis koristi metodu „createAndSave()“ kojom odmah kreira i sprema novog klijenta.

Nadalje, potrebno je izmjeniti na isti način sve preostale rute. Na slici ispod imamo primjer rute za registraciju. Dakle, kao i do sada metodu „find()“ zamjenjujemo sa „search()“, a metodu „save()“ sa „createAndSave()“.

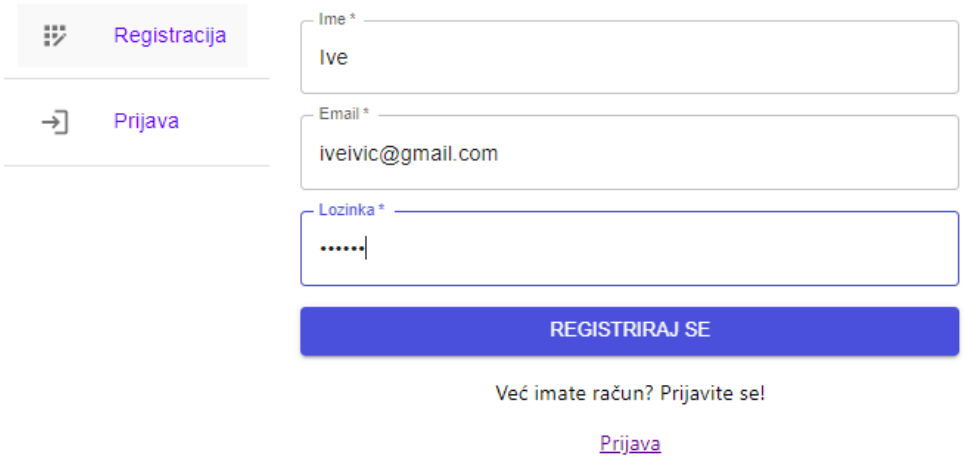


```
projekt-ndujic > backend > controllers > JS korisniciRegistracija.js > ...
const registerRouter = require("express").Router();
const Korisnik = require("../models/korisnik");
const bcrypt = require("bcrypt");
const jwtSimple = require("jwt-simple");

registerRouter.post("/", async (req, res) => {
  const sadrzaj = req.body;
  try {
    const postojeciKorisnik = await Korisnik.findOne({ email: sadrzaj.email });

    if (postojeciKorisnik)
      return res
        .status(400)
        .json({ message: "Korisnik s unesenim email-om već postoji!" });

    const hassLozinka = await bcrypt.hash(sadrzaj.lozinka, 10);

    const korisnik = new Korisnik({
      ime: sadrzaj.ime,
      email: sadrzaj.email,
      lozinka: hassLozinka,
    });

    const noviKorisnik = await korisnik.save();

    const token = jwtSimple.encode(
      { email: noviKorisnik.email },
      process.env.SECRET
    );

    res.status(200).json({ token });
  } catch (error) {
    res.status(500).json({ message: error.message || "Greška" });
  }
});

module.exports = registerRouter;
```

```
const registerRouter = require("express").Router();
const { korisnikRepozitorij } = require("../models/korisnik");
const bcrypt = require("bcrypt");
const jwtSimple = require("jwt-simple");

registerRouter.post("/", async (req, res) => {
  const sadrzaj = req.body;
  try {
    const repozitorij = await korisnikRepozitorij;
    const postojeciKorisnik = await repozitorij
      .search()
      .where("email")
      .equal(sadrzaj.email)
      .first();

    if (postojeciKorisnik)
      return res
        .status(400)
        .json({ message: "Korisnik s unesenim email-om već postoji!" });

    const hassLozinka = await bcrypt.hash(sadrzaj.lozinka, 10);

    const korisnik = await repozitorij.createAndSave({
      ime: sadrzaj.ime,
      email: sadrzaj.email,
      lozinka: hassLozinka,
    });

    const token = jwtSimple.encode(
      { email: korisnik.email },
      process.env.SECRET
    );

    res.status(200).json({ token });
  } catch (error) {
    res.status(500).json({ message: error.message || "Greška" });
  }
});

module.exports = registerRouter;
```

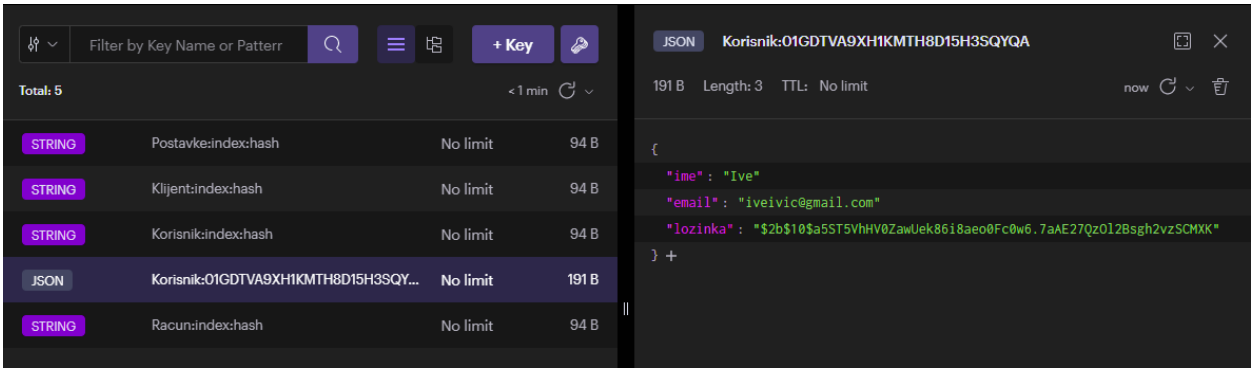
Slika 64 Primjer MongoDB i Redis rute za registraciju

Pokrećemo aplikaciju i testiramo našu aplikaciju s implementiranom Redis bazom na isti način kako je opisano u početku poglavlja. Registriramo se s novim korisnikom Ivom.



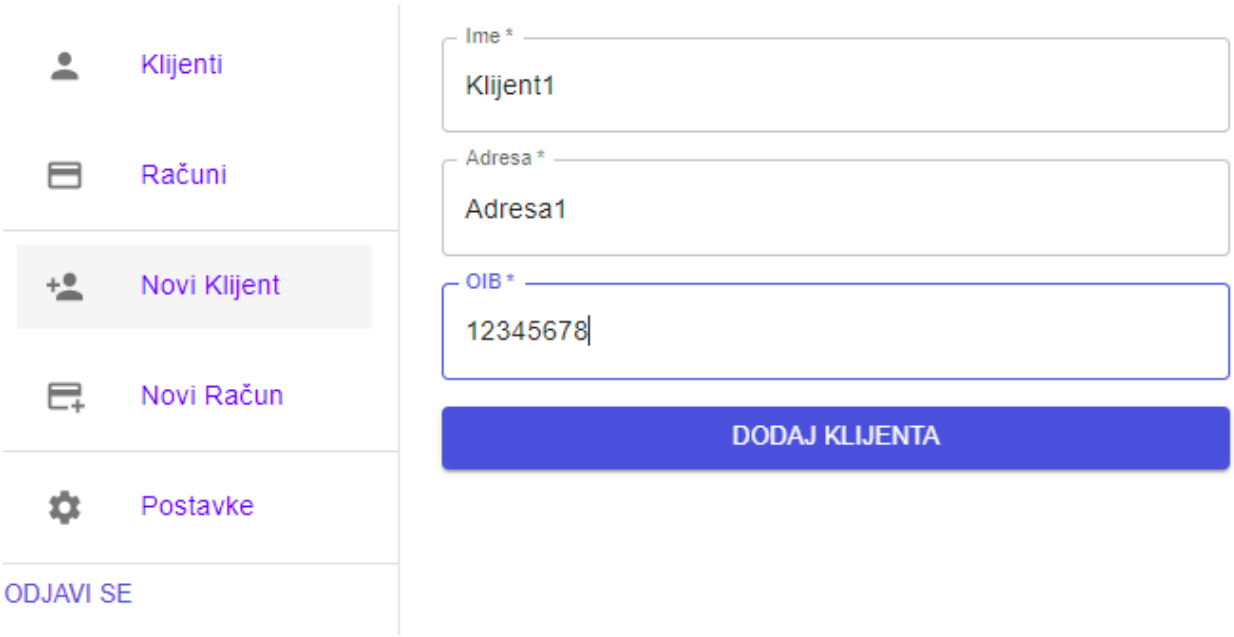
Slika 65 Registracija novog korisnika (Redis)

Nakon što smo se registrirali, otvaramo Redis Insight i uočavamo da je naš korisnik Ive uspješno registriran u aplikaciji. Možemo vidjeti da je korisnik dodan kao novi ključ, te da je vrijednost ključa u JSON formatu. RedisJSON je modul koji nam omogućuje ovu opciju, točnije on Redisu pruža JSON podršku.



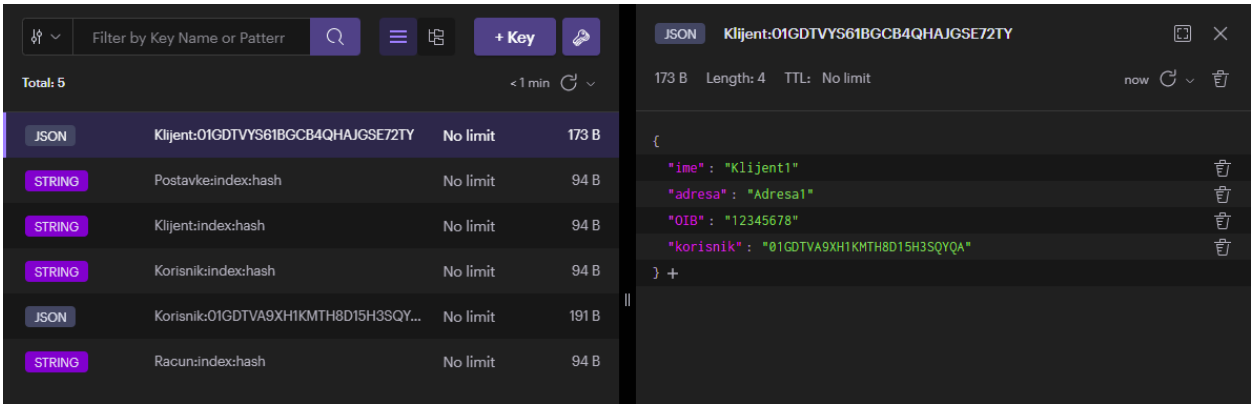
Slika 66 Prikaz registriranog korisnika koristeći Redis

Možemo još pogledati i testiranje klijenata na isti način kao što je opisano na početku poglavlja.



Slika 67 Dodavanje novih klijenata (Redis)

Možemo uočiti da se novi klijent uspješno dodao.



Slika 68 Prikaz novog klijenta koristeći Redis

Također, možemo provjeriti ovo i u samoj aplikaciji, na način da kliknemo na opciju „Klijenti“ i vidimo da se u našoj tablici pojavio dodani klijent.

Slika 69 Prikaz klijenata (Redis)

5. Zaključak

U ovom radu smo se detaljnije upoznali sa samim pojmom baze podataka. Osim toga, pobliže smo se upoznali s SQL bazama podataka, kao i NoSQL bazama podataka koje smo detaljnije obradili. Od četiri vrste NoSQL baza, odabrala sam MongoDB i Redis kako bih njihovim implementacijama u svojoj aplikaciji za izdavanje računa uočila razlike među njima.

Najočitija razlika između MongoDB-a i Redisa je njihov konceptualno različit model pohrane podataka. Redis je pohrana podataka u memoriji koja omogućuje i predmemoriju, dok MongoDB pohranjuje podatke na disk. Stoga je Redis mnogo brži jer je on baza podataka u memoriji. No, s brzinom dolaze i neki nedostatci poput korištenja više RAM-a. Budući da je Redis munjevito brz u memoriji, možemo ga koristiti u aplikacijama za obradu u stvarnom vremenu. Na primjer, može biti učinkovit u aplikacijama koje nude značajke poput upozorenja o cijenama dionica, koje vode brigu o najboljim rezultatima i analitikama u stvarnom vremenu. S druge strane, MongoDB pruža dinamičku schemu koja se sama opisuje, te olakšava dodavanje proizvoda, značajki i preporuka za kataloge kao što su e-trgovina, upravljanje imovinom i slično. Radi mogućnosti pohranjivanja bogatog sadržaja, MongoDB se koristi i u mobilnim aplikacijama i igrama.

Još jedna od bitnih razlika između ove dvije baze podataka jest ta da ne pripadaju istoj vrsti NoSQL baza. MongoDB pripada dokument orijentiranim bazama i podatci se pohranjuju u kolekcije koje sadrže dokumente. Svaki dokument se sastoji od polja ključ-vrijednost koji mogu biti tipa *string, int, double, date, bool* itd. Upravo zbog svog fleksibilnog dizajna, MongoDB se bolje skalira nego Redis, odnosno MongoDB se bolje nosi s rastućom količinom podataka. Vrijedi spomenuti i da MongoDB dopušta ograničenu veličinu od samo 16MB za dokument. S druge strane, Redis pripada ključ-vrijednost bazama i pohranjuje podatke u obliku ključa i vrijednosti kao što i sam naziv govori. Svaka vrijednost može biti tipa *string, hash, bitmap, set, list* itd. Također, modul RedisJSON pruža Redisu i JSON podršku.

Iako ove dvije baze podataka imaju mnogo razlika, dijele neke iste značajke. Na primjer, obje baze imaju licence otvorenog koda, te obje koriste sekundardno indeksiranje. Redis koristi sekundardno indeksiranje na način da se koriste sortirani skupovi za izradu tih indeksa. Mogući su i napredni indeksi, indeksi obilaska grada i kompozitni indeksi. MongoDB ih također koristi. Da ih nema, svaki dokument bi morali pretraživati unutar kolekcije što usporava vrijeme čitanja.

Za kraj, možemo zaključiti da i MongoDB i Redis imaju svoje prednosti i nedostatke. Stoga je teško zaključiti koja je od ove dvije baze bolja. Naravno, odabir baze podataka ovisi i o samom projektu, te uvjetima i zahtjevima klijenata. Ukoliko bih trebala birati između ove dvije baze podataka, opredijelila bih se za MongoDB iz razloga što se bolje nosi s rastućom količinom podataka. Odnosno, MongoDB može služiti tone korisnika, obrađivati ogromne količine podataka i podržavati stotine tisuće operacija u sekundi.

Literatura

[1] Mullins, Craig S.;, »database management system (DBMS),« 22 July 2022. [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/definition/database-management-system.

[2] Taylor, David;, »What is Data Modelling? Types (Conceptual, Logical, Physical),« Guru99, 17 August 2022. [Mrežno]. Available: https://www.guru99.com/data-modelling-conceptual-logical.html.

[3] embarcadero/IDERA, »Creating Mappings,« embarcadero/IDERA, 4 July 2017. [Mrežno]. Available: https://docwiki.embarcadero.com/ERStudioBA/190/en/Creating_Mappings.

[4] QS Study, »Different types of relationship in Database,« QS Study, [Mrežno]. Available: https://qsstudy.com/different-types-of-relationship-in-database/.

[5] Lucidchart, »What are your database diagram needs?,« Lucidchart, 2022. [Mrežno]. Available: https://www.lucidchart.com/pages/database-diagram/database-models.

[6] MongoDB, »What is NoSQL?,« MongoDB, [Mrežno]. Available: https://www.mongodb.com/nosql-explained.

[7] GeeksForGeeks, »SQL | DDL, DQL, DML, DCL and TCL Commands,« GeeksForGeeks, 30 September 2021. [Mrežno]. Available: https://www.geeksforgeeks.org/sql-ddl-dql-dml-dcl-tcl-commands/.

[8] Cloud x Lab, »NoSQL - CAP Theorem,« Cloud x Lab, [Mrežno]. Available: https://cloudxlab.com/assessment/displayslide/345/nosql-cap-theorem.

[9] HAZELCAST, »What is the CAP Theorem?,« HAZELCAST, [Mrežno]. Available: https://hazelcast.com/glossary/cap-theorem/.

[10] shubhanjaytiwari, »Key-Value Data Model in NoSQL,« GeeksForGeeks, [Mrežno]. Available: https://www.geeksforgeeks.org/key-value-data-model-in-nosql/.

[11] Sofija Simic, »NoSQL Database Types,« phoenixNAP, 10 June 2020. [Mrežno]. Available: https://phoenixnap.com/kb/nosql-database-types.

[12] redis, »Introduction to Redis,« redis, [Mrežno]. Available: https://redis.io/docs/about/.

[13] Heroku Dev Center, »Redis Enterprise Cloud,« Heroku Dev Center, 31 August 2022. [Mrežno]. Available: https://devcenter.heroku.com/articles/rediscloud.

[14] Wikipedia, »Amazon DynamoDB,« Wikipedia, 29 August 2022. [Mrežno]. Available: https://en.wikipedia.org/wiki/Amazon_DynamoDB#cite_note-ZDNet_2012-01-19-3.

[15] Dzenan Dzevlan, »Uvod u Amazon Web Servise (AWS),« 16 January 2020. [Mrežno]. Available: https://sqlheisenberg.com/uvod-u-amazon-web-servise-aws/index.html.

[16] SCYLLA, »What is DynamoDB?,« SCYLLA, [Mrežno]. Available: https://www.scylladb.com/learn/dynamodb/introduction-to-dynamodb/.

[17] Nishani Dissanayake, »DynamoDB vs Redis - The Ultimate Comparison,« Dynobase, 18 May 2022. [Mrežno]. Available: https://dynobase.dev/dynamodb-vs-redis/.

[18] Stackshare, »Amazon DynamoDB vs Redis: What are the differences?,« Stackshare, [Mrežno]. Available: https://stackshare.io/stackups/amazon-dynamodb-vs-redis.

[19] Mark Drake, »A Comparison of NoSQL Database Management Systems and Models,« Community, 9 August 2019. [Mrežno]. Available: https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models.

[20] MongoDB, »Introduction to MongoDB,« MongoDB, [Mrežno]. Available: https://www.mongodb.com/docs/manual/introduction/.

[21] TechTarget, »MongoDB,« TechTarget, [Mrežno]. Available: https://www.techtarget.com/searchdatamanagement/definition/MongoDB.

[22] D. Djermanović, »Introduction to Firebase Realtime Database,« u *Saving Data on Android*, 2021.

[23] Firebase, »Firebase,« Firebase, [Mrežno]. Available: https://firebase.google.com/products/realtime-database.

[24] TutorialsPoint, »SQL - Syntax,« TutorialsPoint, [Mrežno]. Available: https://www.tutorialspoint.com/sql/sql-syntax.htm.

[25] JavaTPoint, »SQL Languages,« JavaTPoint, [Mrežno]. Available: https://www.javatpoint.com/sql-languages.