Upotreba dokumentnih baza podataka prilikom izrade web-aplikacije

Lunko, Dominik

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:529552

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Dominik Lunko

UPOTREBA DOKUMENTNIH BAZA PODATAKA PRILIKOM IZRADE WEB-APLIKACIJE

DIPLOMSKI RAD

Varaždin, 2022.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Dominik Lunko

Matični broj: 0016129271

Studij: Organizacija poslovnih sustava

UPOTREBA DOKUMENTNIH BAZA PODATAKA PRILIKOM IZRADE WEB-APLIKACIJE

DIPLOMSKI RAD

Mentor:

dr. sc. Bogdan Okreša Đurić

Varaždin, rujan 2022.

Dominik Lunko

Izjava o izvornosti

Izjavljujem da je ovaj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

Cilj ovog rada je detaljno objasniti dokumentne baze podataka, kao dio skupa nerelacijskih (NoSQL) baza podataka, koje pokazuju rastući trend u upotrebi u informacijsko-komunikacijskoj industriji. Rad će obuhvatiti i usporedbu odabranih dokumentnih nerelacijskih baza podataka s odabranim sustavima za upravljanje relacijskim bazama podataka temeljem njihovih performansi, tipova podataka koji se koriste, pristupačnosti, jednostavnosti korištenja te sigurnosti podataka. Glavni dio rada bit će usredotočen na upotrebu odabranih dokumentnih nerelacijskih baza podataka prilikom razvoja vlastite web-aplikacije te usporedbu implementacije pomoću dviju popularnih dokumentnih baza podataka: Firebase i MongoDB.

Ključne riječi: baza podataka, nosql, nerelacijska baza podataka, web-aplikacija, dokumentna baza podataka

Sadržaj

1. Uvod

Otkako su se nerelacijske baze podataka (engl. not only SQL (NoSQL)) pojavile na tržištu 1998. godine, pružile su veliki izazov do tada poznatim relacijskim bazama podataka (engl. relational database (RDB)) [1] koje podatke pohranjuju isključivo pomoću tabličnih relacija unutar stroge sheme. RDB sadrže neke bitne značajke kao što su integritet, dosljednost, provjera valjanosti tipa i transakcijska jamstva, međutim nekim aplikacija nisu potrebne te značajke. Jedan od glavnih problema kod RDB-a su horizontalno skaliranje (preko više poslužitelja) i loša fleksibilnost sheme. Upravo zbog toga se pojavljuje potreba za korištenjem NoSQL dokumentno-orijentiranih baza podataka koje nude rješenje za prethodno navedene probleme.

Usprkos eksponencijalom rastu popularnosti NoSQL-a, njihov cilj nije bio eliminirati korištenje RDB-a. Kratica NoSQL nosi puni naziv "Not Only SQL" [1], što zapravo označava da NoSQL koriste i strukturni upitni jezik (engl. structured query language (SQL)) kako bi se unaprijedile performanse, skalabilnost i fleksibilnost shema uz integritet i dosljednost podataka. Dakle, svrha NoSQL rješenja nije zamjena relacijskih modela u cjelini, već samo u slučajevima u kojima postoji potreba za skalabilnošću i upravljanjem velikom količinom podataka. NoSQL se najviše koriste za aplikacije u realnom vremenu i aplikacije sa velikim skupovima podataka ("Big data") [1].

Nadalje, u radu se detaljnije objašnjava kako i zašto su dokumentno-orijentirane baze podataka, kao dio NoSQL-a, doživijele toliki uspijeh u današnjici te će se napraviti detaljna usporedba dokumentno-orijentirane baze podataka MongoDB sa sustavom za upravljanje relacijskim bazama podataka (engl. relational database management system (RDBMS)) MySQL na temelju njihovih performansi, sigurnosti, skalabilnosti, itd. Rad je potkrijepljen primjerima te implementacijom aplikacije uz pomoć dokumentnih baza podataka MongoDB i Firebase.

2. Metode i tehnike rada

Prilikom izrade rada korištene su knjige te stručni i znanstveni članci koji su navedeni u popisu literature. Također, prilikom opisivanja primjera i izrade aplikacije, korištene su MongoDB Atlas, Mongoose, Angular, Firebase Firestore i MySQL službene dokumetacije. Osim NoSQL dokumentno-orijentiranih baza podataka MongoDB i Firestore, korišteni su Mongoose, Rapid API, JavaScript razvojno okruženje Angular za izradu sučelja (klijent strane) aplikacije te NodeJS i Express za izradu poslužitelja aplikacije.

2.1. MongoDB Atlas

MongoDB Atlas [2] je baza podataka u oblaku, poznatija i kao baza podataka kao usluga (engl. Database-as-a-Service (DBaaS)). MongoDB Atlas je vrlo jednostavan za korištenje, što znači da korisnici nemoraju voditi brigu o lokalnom fizičkom sklopovlju (engl. hardware), programskim ažuriranjima te potrebnoj konfiguraciji. Također je vrlo siguran za korištenje jer sadrži sofisticirane sigurnosne kontrole za privatnost podataka, a dostupan je u više od 80 regija na "AWS"-u, "Google Cloud"-u, i "Azure"-u [2].

2.2. Firebase Firestore

Cloud Firestore [3] je NoSQL dokumentno-orijentirana baza podataka u oblaku koja pruža programsko sučelje za dohvaćanje podataka na temelju referenci, a koristi se za mobilne, web internet stvari (engl. Internet of Things (IOT)) aplikacije na globalnoj razini. Prilikom pisanja u Firestore, kreira se referenca dokumenta (engl. Document reference) koja sadrži lokaciju dokumenta u Firestore bazi podataka [4] i identifikator dokumenta (id) unutar kolekcije. Referenca dokumenta može se definirati i kao opis dokumenta. Zbog takvog način zapisivanja u bazu podataka, dokumenti u Firestore-u ne trebaju imati dodatno polje s jedinstvenom vrijednošću (kao što je polje `_id` kod MongoDB-a).

2.3. Angular

AngularJS je JavaScript razvojni okvir otvorenog koda razvijen od strane Google-a [5]. Služi za izgradnju JavaScript skalabilnih, klientskih web-aplikacija temeljenih na jednoj stranici (engl. Single page apps (SPA)) pomoću prezentacijskog jezika za izradu web stranica (engl. HyperText Markup Language (HTML)) i TypeScript-a [5]. TypeScript je nadskup JavaScript programskog jezika koji omogućuje i provjeru statičkog tipa podataka.

2.4. NodeJS

NodeJS [6] je okruženje za izvršavanje JavaScript programskog koda na više različitih platformi (engl. Cross-platform), otvorenog je koda, a pokreće ga Google Chrome-ov V8 JavaScript mehanizam. NodeJS se najčešće koristi za izradu poslužitelja web-aplikacija [6] te kao takav predstavlja poveznicu između sučelja (klijent strane) i baze podataka.

2.4.1. Node package manager

Node package manager (NPM) [7] je upravitelj paketa za JavaScript programski jezik, a osim toga može se definirati i kao repozitorij za objavljivanje projekata otvorenog koda. Putem sučelja naredbenog retka (engl. command-line interface (CLI)) uspostavlja se interakcija s repozitorijem na kojem se nalaze paketi ili biblioteke koje je potrebno preuzeti i instalirati tokom razvoja aplikacije [7]. Na slici 1 prikazano je korištenje naredbe `npm install`.

```
PS C:\Users\domin\Desktop\5.GODINA\DIPLOMSKI_RAD> npm install mongoose
npm WARN saveError ENOENT: no such file or directory, open 'C:\Users\domin\Desktop\5.GODINA\DIPLOMSKI_RAD\package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open 'C:\Users\domin\Desktop\5.GODINA\DIPLOMSKI_RAD\package.json'
npm WARN DIPLOMSKI_RAD No description
npm WARN DIPLOMSKI_RAD No repository field.
npm WARN DIPLOMSKI_RAD No README data
npm WARN DIPLOMSKI_RAD No license field.

+ mongoose@6.5.2
added 28 packages from 68 contributors and audited 28 packages in 3.396s

4 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Slika 1: Primjer korištenja naredbe "npm install"

2.4.2. ExpressJS

ExpressJS [8] je JavaScript razvojni okvir koji pruža aplikacijsko programsko sučelje (engl. Application Programming Interfaces (API)) za izradu web-aplikacija i poslužitelja. Vrlo je fleksibilan za korištenje budući da je velik broj modula s NPM repozitorija moguće izravno dodati u ExpressJS aplikaciju [8]. Osim fleksibilnosti, vrlo je jednostavan za korištenje, a kreiranje ExpressJS aplikacije 1 postiže se pozivom metode `express()`;

Isječak kôda 1: Kreiranje ExpressJS aplikacije

```
1 import express from 'express';
2
3 const app = express();
```

2.5. Mongoose

Prilikom implementacije s MongoDB-om korištena je JavaScript objektno-orijentirana programska biblioteka Mongoose [9] koja služi za povezivanje MongoDB-a i Express okvira web-aplikacije. Mongoose uvelike olakšava rad sa MongoDB-om zato što sadrži ugrađeno pretvaranje tipova i provjeru ispravnosti podataka. Osim toga Mongoose model pruža sučelje za postavljanje upita, odnosno za čitanje, kreiranje, ažuriranje i brisanje zapisa iz baze podataka. Za kreiranje modela s Mongoose-om 2, potrebno je kreirati shemu kojom se definiraju struktura dokumenta, zadane vrijednosti, validatori, itd.

Isječak kôda 2: Kreiranje modela s Mongoose-om

```
1
2  import mongoose from "mongoose";
3
4  const userSchema = new mongoose.Schema({
5      name: {type: String, required: true},
6      email: {type: String, required: true, unique: true},
7      password: {type: String, required: true},
8      age: {type: Number},
9      weight: {type: Number},
10     height: {type: Number},
11     gender: {type: String},
12     activity_level: { type: String },
13 });
14
15 const User = mongoose.model('User', userSchema);
16
17 export default User;
```

2.6. Kaggle

Kaggle je javna podatkovna platforma koja omogućuje korisnicima objavljivanje, pretraživanje i preuzimanje skupova podataka različitih kategorija [10]. Platformu najčešće koriste podatkovni znanstvenici (engl. Data scientists) te inžinjeri strojnog učenja (engl. machine learning engineers), a vrlo je prisutna upotreba i u akademske svrhe. Kaggle je osnovao Nicholas Gruen 2010. godine, a danas broji preko 8 milijuna registriranih korisnika [10].

3. Dokumentno-orijentirane baze podataka

Dokumentno-orijentirana baza podataka jedna je od vrsta NoSQL baza podataka uz stupičaste, grafovske i ključ-vrijednost baze podataka [11]. Primarno su nastale kao podklasa ključ-vrijednost baza podataka, no za razliku od ključ-vrijednost baza podataka, dokumentno-orijentirane baze podataka podržavaju sekundarne indekse. Dokumentne baze podataka pohranjuju dinamičke podatke u nestrukturiranim i polustrukturiranim formatima [1], a dopuštaju pohranu ugniježđenih parova ključ-vrijednost. Pohrana dokumenata obično podržava formate kao što su [11]: XML (eXtensible Markup Language), YAML (Yet Another Markup Language) i JSON (JavaScript Object Notation).

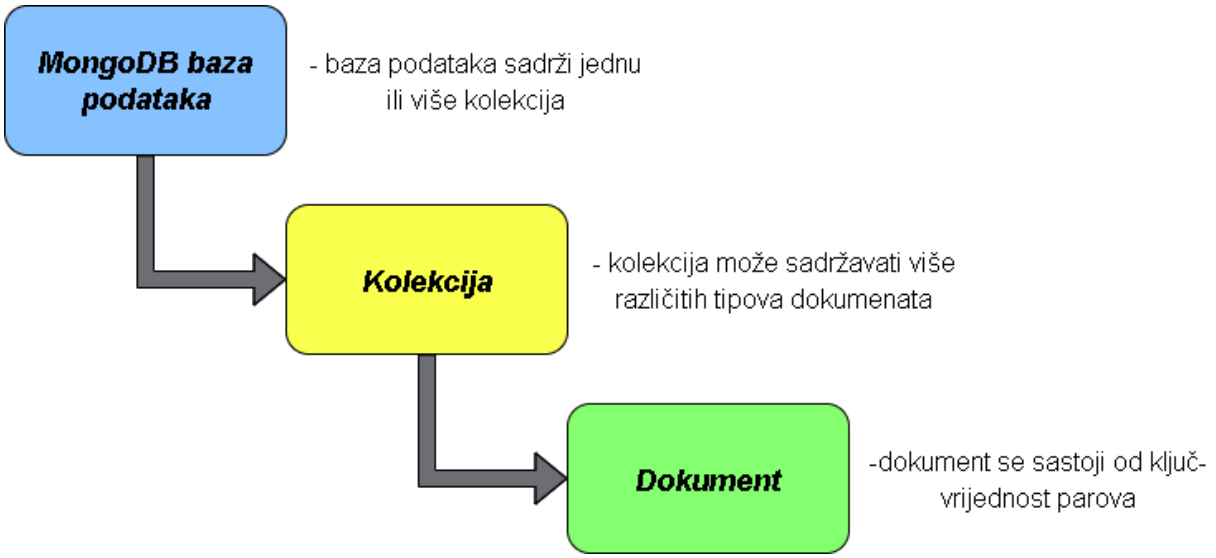
Dokumentno-orijentirane baze podataka imaju mogućnost pohrane svih podataka jednog objekta unutar jednog dokumenta, dok se prilikom korištenja RDB-a podaci jednog entiteta moraju raspodijeliti unutar više tablica ili baza podataka. Na primjer, u aplikaciji za e-trgovinu, različiti proizvodi imaju različit broj atributa koji se mogu opisati u jednom zasebnom dokumentu, što omogućava jednostavnije upravljanje i brže čitanje podataka, a promjene polja (ključa i vrijednosti) jednog dokumenta neće utjecati na druge dokumente unutar kolekcije baze podataka kojoj dokument pripada. S druge strane, upravljanje tisućama atributa u relacijskim bazama podataka je neučinkovito, a svaki zapis unutar tablice mora imati isti broj polja. Dakle, dodavanje novog polja jednom dokumentu, ne zahtijeva dodavanje tog istog polja drugim dokumentima u kolekciji. Kolecije ili zbirke dokumentnih baza podataka opisane su u sljedećoj sekciji 3.1.

Baze podataka orijentirane na dokumente posljednjih su godina doživjele ogroman rast popularnosti [11] jer su izvrsne za čuvanje velikih količina nepovezanih, složenih informacija koje se razlikuju po strukturi. Neke od najpoznatijih dokumentnih baza podataka su MongoDB, Firebase, CouchDB i RavenDB [11]. MongoDB je najpopularnija dokumentno-orijentirana baza podataka [11] i upravo ona je detaljno objašnjena u daljnjem radu.

3.1. Struktura

Dokumentno-orijentirane baze podataka usmjerene su na metode pohranjivanja i pristupa optimizirane za dokumente, za razliku od redaka ili zapisa u RDBMS-u. Struktura dokumentnih baza podataka sastoji se od tri vrste komponenti [12], a to su baza podataka, zbirka ili kolekcija i dokument. Baza podataka nalazi se na vrhu hijerarhije, kolekcije na sljedećoj razini, a dokumenti na dnu. Poslužitelj ima mogućnost pohranjivanja više baza podataka, a svaka baza podataka može sadržavati jednu ili više kolekcija unutar kojih se nalazi nula ili više dokumenata. Baza podataka može sadržavati više kolekcija, ali jedna kolekcija ne može biti dijelom više baza podataka. Isto tako, kolekcija može sadržavati više dokumenata, ali dokument ne može biti dijelom više kolekcija. Kolekcije u dokumentno-orijentiranim bazama podataka odgovaraju tablicama u relacijskim bazama podataka. Svaka kolekcija sadrži dokumente koji se mogu ugnijezditi u složene hijerarhije koje podržavaju učinkovite implementacije upita i indeksa. Dokument se pak sastoji od naziva polja i vrijednosti, a polja su analogna stupcima u

relacijskim bazama podataka. Dokumenti u jednoj kolekciji vjerojatno će imati sličnu strukturu u praksi, no sam sustav baze podataka to ne nameće i radit će stabilno i brzo bez obzira na to kako podaci izgledaju. Kao što je već prethodno navedeno, unutar kolekcija nalazi se nula ili više dokumenata koji odgovaraju zapisima u relacijskim bazama podataka. Slika 2 prikazuje odnos između baze podataka, kolekcija i dokumenata u dokumentno-orijentiranoj bazi podataka MongoDB.



Slika 2: Struktura dokumentne baze podataka MonogoDB; izvor: [12]

3.1.1. Dokument

Dokument je zapis u dokumentno-orijentiranoj bazi dokumenata [11] koji pohranjuje podatke u parovima ključ-vrijednost. Vrijednosti koje se pohranjuju unutar dokumenta mogu biti različite vrste i strukture kao što su datumi, brojevi, objekti, nizovi, itd. Jedan dokument u bazi obično pohranjuje informacije o jednom objektu i svim njegovim povezanim metapodacima. Dokumenti unutar kolekcije su neovisni jedni o drugima i ne postoji odnos vanjskog ključa o kojem treba brinuti prilikom izvršavanja upita. Jednostavna izrada dokumenata omogućuje jednokratno stvaranja složenih objekata i minimalno održavanje nakon što se dokument stvori. Samim time dokumentno-orijentirane baze podataka imaju sposobnost pohranjivanja i obrade velikih skupova podataka [1].

Dokument pohranjen u NoSQL bazi podataka može se gotovo izravno mapirati na strukturu klase programskih jezika jer je pohranjen u formatu kojeg koriste programski jezici kao što su [1] JSON ili XML. Dakle, korištenje dokumentno-orijentiranih baza podataka itekako utječe na jednostavnost koda te na uštedu vremena programera. Osim toga, ukoliko se model podataka treba promijeniti, potrebno je ažurirati samo određene dokumente jer ne postoji shema te tako ne dolazi do "zastoja" u bazi podataka kako bi se izvršile određene promjene.

3.1.1.1. JSON

JavaScript Object Notation ili JSON [13] je standardni format za prikaz struktura podataka koji se bazira na JavaScript objektnoj sintaksi. Potpuno je neovisan o jeziku, ali koristi konvencije koje su poznate programerima iz porijekla jezika C, uključujući C, C++, C Sharp, Java, JavaScript, Perl, Python i mnoge druge. JSON format je vrlo jednostavan za čitanje i pisanje te je zato prisutna njegova česta upotreba. Najviše se koristi unutar web-aplikacija za razmjenu podataka između servera ili poslužitelja (engl. backend) i klijenta (engl. frontend) [13]. JSON format je izgrađen na dvije strukture, a to su [13]: zbirka parova ime(ključ)/vrijednost i uređena lista vrijednost. Zbirka parova ime/vrijednost se u raznim jezicima ostvaruje kao objekt, zapis, struktura, rječnik, hash tablica, popis s ključevima ili asocijativni niz, dok se uređena lista vrijednosti u većini jezika ostvaruje kao vektor, lista ili niz. Primjer JSON objekta prikazan je u isječku koda 3.

Isječak kôda 3: Primjer JSON objekta

```
{
     id: "a1233_dsd2",
     glumci: [{
     ime: "Zoe",
        prezime: "Saldana",
        uloga: "Neytiri"
    },
    {
        ime: "Sam",
        prezime: "Worthington",
        uloga: "Jake Sully"
    },
    {
        ime: "Michelle",
        prezime: "Rodriguez",
        uloga: "Trudy Chacon"
    }],
    naslov: "Avatar",
    godina: "2010",
    filmski_redatelj: {
        ime: "James",
        prezime: "Cameron",
        nagrade : ["Academy Awards" , "ACE Eddie Awards" , "BAFTA Awards" ],
    },
    zanr: "SF",
    vrijeme_trajanja: 110
}
```

3.1.1.2. XML

Extensible Markup Language ili XML je označni jezik za opisivanje, pohranjivanje i prijenos podataka, a temelji se na standardnom generaliziranom prezentacijskom jeziku (engl. Standard Generalized Markup Language (SGML)) [14]. XML dokumenti pohranjuju se kao datoteke američkog standardnog koda za razmjenu informacija (engl. American Standard Code for Information Interchange (ASCII)) i mogu se uređivati pomoću bilo kojeg uređivača teksta.

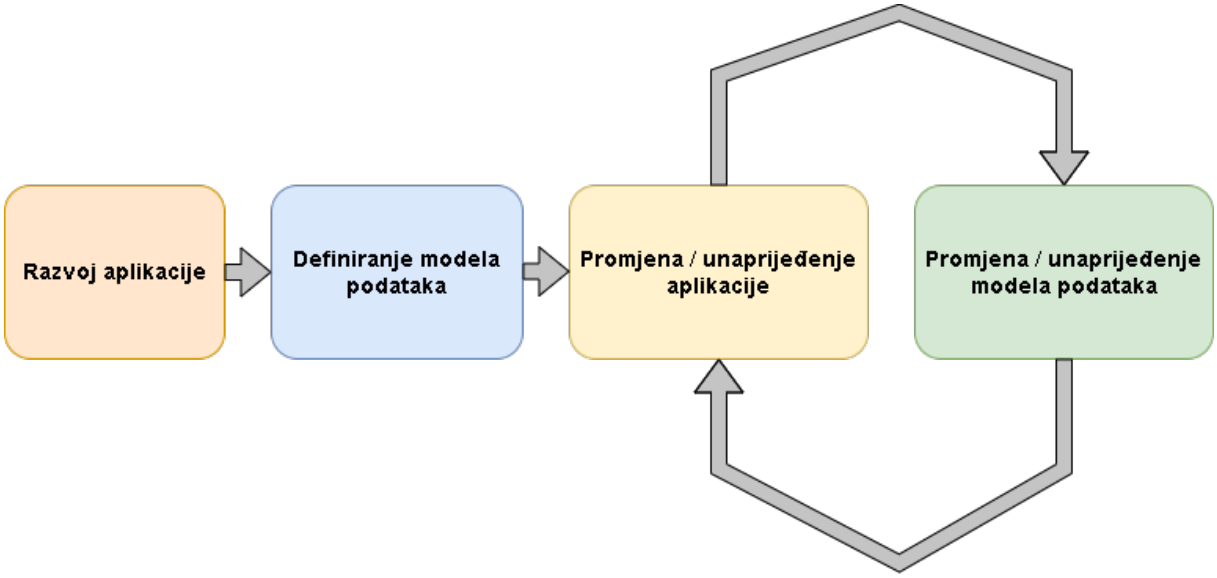
XML omogućuje komunikaciju strukturiranih informacija između više programa, programa i ljudi, lokalno i putem mreže [14]. XML podaci se mogu koristiti za stvaranje različitih tipova sadržaja uključujući web, ispis i mobilni sadržaj, a prema World Wide Web Consortiumu (W3C) [14], standardnom tijelu za web, XML se koristi za oblikovanje dokumenata, pisanje tehničke dokumentacije, mogućnost konfiguracije za aplikacijske softvere i za prijenos podataka . Isječak koda 4 prikazuje primjer XML-a.

Isječak kôda 4: Primjer XML-a

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <id>a1233_dsd2</id>
  <glumci>
    <ime>Zoe</ime>
    <prezime>Saldana</prezime>
    <uloga>Neytiri</uloga>
  </glumci>
  <glumci>
    <ime>Sam</ime>
    <prezime>Worthington</prezime>
    <uloga>Jake Sully</uloga>
  </glumci>
  <glumci>
    <ime>Michelle</ime>
    <prezime>Rodriguez</prezime>
    <uloga>Trudy Chacon</uloga>
  </glumci>
  <naslov>Avatar</naslov>
  <godina>2010</godina>
  <filmski_redatelj>
    <ime>James</ime>
    <prezime>Cameron</prezime>
    <nagrade>Academy Awards</nagrade>
    <nagrade>ACE Eddie Awards</nagrade>
    <nagrade>BAFTA Awards</nagrade>
  </filmski_redatelj>
  <zanr>SF</zanr>
  <vrijeme_trajanja>110</vrijeme_trajanja>
</root>
```

3.2. Modeliranje podataka

Važan korak u implementaciji baze podataka je modeliranje podataka, jer olakšava razumijevanje projekta kroz ključne značajke koje mogu spriječiti pogreške u programiranju i radu. Proces modeliranja [15] podataka uključuje stvaranje vizualnog prikaza koji predstavlja značenje podataka i odnos između podataka. Ključni izazov u modeliranju podataka je balansiranje potreba aplikacije na način da aplikacija primi sve potrebne informacije, a da pritom ne dolazi do nepotrebnog repliciranja podataka u bazi podataka. Prilikom dizajniranja podatkovnih modela, uvijek je potrebno uzeti u obzir primjenu podataka u aplikaciji (tj. upite, ažuriranja i obradu podataka), kao i strukturu samih podataka [16]. Kod dokumentno-orijentiranih baza podataka proces modeliranja podataka započinje razvojem aplikacije [17] na temelju koje se potom definiraju modeli podataka. Nakon definiranja modela podataka, aplikacija se naknadno može unaprijeđivati, a na temelju promjena u aplikaciji se unaprijeđuju i modeli podataka unutar baze. Taj proces se može ponavljati više puta i upravo to čini dokumentno-orijentirane baze podataka vrlo fleksibilnima [17]. Također, potrebno je uzeti u obzir i različite operativne čimbenike koji utječu na performanse. Na primjer, različiti modeli podataka mogu omogućiti učinkovitije upite te povećati propusnost operacija umetanja i ažuriranja, što će detaljnije biti razjašnjeno u podsekcijama 3.2.1 i 3.2.2.



Slika 3: Proces modeliranja podataka u MongoDB-u; izvor: [17]

Proces modeliranja podataka puno je jednostavniji s dokumentno-orijentiranim bazama podataka nego s relacijskim bazama podataka zato što su dokumentno-orijentirane baze podataka vrlo fleksibilne i lako se prilagođavaju zahtjevima aplikacije u bilo kojem trenutku. s druge strane, prilikom modeliranja podataka u relacijskim bazama podataka bitno je vrlo dobro razumijeti i analizirati tehničke i funkcionalne poslovne zahtjeve u samom početku procesa modeliranja jer relacijske baze podataka nisu fleksibilne te naknadno mijenjanje sheme zahtjeva puno posla. Dva osnovna tipa modeliranja podataka su [15]: denormalizirani ili ugnježđeni i normalizirani ili referentni model, koji su opisani u nastavku rada.

3.2.1. Ugniježđeni ili denormalizirani model podataka

Ugniježđeni dokument [18] je dokument koji se nalazi unutar drugog dokumenta, a podaci koje pohranjuje su povezani s određenim poljem u roditeljskom dokumentu. Takav ugniježđeni podatkovni model koji pohranjuje sve povazane podatke u jednom dokumentu naziva se još i "denormalizirani" model podataka. Isječak koda 5 prikazuje primjer ugnjiežđenog dokumenta. Ugnjiežđeni dokumenti omogućavaju bolje performanse za operacije čitanja, kao i mogućnost dohvaćanja i ažuriranja povezanih podataka u jednoj operaciji baze podataka. Potencijalni problem s ugniježđenim dokumentima je taj što prekomjerno ugnježđivanje dokumenata može dovesti do velike hijerarhije unutar jednog dokumenta u kojem se nalaze podaci koji aplikaciji nisu potrebni, što uzorkuje dodatno opterećenje poslužitelja i usporava operacije čitanja. Ugniježđeni podatkovni modeli uobičajeno se koriste kada [18]:

- Postoji odnos kardinalnosti jedan prema jedan između entiteta;
- Postoji odnos kardinalnosti jedan prema više između entiteta, a pri tome se podređeni dokumenti uvijek pojavljuju s ili se gledaju u kontekstu nadređenog dokumenta;

Isječak kôda 5: Primjer ugnjiežđenog dokumenta

```
// dokument filmski redatelj
{
    _id: "jamescameron"
    ime: "James",
    prezime: "Cameron",
    nagrade : ["Academy Awards" , "ACE Eddie Awards" , "BAFTA Awards" ],
    filmovi: [{
        naslov: "Avatar",
        godina: "2010",
        zanr: "SF",
        vrijeme_trajanja: 110
    }],
    adresa: {
         drzava: Kanada,
         grad: "Kapuskasing",
         ulica: "Anderson Street"
    }
}
```

3.2.2. Referentni ili normalizirani modeli podataka

Normalizirani ili referentni podatkovni modeli [18] opisuju odnose pomoću referenci između dokumenata. Na primjeru 5 prikazan je ugnježđeni dokument gdje se podaci o filmu kojeg je redatelj režirao i adresi redatelja nalaze u istom dokumentu kao i osnovni podaci o redatelju (ime, prezime, itd.). Korištenjem referentnog podatkovnog modela, prethodni primjer 5 se može razdvojiti u 3 različita modela na način da osnovni podaci o filmskom redatelju budu jedan model, podaci o filmovima drugi i podaci o adresi treći model, kao što se vidi na primjeru 6. Referentni podatkovni modeli najčešće se koriste [18]:

- Kada ugnježđivanje dokumenata rezultira dupliciranjem podataka, ali ne pruža dovoljne prednosti u izvedbama čitanja;
- Za predstavljanje složenijih odnosa kardinalnosti više na više;
- Za modeliranje velikih hijerarhijskih skupova podataka;

Isječak kôda 6: Primjer referentnog dokumenta

```
// dokument filmski redatelj
{
    _id: "jamescameron"
    ime: "James",
    prezime: "Cameron",
    nagrade : ["Academy Awards" , "ACE Eddie Awards" , "BAFTA Awards" ]
}
// dokument film
{
    _id: "avatar_id",
    filmski_redatelj_id: "jamescameron",
    naslov: "Avatar",
    godina: "2010",
    zanr: "SF",
    vrijeme_trajanja: 110
}
// dokument adresa
{
    _id: "adresa_id",
    filmski_redatelj_id: "jamescameron",
    drzava: Kanada,
    grad: "Kapuskasing",
    ulica: "Anderson Street"

}
```

3.2.3. Primjeri i usporedba ugnježđenog i referentnog modela podataka

Ukoliko je potrebno smjestiti jedan ili nekoliko jednostavnijih podatkovnih entiteta u kontekstu drugoga, u odnosu kardinalnosti jedan prema jedan, ugnježđivanje dokumenata je efikasnije rješenje u odnosu na referenciranje [18]. Primjer 5 prikazuje bolji način modeliranja podataka od primjera 6 zato što u takvom slučaju ne dolazi do dupliciranja podataka (jedan film veže uz jednog filmskog redatelja), a podređeni dokumenti "film" i "adresa" se uvijek gledaju u kontekstu nadređenog dokumenta "filmski redatelj".

Ponekad ugnježđivanje dokumenata može dovesti do dupliciranja podataka kao što prikazuje primjer 7, stoga je u takvim slučajevima potrebno koristiti referentne modele podataka kao što prikazuje primjer 8. Dupliciranje podataka riješeno je na način da se u dokumentu "pjevač" kreira polje "pjesme" koje predstavlja listu jedinstvenih ključeva pjesama pjevača.

Isječak kôda 7: Primjer ugnježđenog modela podataka (jedan na više)

```
1
2  // dokumenti pjesma
3  {
4     naziv_pjesme: "Poljubi zemlju",
5     godina_objave: 1987,
6     duljina_trajanja: "4:25",
7     zanr: ["Pop"],
8     album: "The Platinum Collection",
9     autor: {
10                ime_i_prezime: "MATE MIŠO KOVAČ" ,
11                datum_rodenja: ISODate("1941-07-16"),
12                mjesto_rodenja: "Šibenik",
13                drzavljanstvo: "Hrvat"
14             }
15 }
16 {
17    naziv_pjesme: "Svi Pjevaju, Ja Ne Čujem" ,
18    godina_objave: 1987,
19    duljina_trajanja: "4:08",
20    zanr: ["Pop", "Folk", "Narodna"],
21    album: "The Platinum Collection" ,
22    autor: {
23                ime_i_prezime: "MATE MIŠO KOVAČ" ,
24                datum_rodenja: ISODate("1941-07-16"),
25                mjesto_rodenja: "Šibenik" ,
26                drzavljanstvo: "Hrvat"
27             }
28 }
```

Isječak kôda 8: Primjer pohranjivanja reference unutar dokumenta pjevač

```
1
2  // dokument pjevač
3  {
4     ime_i_prezime: "MATE MIŠO KOVAČ" ,
5     datum_rodenja: ISODate("1941-07-16"),
6     mjesto_rodenja: "Šibenik" ,
7     drzavljanstvo: "Hrvat",
8     pjesme: ["poljubiZemlju", "sviPjevaju", ...]
9  }
10
11 // dokumenti pjesma
12 {
13     _id: "poljubiZemlju",
14     naziv_pjesme: "Poljubi zemlju",
15    godina_objave: 1987,
16    duljina_trajanja: "4:25",
17    zanr: ["Pop"],
18    album: "The Platinum Collection"
19 }
20 {
21    _id: "sviPjevaju",
22    naziv_pjesme: "Svi Pjevaju, Ja Ne Čujem" ,
23    godina_objave: 1987,
24    duljina_trajanja: "4:08",
25    zanr: ["Pop", "Folk", "Nardna"],
26    album: "The Platinum Collection"
27 }
```

Prilikom korištenja reference, rast odnosa određuje gdje pohraniti referencu. Ako je u ovom slučaju broj pjesama po pjevaču mali s ograničenim rastom, pohranjivanje reference dokumenta "pjesma" unutar polja u dokumentu "pjevač" ponekad može biti korisno. U suprotnom, ako je broj pjesma po pjevaču neograničen (npr. pjevač ima 1000 pjesama), ovaj bi model podataka doveo do promjenjivih, rastućih nizova, stoga bi bilo korisnije pohranjivati referencu na dokument "pjevač" unutar dokumenta "pjesma" kao što prikazuje primjer 9.

Isječak kôda 9: Primjer pohranjivanja reference unutar dokumenta pjesma

```
1  // dokument pjevač
2  {
3     _id: "mmkovac",
4     ime_i_prezime: "MATE MIŠO KOVAČ" ,
5     datum_rodenja: ISODate("1941-07-16"),
6     mjesto_rodenja: "Šibenik" ,
7     drzavljanstvo: "Hrvat"
8  }
9
10
11
12 // dokumenti pjesma
13 {
14    _id: "poljubiZemlju",
```

```
    naziv_pjesme: "Poljubi zemlju",
    godina_objave: 1987,
    duljina_trajanja: "4:25",
    zanr: ["Pop"],
    album: "The Platinum Collection" ,
    pjevac_id: "mmkovac"
}
{
    _id: "sviPjevaju",
    naziv_pjesme: "Svi Pjevaju, Ja Ne Čujem" ,
    godina_objave: 1987,
    duljina_trajanja: "4:08",
    zanr: ["Pop", "Folk", "Nardna"],
    album: "The Platinum Collection" ,
    pjevac_id: "mmkovac"
}
```

3.3. Indeksiranje

Indeks [19] je podatkovna struktura koja omogućava učinkovito i brzo pretraživanje podataka bez potrebe za pretraživanjem svakog retka u tablici baze podataka prilikom korištenja RDBMS-a, ili pretraživanja svakog dokumenta unutar kolekcije prilikom korištenja dokumentno-orijentiranih baza podataka. Dakle, indeks je sastavni dio svakog sustava za upravljanje bazama podataka jer omogućuje brže izvršavanje upita. Dokumentno-orijentirane baze podataka podržavaju indekse na bilo kojem polju ili potpolju dokumenta u kolekciji. Indeks nad potpoljem dokumenta naziva se još i sekundarni indeks [20]. U sljedećem dijelu prikazani su primjeri kreiranja indeksa unutar dokumentno-orijentirane baze podataka MongoDB.

Prilikom kreiranja kolekcije, MongoDB automatski kreira jedinstveni indeks u polju `_id`. Za kreiranje vlastitih indeksa u MongoDB-u koristi se metoda createIndex() unutar koje je potrebno proslijediti jedno ili više polja ili potpolja dokumenta zajedno sa smjerom ključa u indeksu (1 ili -1) gdje vrijednost 1 određuje sortiranje uzlaznim redoslijedom, a vrijednost -1 određuje sortiranje indeksa silaznim redoslijedom. Indeks kreiran nad samo jednim poljem dokumenta ili samo jednim poljem ugniježđenog dokumenta (potpolje) 10, naziva se jednostavni indeks (engl. Single field index) [21].

Isječak kôda 10: Primjer kreiranja jednostavnog indeksa s MongoDB-om

```

{
  _id: "ppero",
  ime: "Petar",
  nadimak: "Pero",
  prezime: "Perić" ,
  adresa: { zupanija: "Zagrebacka", grad: "Zapresic" }
  godine: 25
}

// Kreiranje jednostavnog indeksa nad poljem dokumenta
```

```
db.korisnici.createIndex( { ime: 1 } )

// Kreiranje jednostavnog indeksa nad poljem ugniježđenog dokumenta
db.zapisi.createIndex( { "adresa.zupanija": 1 } )
```

Indeks koji sadrži referencu na više polja unutar dokumenta kolekcije naziva se složeni indeks (engl. Compound index) [22], a takav indeks podržava upite koji se podudaraju u više polja. Redoslijed polja navedenih u složenom indeksu vrlo je važan. Složeni indeks će sadržavati reference na dokumente sortirane prema prvom navedenom polju, a zatim unutar svake vrijednosti prvog navedenog polja, reference će biti poredane prema drugom navedenom polju. Na primjeru 11 reference na dokumente će prvo biti sortirane uzlazno prema polju "ime", a zatim unutar svake vrijednosti polja "ime", biti će sortirane silazno prema vrijednostima polja "godine".

Ukoliko korisnik ne postavi vlastiti naziv indeksa, MongoDB stvara zadani naziv za indeks tako što ulančava indeksirane ključeve i smjer svakog ključa u indeksu (tj. 1 ili -1) uz korištenje podvlaka kao razdjelnika. Na primjer, indeks kreiran nad poljima "ime": 1 i "godine": -1 imat će zadani naziv `"ime_1_godine_-1"`. MongoDB omogućuje korisnicima stvaranje indeksa s prilagođenim nazivom koji je čitljiv i razumljiv [19], kao što prikazuje primjer 11.

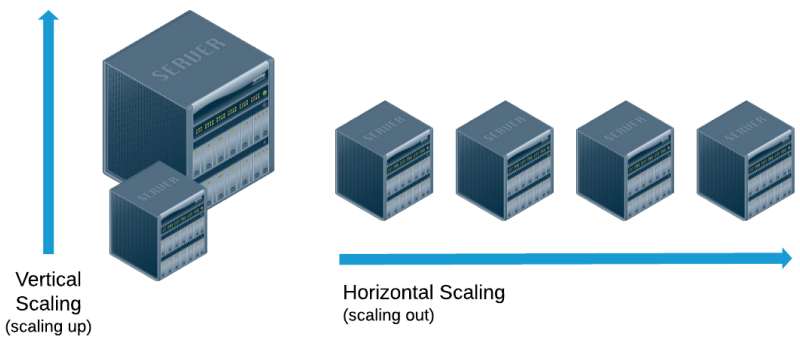
Isječak kôda 11: Primjer kreiranja složenog indeksa s prilagođenim nazivom

```
 {
  _id: "ppero",
  ime: "Petar",
  nadimak: "Pero",
  prezime: "Perić" ,
  godine: 25
}

// Kreiranje složenog indeksa sa zadanim nazivom
db.korisnici.createIndex(
  { ime: 1, godine: -1 }
)

// Kreiranje složenog indeksa s prilagođenim nazivom
db.korisnici.createIndex(
  { ime: 1, godine: -1 },
  { name: "upit za korisnika"  }
)
```

3.4. Skalabilnost

Skalabilnost baze podataka [23] određuje broj zahtjeva za čitanje i pisanje koje baza podataka može učinkovito podržati. Sustavi baza podataka s velikim skupovima podataka ili aplikacijama visoke propusnosti mogu dovesti u pitanje kapacitet svog poslužitelja jer velik broj zahtjevnih upita može iscrpiti kapacitet procesora (Central processing unit (CPU)) ili radnu memoriju (engl. Random Access Memory (RAM)) poslužitelja pa je u tom slučaju potrebno skalirati tj. povećati kapacitet baze podataka. Postoje dvije osnovne metode za skaliranje baze podataka, a to su vertikalno i horizontalno skaliranje [24].

Vertikalno skaliranje [24] uključuje povećanje kapaciteta jednog poslužitelja, kao što je korištenje jačeg CPU-a, dodavanje više RAM-a ili povećanje količine prostora za pohranu. Dokumentno-orijentirane baze podataka su horizontalno skalabilne, što znači da mogu podnijeti povećani promet jednostavnim dodavanjem više poslužitelja bazi podataka. Horizontalno skaliranje [23] postiže se metodama fragmentiranja (engl. Shardnig) i replikacije (engl. Replication) koje su opisane u daljnjem radu. NoSQL baze podataka imaju mogućnost postati veće i mnogo snažnije od relacijskih baza podataka, što ih čini preferiranim izborom za velike skupove podataka koji se stalno razvijaju.



Slika 4: Prikaz vertikalnog i horizontalnog skaliranja; izvor: [24]

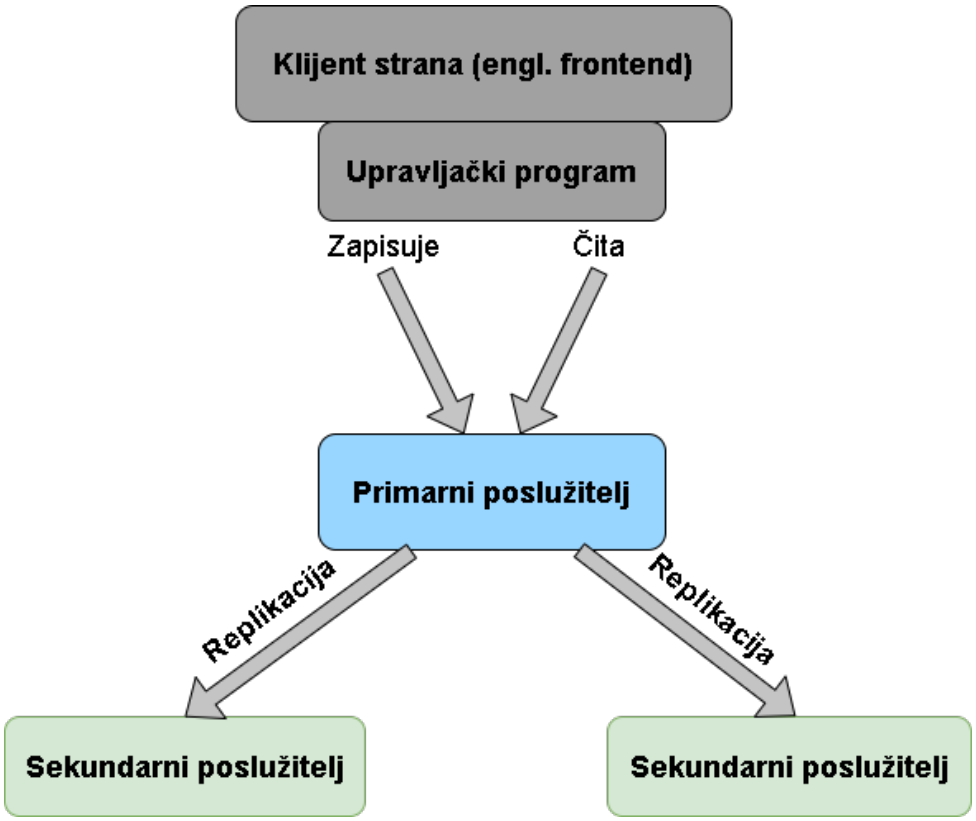
3.4.1. Fragmentiranje

Fragmentiranje [25] uključuje dijeljenje skupa podataka i ukupnog opterećenja sustava na više poslužitelja, što rezultira povećanjem ukupnog kapaciteta čitanja i pisanja jer se smanjuje broj operacija kojima upravlja jedan poslužitelj. Takav način rada rezultira manjim troškovima za razliku od povećanja kapaciteta i brzine samo jednog poslužitelja (vertikalno skaliranje) [25], međutim fragmentiranje povećava složenost infrastrukture, što otežava održavanje. Dokumentno-orijentirane baze podataka izvorno su dizajnirane za podršku automatske distribucije podataka na više poslužitelja u razlicitim geografskim regijama [25].

3.4.2. Replikacija

Replikacija [26] podrazumijeva stvaranja kopija skupova podataka na različitim poslužiteljima, što osigurava redundantnost i povećava dostupnost podataka jer se istom skupu podataka može pristupiti s različitih poslužitelja. Replikacija je također vrlo korisna u slučajevima kvara sklopovlja ili pada poslužitelja [26] zato što čuva kopije ili replike istih podataka na više različitih poslužitelja. Ukoliko bi se svi podaci nalazili na samo jednom poslužitelju, pad tog poslužitelja onemogućio bi pristup podacima. Skup poslužitelja koji održavaju isti fragment ili skup podataka naziva se set replika (engl. replica set) [26].

Dokumentno-orijentirane baze podataka sadrže dvije osnovne metode replikacije podataka [26]: nadređeni-podređeni (engl. master-slave) replikacija i ravnopravna (engl. peer-to-peer) replikacija. MongoDB koristi master-slave replikaciju, stoga će u nastavku biti prikazan i opisan primjer takve vrste replikacije. Master-slave replikacija podataka stvara skupove replika (engl. Replica Set) koji održavaju istu kopiju skupa podataka [27] na način da primarni poslužitelj prima sve operacije pisanja i bilježi sve promjene podataka, dok sekundarni poslužitelji kopiraju i primjenjuju te iste promjene u asinkronom procesu. Ako primarni poslužitelj prestane raditi, jedan od sekundarnih poslužitelja automatski preuzima ulogu primarnog. Na primjer, ako postoji jedan primarni (master) i četiri sekundarna (slave) poslužitelja, pisanje se može dogoditi samo na primarnom poslužitelju, dok na zahtjeve čitanja mogu odgovarati svih pet poslužitelja (i primarni i sekundarni), što omogućuje povećanje kapaciteta čitanja [26]. Na slici 5 prikazana je replikacije u MongoDB-u.



Slika 5: Prikaz replikacije u MongoDB-u; izvor: [26]

4. Usporedba MongoDB-a i MySQL-a

Prilikom odabira baze podataka i sustava za upravljanje bazom podataka vrlo je bitno detaljno sagledati što aplikacija zahtijeva te na temelju toga odabrati najbolju opciju. Količina i struktura podataka koje baza podataka mora pohraniti, očekivani broj korisnika aplikacije, sigurnost, skalabilnost i dosljednost baze podataka koju aplikacija zahtijeva, učestalost mijenjanja sheme baze podataka, preferirani programski jezik, itd. samo su neki od čimbenika za odabir baze podataka prikladne aplikaciji. Osim toga, odabir baze podataka može biti i pitanje pristupa, a ne čisto tehničke odluke te će neki korisnici možda odabrati onu bazu podataka s kojom im je jednostavnije raditi ne uzimajući u obzir tehničke čimbenike. MySQL [28] je dobar izbor za rad sa strukturiranim podacima i tradicionalnom relacijskom bazom podataka koja neće puno rasti, a sigurnost podataka jedan je od glavnih pripriteta. S druge strane, MongoDB [28] je odličan izbor ukoliko su podaci koje aplikacije koristi nestrukturirani ili dinamični, ili ako nije moguće unaprijed definirati shemu baze podataka. Ukoliko aplikacija sadrži velik broj korisnika s tendencijom rasta količine podataka koje je potrebno pohraniti, MongoDB će se pokazati kao bolja opcija od MySQL-a. Na primjer, MongoDB je prikladan za korištenje u aplikacijama za analitiku u stvarnom vremenu, sustavima za upravljanje sadržajem, mobilnim aplikacijama, igrama, itd. [29]. U nastavku rada su detaljnije prikazane prednosti i nedostaci te razlike između MySQL-a i MongoDB-a.

"MySQL je popularan sustav za upravljanje relacijskim bazama podataka razvijen od strane Oracle-a. Kao i ostali sustavi za upravljanje relacijskim bazama podataka, MySQL pohranjuje podatke pomoću tablica i redaka, provodi referentni integritet i koristi SQL za komunikaciju s bazom podataka." [30]

MongoDB [30] je dokumentno-orijentirana baza podataka koja za razliku od MySQL-a podatke pohranjuje u obliku dokumenata (binarni JSON (BSON)), umjesto u tablice i retke. MongoDB [11] je vrlo fleksibilna baza podataka, što znači da omogućava jednostavnu prilagodbu aplikaciji u sklopu koje se koristi. Osim fleksibilnosti, karakterizira ju i horizontalna skalabilnost koja raspoređuje opterećenje na više poslužitelja.

Tablica 1: Ključne razlike između MongoDB-a i MySQL-a; izvor: [31]

MongoDB	MYSQL
MongoDB predstavlja podatke kao JSON dokumente.	MySQL predstavlja podatke u tablicama i redovima.
Nije potrebno definirati shemu, već se samo dodaju dokumenti koji čak nemoraju imati ista polja	Potrebno je definirati tablice i stupce prije pohrane podataka, a svaki redak u tablici mora imati iste stupce
MongoDB koristi MongoDB Query Language (MQL) kao jezik upita.	MySQL koristi SQL.
Ima sposobnost rukovanja velikim nestrukturiranim podacima	MySQL je prilično spor u usporedbi s MongoDB-om dok radi s velikim bazama podataka.
Idealan izbor za nestrukturirane i/ili strukturirane podatke s potencijalom brzog rasta.	Izvrstan izbor za strukturirane podatke i tradicionalnu relacijsku bazu podataka.

4.1. Jednostavnost korištenja

Kao što je navedeno u sekciji 3, dokumenti pohranjeni u dokumentno-orijentiranoj bazi podataka se jednostavno preslikavaju na tipove podataka koje koriste moderni, objektno-orijentirani programski jezici poput JavaScript-a. Osim toga, moguće je pohraniti sve podatke jednog objeka unutar jednog dokumenta, što utječe na složenost programskog koda, a programerima olakšava rad zato što ne moraju podatke jednog entiteta rastavljati na manje dijelove. S druge strane, relacijske baze podataka zahtijevaju raspodijelu podataka jednog entiteta na način prikladan za spremanje unutar više tablica. Jednostavna prilagodba sheme dokumentne baze podataka olakšava nadogradnju aplikacije bez potrebe za skupim migracijama sheme kao kod relacijske baze podataka [30]. MongoDB, kao predstavnik dokumentno-orijentirane baze podataka u ovom radu, podržava sva prethodno navedena svojstva te kao takav ima prednosti nad odabranim sustavom za upravljanje relacijskom bazom podataka, MySQL. Slijedom toga, velik broj tvrtki odlučuje se na korištenje dokumentnih baza podataka poput MongoDB-a [30], koja omogućuje jednostavniju izradu aplikacija za upravljanje velikim skupovima raznolikih tipova podataka.

4.1.1. Izvršavanje upita

Svaki sustav za upravljanje bazom podataka koristi neki upitni jezik koji predstavlja sredstvo komunikacije između sustava za upravaljnje bazom podataka i same baze podataka. Drugim riječima, upitni jezik služi za pisanje, ažuriranje, brisanje i čitanje iz baze podataka. MongoDB za postavljanje upita koristi vlastiti MongoDB upitni jezik (engl. MongoDB query language (MQL)) baziran na JavaScript-u [30], dok MySQL, kao i većina relacijskih baza podataka, koristi strukturni upitni jezik (SQL). Oba navedena upitna jezika podržavaju operacije stvaranja, čitanja, ažuriranja, brisanja (engl. create, read, update, delete (CRUD)), agregaciju podataka, pretraživanje teksta i geoprostorne upite.

Iako su MQL upiti jednostavniji za formiranje, kao što se može vidjeti na slici 6, vrlo je bitno naglasiti da SQL općenito ima prednost prilikom timskog korištenja jer gotovo svi popularni sustavi za upravljanje relacijskom bazom podataka koriste upravo SQL [28]. Shodno tome, članovi razvojnog tima koji nemaju iskustva u radu MySQL-om, vrlo će se lako prilagoditi MySQL-u ukoliko su upoznati s ,na primjer, PostgreSQL-om. S druge strane, MQL se uvelike razlikuje od upitnih jezika drugih dokumentnih baza podataka poput CouchDB-a, što rezultira zahtijevnijim prelaskom s MongoDB-a na CouchDB.

MongoDB	MySQL
INSERT	
db.users.insert({name: 'John',age: 35})	INSERT INTO users(name, age) VALUES ('John', 35)
SELECT	
db.users.find({name: 'John'})	SELECT * FROM users WHERE name = 'John'
UPDATE	
db.users.update({name: 'John'}, {$set: {age: 40}})	UPDATE users SET age = 40 WHERE name = 'John'
DELETE	
db.users.remove({name: 'John'})	DELETE FROM users WHERE name = 'John'

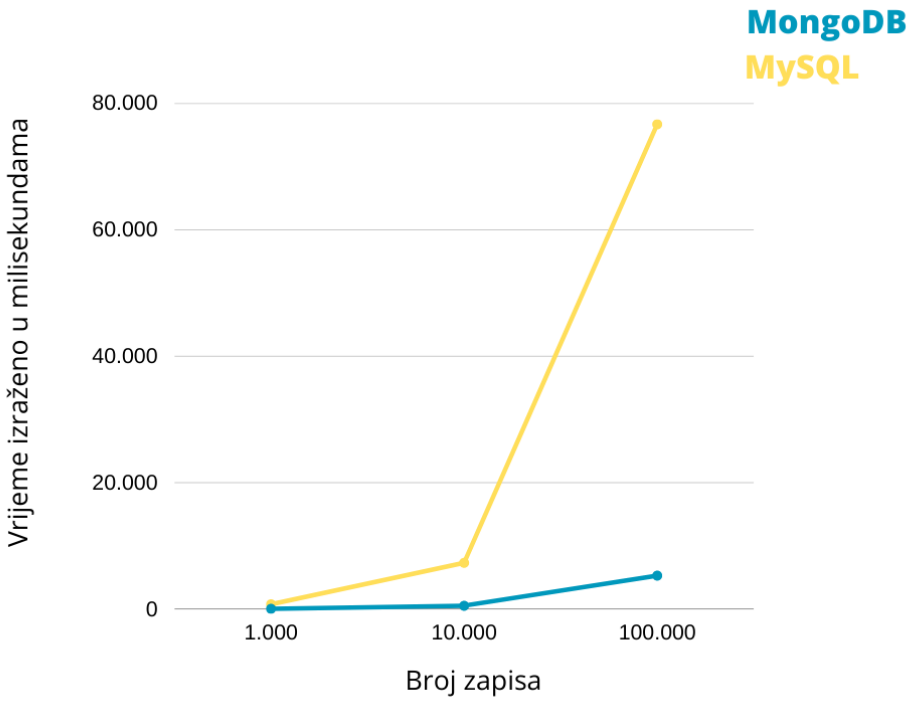
Slika 6: Primjeri CRUD operacija u MongoDB-u i MySQL-u

4.2. Performanse

Prilikom odabira baze podataka, potrebno je znati što se može očekivati od koje baze podataka, za koje svrhe i koji su prioriteti aplikacije (sigurnost, dostupnost, način pohrane podataka, itd.). Jedan od čimbenika za odabir je svakako i brzina baze podataka jer je cilj svake aplikacije omogućiti korisnicima što brži rad. Slijedom toga, u nastavku rada je prikazana i opisana usporeba brzine izvođenja CRUD operacija korištenjem MongoDB-a i MySQL-a.

Tablica 2: Vrijeme potrebno za operaciju umetanja u milisekundama; izvor: [32]

Broj zapisa	MYSQL	MongoDB
1.000	757,1	54,9
10.000	7.326,4	533,8
100.000	76.705,7	5.282,5

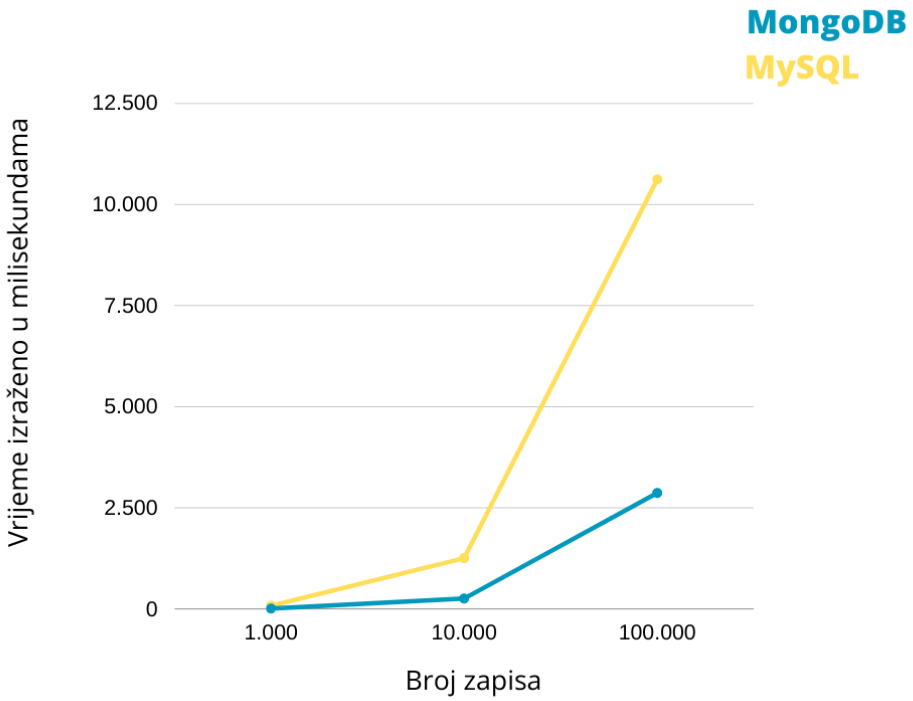


Slika 7: Vrijeme potrebno za operaciju umetanja u milisekundama; izvor: [32]

Tablica 2 prikazuje performanse umetanja 1.000, 10.000 i 100.000 zapisa s MongoDB-om i MySQL-om. Na temelju dobivenih rezultata može se uočiti da je MongoDB bio otprilike 14 puta brži od MySQL-a u sve tri izvršene operacije umetanja. Umetanje 100.000 zapisa s MySQL-om predstavlja problem jer izvršavanje operacije traje skoro 77 sekundi, dok MongoDB treba svega 5 sekundi za postizanje istog. Na slici 7 vidi se i grafički prikaz dobivenih rezultata umetanja.

Tablica 3: Vrijeme potrebno za operaciju ažuriranja u milisekundama; izvor: [32]

Broj zapisa	MYSQL	MongoDB
1.000	87,7	17,3
10.000	1.264	265,4
100.000	10.620,5	2.875,9

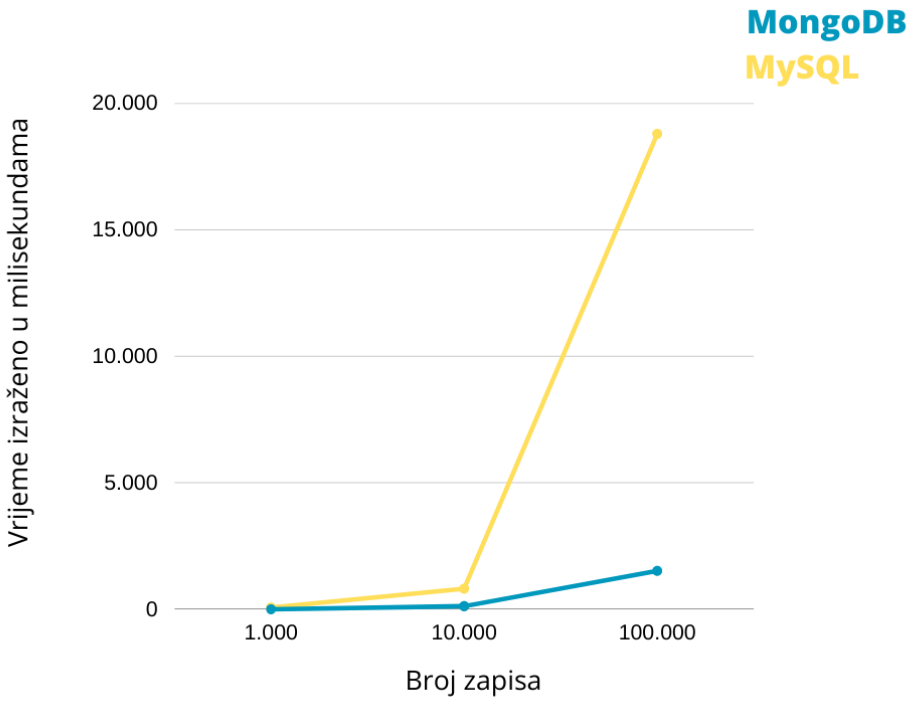


Slika 8: Vrijeme potrebno za operaciju ažuriranja u milisekundama; izvor: [32]

Na tablici 3 prikazani su rezultati ažuriranja 1.000, 10.000 i 100.000 zapisa s MongoDB-om i MySQL-om. Za razliku od umetanja podataka, ažuriranje se izvršava dosta brže, a ta razlika se najviše vidi između umetanja i ažuriranja 100.000 zapisa s MySQL-om. Ažuriranje podataka s MongoDB-om je optrilike 5 puta brže nego s MySQL-om. Slika 8 grafički prikazuje opisane rezultate.

Tablica 4: Vrijeme potrebno za operaciju brisanja u milisekundama; izvor: [32]

Broj zapisa	MYSQL	MongoDB
1.000	78,3	9
10.000	825,8	133,8
100.000	18.794,4	1.530,9

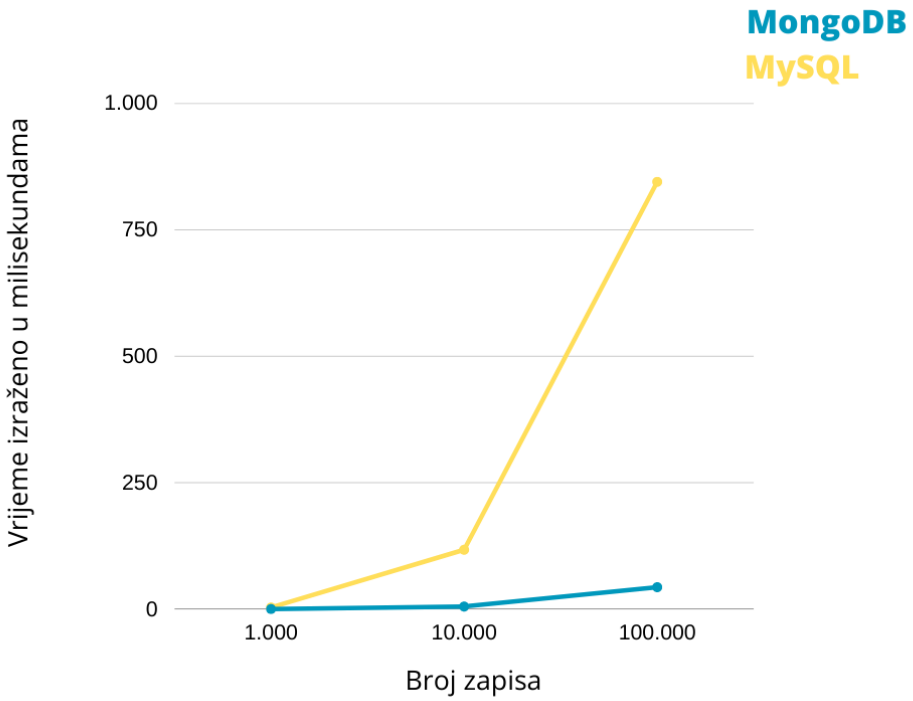


Slika 9: Vrijeme potrebno za operaciju brisanja u milisekundama; izvor: [32]

Na tablici iznad 4 vide se rezultati brisanja 1.000, 10.000 i 100.000 zapisa s MongoDB-om i MySQL-om. Na temelju dobivenih rezultata se može uočiti da je operacija brisanja brža od operacije ažuriranja, osim prilikom korištenja MySQL-a u radu sa 100.000 zapisa, gdje je ažuriranje nešto brže od brisanja. Na slici 9 vidi se grafički prikaz podataka iz tablice 4.

Tablica 5: Vrijeme potrebno za operaciju čitanja u milisekundama; izvor: [32]

Broj zapisa	MYSQL	MongoDB
1.000	4,1	1
10.000	117,8	6
100.000	844,8	43,5



Slika 10: Vrijeme potrebno za operaciju čitanja u milisekundama; izvor: [32]

U tablici 5 nalaze se rezultati izvršavanja čitanja 1.000, 10.000 i 100.000 zapisa s MongoDB-om i MySQL-om. Izvršavanje operacije čitanja 1.000 zapisa s MongoDB-om se izvršilo 4 puta brže nego s MySQL-om, što ne čini preveliku razliku u odnosu na čitanje 10.000 i 100.000 zapisa, gdje je MongoDB bio gotovo 20 puta brži od MySQL-a. Nakon izvođenja svih CRUD operacija, može se uočiti da je čitanje najbrža operacija. Slika 10 prikazuje grafički prikaz dobivenih rezultata.

Kao što je vidljivo na prethodnim grafovima, MongoDB puno brže izvršava osnovne CRUD operacije od MySQL-a. Razlog tome je bolja mogućnost indeksiranja te mogućnost pohrane velike količine nestrukturiranih podataka unutar ugnježđenih dokumenata što omogućava brže pisanje i čitanje podataka . S druge strane MySQL pokazuje dosta sporije performanse prilikom rada s velikom količinom podataka iz razloga što pohranjuje podatke na normaliziran način, što znači da upit mora prolaziti kroz velik broj tablica kako bi pronašao potrebne zapise [33]. Takav način pohrane svakako utječe na brzinu izvedbe upita.

4.3. Fleksibilnost

Prilikom odabira baze podataka za aplikaciju vrlo je bitno gledati na fleksibilnost iste. Fleksibilnost baze podataka određuje jednostavnost njezine prilagodbe aplikaciji, odnosno projektu za koji se koristi. MongoDB pruža visoku razinu fleksibilnosti [30] jer ne sadrži strogo definiranu shemu koje se mora pridržavati prilikom pohranjivanja dokumenata, što znači da dokumenti unutar kolekcije ne moraju imati isti broj polja, a polje u dokumentu može pohranjivati različite tipove podataka. Također, MongoDB omogućava pohranjivanje više dokumenata u kolekciju bez ikakve međusobne veze. S druge strane, MySQL ima lošu fleksibilnost jer zahtijeva definiranje stroge sheme prije spremanja podataka, odnosno potrebno je definirati tablice i stupce na način da svi zapisi u tablici imaju iste stupce. Razina fleksibilnosti baze podataka pogotovo dolazi do izražaj u radu s velikim skupovima podataka (engl. Big Data). Dakle, ukoliko aplikacija zahtjeva učestale promjene shema, MongoDB je definitivno bolja opcija od MySQL-a [33].

4.4. Skalabilnost

Rastom aplikacije, odnosno povećanjem skupova podataka koje aplikacija koristi te povećanjem broja korisnika i prometa aplikacije, dolazi do sve većeg broja zahtjeva za čitanje i pisanje u bazu podataka. Ukoliko baza podataka nije u stanju učinkovito podržavati sve zahtijeve, korisnici će se suočavati s učestalim rušenjem, zastojem i kašnjenjem u radu s aplikacijom, što će rezultirati gubitkom korisnika. Izbjegavanje takvog scenarija postiže se povećanjem kapacitata baze podataka s ciljem učinkovitog odgovaranja na sve zahtijeve čitanja i pisanja. Proširenje kapaciteta baze podataka može se još definirati kao skaliranje baze podataka [23]. U sekciji 3.4 navedena su i detaljnije opisana dva načina skaliranja: vertikalno i horizontalno.

MongoDB ima mogućnost vertikalnog i horizontalnog skaliranja, što znači da podržava povećanje snage (CPU, RAM) jednog poslužitelja (vertikalno skaliranje), dodavanje novih poslužitelja te fragmentiranje ili dijeljenje skupova podataka i opterećenja preko više poslužitelja (horizontalno slakiranje) [23]. MongoDB podržava metode replikacije i fragmentacije opisane u podsekcijama 3.4.1 i 3.4.2.

MySQL omogućuje vertikalno skaliranje i ograničeno horizontalno skaliranje, odnosno stvaranje ograničenih replika poslužitelja samo za čitanje [33]. Takav ograničen način replikacije nije previše koristan za aplikacije koje zahtjevaju velik broj pisanja u bazu podataka. Horizontalno skaliranje u relacijskim bazama podataka je teško postići jer operacije spajanja više tablica (engl. JOIN) postaju neučinkovite ukoliko se te tablice nalaze na više različitih poslužitelja. Osim toga, teško je osigurati dosljednost podataka u situaciji u kojoj se podaci u jednoj tablici mogu mjenjati sa više različitih poslužitelja. Upravo zato nerelacijske baze podataka smanjuju dosljednost kako bi povećale dostupnost. MySQL je problem horizontalnog skaliranja riješio razvojem MySQL klastera (engl. MySQL Cluster), koji osigurava visoku razinu dostupnosti i horizontalne skalabilnosti [34] te pritom ne narušava doslijednost i konzistentnost baze podataka.

4.5. Sigurnost korištenja

Za neke slučajeve upotrebe sigurnost je vrlo važna npr. za zaštitu vrijednih, povjerljivih ili osjetljivih podataka, stoga je prilikom uspoređivanja dva sustava za upravljanje bazama podataka vrlo bitno sagledati sigurnost istih, zbog sve većeg broja kibernetičkih napada diljem svijeta. Dok NoSQL baze podataka nude bolju skalabilnost i fleksibilnost za rukovanje velikom količinom podataka, one imaju neke sigurnosne probleme [35] koje pružatelji usluga i istraživači pokušavaju riješiti. U nastavku je opisana usporedba MongoDB-a i MySQL-a temeljem njihove sigurnosti.

MongoDB, kao i Mysql ima omogućenu kontrolu pristupa [30] što znači da se svakom korisniku dodjeljuje uloga koja određuje koje radnje korisnik smije izvršavati. Međutim, MySQL pruža strožu kontrolu pristupa koja, osim dodjeljivanja uloga, omogućava dodjeljivanje privilegija i dopuštenja za određene operacije, objekte i skupove podataka u bazi [33]. Što se tiče autentifikacije, MySQL pruža podršku za dvostruku lozinku koja omogućava dodatnu sigrunost u odnosu na MongoDB. Oba sustava pružaju enkripciju podataka podržanu kriptografskim protokolom (engl. Transport Layer Security (TLS)) i certifikatom za stvaranje šifrirane veze (engl. Secure Sockets Layer (SSL)) [33]. Stroga arhitektura, kakvu koristi MySQL, također ide u prilog sigurnosti jer osigurava veću razinu pouzdanosti i konzistentnosti podataka nego MongoDB.

4.6. Primjeri korištenja MySQL-a

Kao što je prethodno navedeno, MySQL ima široku primjenu u današnjici, a neke od osnovnih upotreba MySQL-a su [33]:

- Aplikacije za vođenje evidencija (Logging applications);
- Skladištenje podataka (engl. Data warehousing);
- Aplikacije s visokom razinom sigurnosti podataka poput društvenih mreža Facebook ili Instagram;
- Softver kao usluga (engl. Software as a service (SaaS)) aplikacije;

Neke od poznatijih organizacija koje koriste MySQL su [33]: Airbnb, NASA, Sony, YouTube, Netflix, Wikipedija, Facebook, itd.

4.7. Primjeri korištenja MongoDB-a

Velik broj organizacija diljem svijeta koristi MongoDB za pohranu velikih skupova podatka upravo zbog svih prednosti koje su pethodno navedene. Neki od osnovnih primjera korištenja MongoDB-a su [33]:

- Sustavi za upravljanje sadržajem, poput WordPress-a;
- Upravljanje velikim skupovima podataka za web i mobilne aplikacije;
- Internet stvari (engl. Internet of Things (IoT));
- Održavanje geoprostornih ili lokacijskih podataka;
- Aplikacije za personalizaciju;
- Katalozi proizvoda e-trgovine i upravljanje imovinom;
- Analitika u stvarnom vremenu velikom brzinom ;

Neke od poznatijih organizacija koje koriste MongoDB su [33]: Twitter, IBM, Oracle, Zendesk, Citrix, Sony, Intercom, HTC

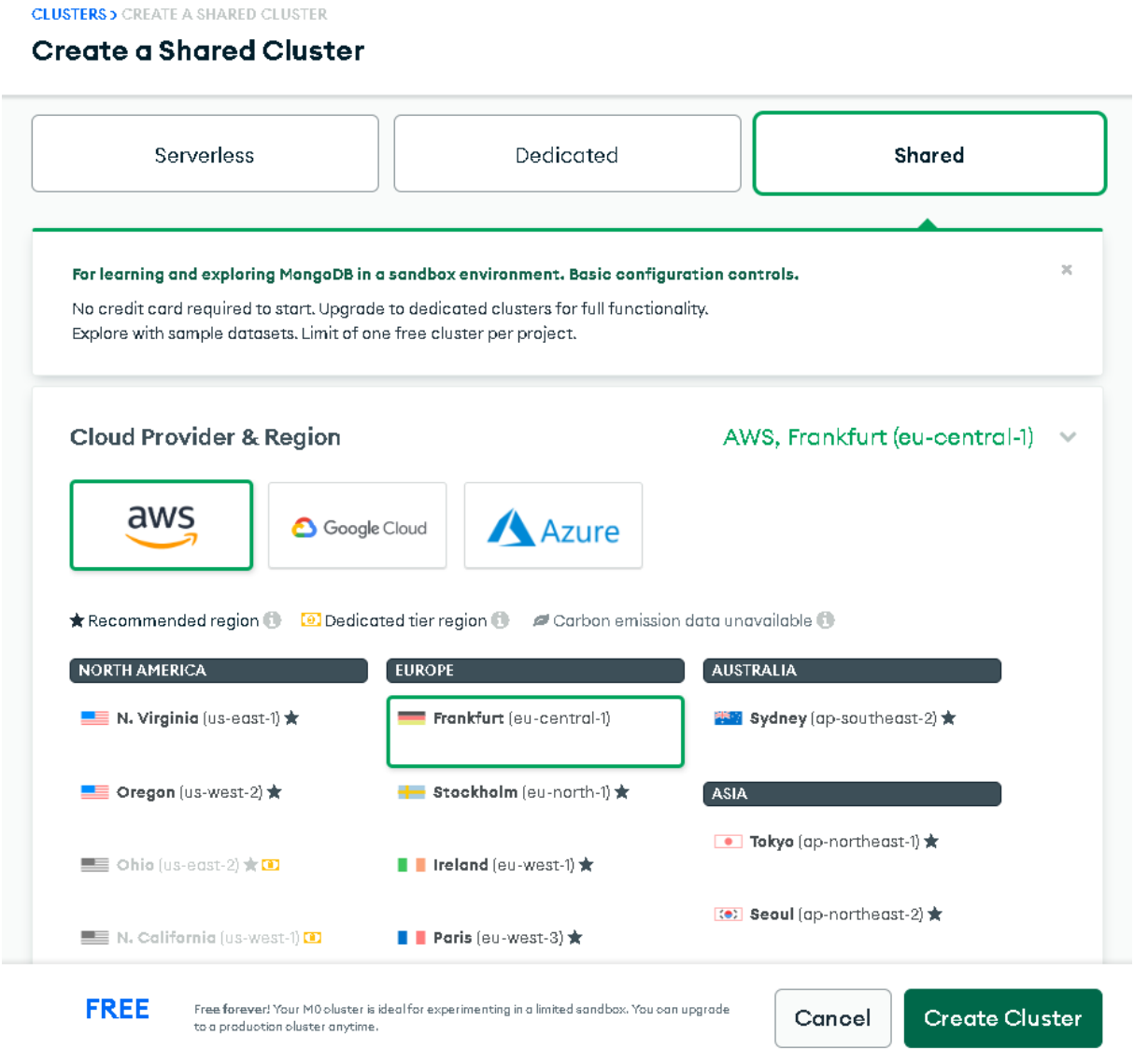
5. Primjer fitness aplikacije za praćenje napretka

Fitnes industrija raste svakim danom te se sukladno tome povećava i broj osoba koje se počinju baviti fitnes-om, kako bi poboljšale razinu vlastitog zdravlja i tjelesne performanse. Također, svaka osoba želi ostvariti što bolje rezultate u što kraćem vremenskom razdoblju, stoga je vrlo bitno održavati kvalitetan plan treninga i zdravu ishranu. Osim toga, početnicima, koji ne žele plaćati fitnes trenere, potrebno je na neki način prezentirati pravilno izvođenje vježbi kako ne bi došlo do neželjenih ozljeda. Upravo to su neki od glavnih razloga za izradu aplikacije koja će vježbačima olaškati cijelokupni proces praćenja napretka te pružiti mogućnosti kao što su: pretraživanje vježbi prema odabranoj mišićnoj skupini, pregled podataka o tjelesnim performansama, praćenje dnevnog unosa kalorija, pregled markonutrijena pojedinog obroka te izrada vlastitog personaliziranog plana treninga.

U nastavku rada opisana su dva načina implementacije apikacije koja sadrži sve prethodno navedene funkcionalnosti. Obje implementacije sastoje je od sučelja (`https://github.com/DominikLunko/admin-dashboard-app`), te poslužitelja aplikacije. Prva implementacija realizirana je uz pomoć dokumentno-orijentirane baze podataka MongoDB (`https://github.com/DominikLunko/angular-mongodb-API`), a druga s Firebase-om (`https://github.com/DominikLunko/angular-firebase-api`). Aplikacija je javno dostupna na: `https://fitness-app-dl.netlify.app`.

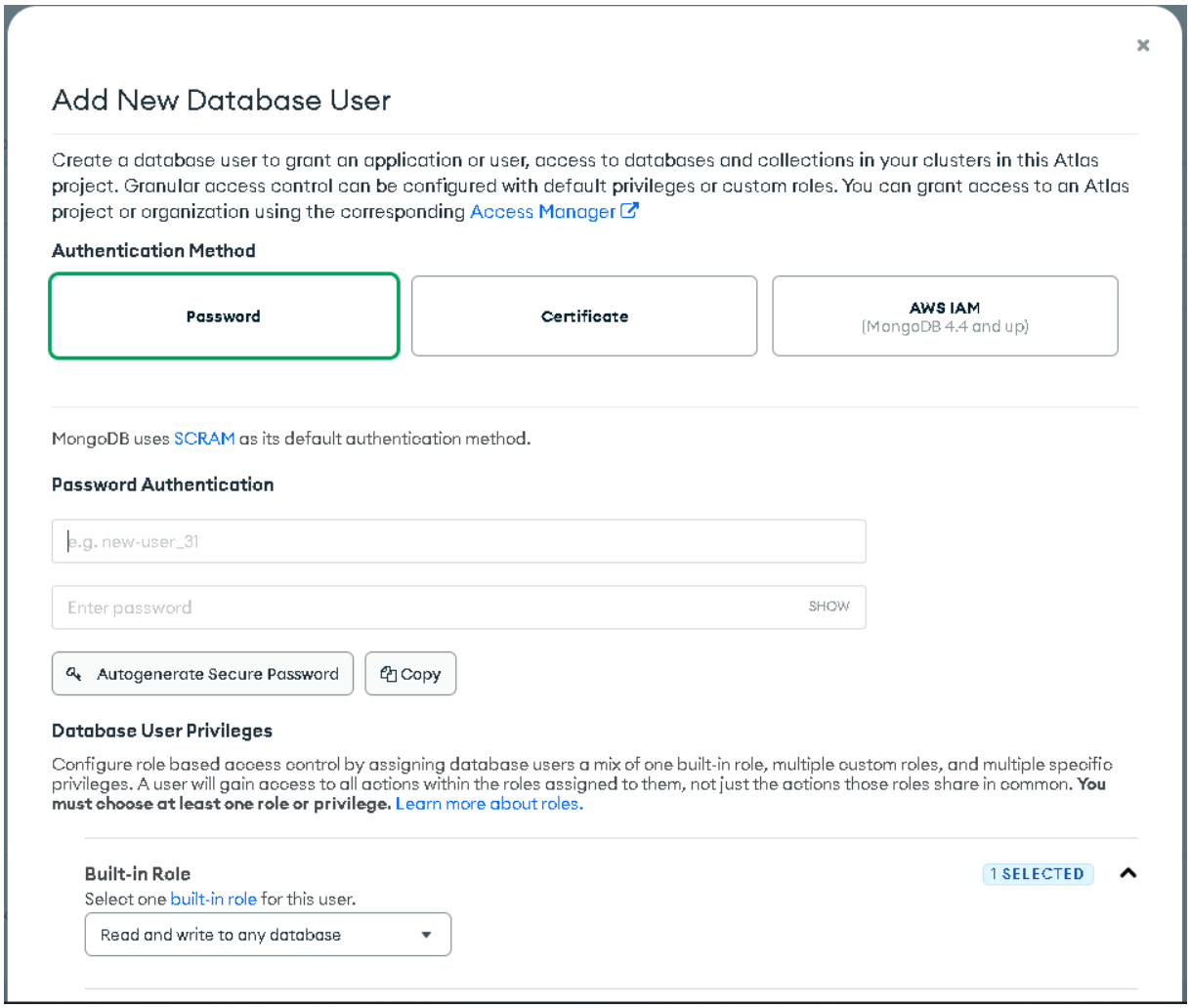
5.1. Integracija MongoDB Atlas-a i NodeJS-a

Integracija MongoDB Atlas-a i NodeJS-a je poprilično jednostavan proces koji se sastoji od tri glavna koraka, a to su kreiranje MongoDB Atlas klastera, kreiranje korisnika te postava IP pristupne liste i konekcija na bazu podataka. MongoDB Atlas klaster [36] predstavlja skup baza podataka kojima upravlja jedna instanca pokrenutog poslužitelja. Za kreiranje novog klastera potrebno je prijaviti se na MongoDB Atlas korisnički račun nakon čega se automatski otvara prozor za kreiranje novog klastera. U tom prozoru potrebno je odabrati način rada te pružatelja usluga i regiju pružatelja usluga. Nakon odabira pritiskom na gumb "kreiraj klaster" (engl. create cluster) završava proces kreiranja klastera. Na slici 11 prikazan je ekran za kreiranje novog MongoDB Atlas klastera.

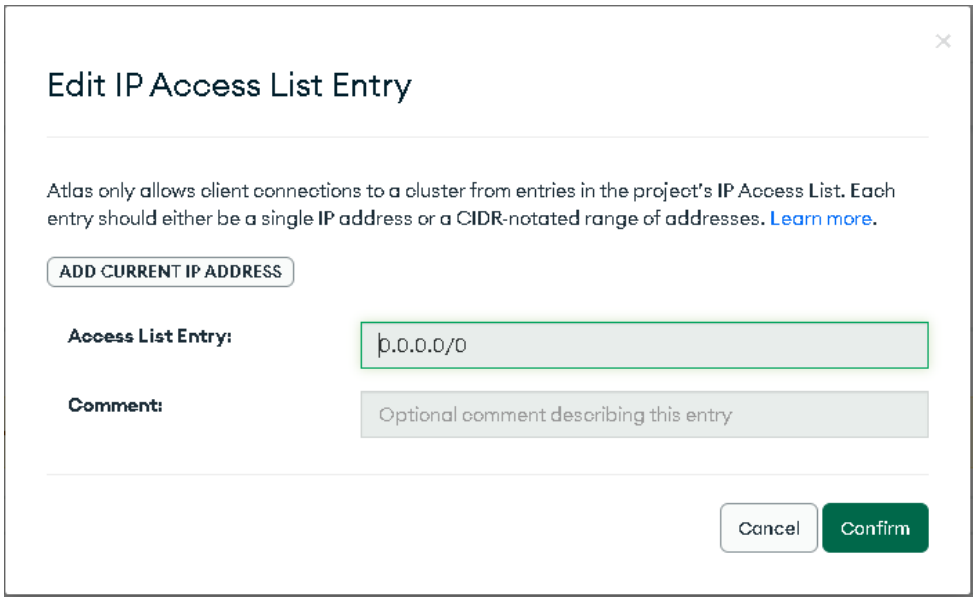


Slika 11: Kreiranje MongoDB Atlas klastera

Nakon uspješno kreiranog klastera potrebno je kreirati korisnika i odrediti pristupnu IP adresu, a za to je potrebno odabrati opciju "pristup bazi podataka" i kliknuti na gumb "dodaj novog korisnika", nakon čega se otvara skočni prozor u kojem je potrebno unijeti korisničko ime i lozinku. Osim toga, moguće je promijeniti neke konfiguracijske postavke kao što su korisnička uloga, zabrane, itd. Na slici 12 prikazan je postupak kreiranja novog korisnika. Sljedeći korak je dodavanje IP adresa na listu dopuštenih IP adresa,a za to je potrebno odabrati opciju "pristup mreži" te u skočnom prozoru dodati željene IP adrese. Na slici 13 prikazano je dodavanje IP adresa na listu dopuštenih IP adresa.

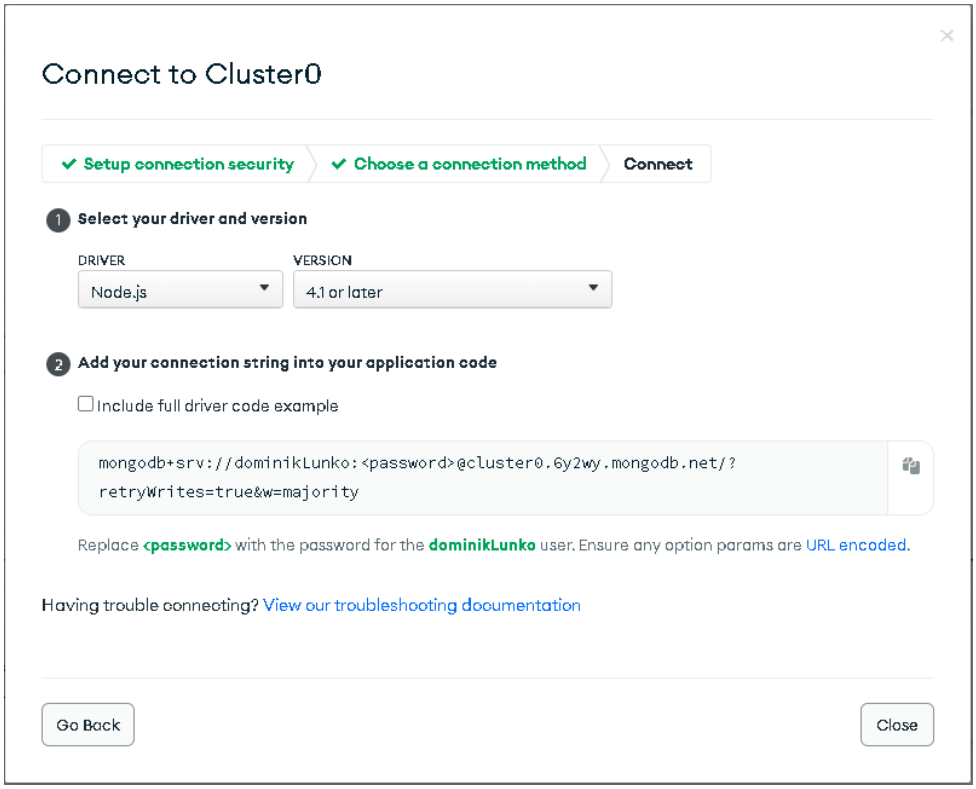


Slika 12: Kreiranje novog korisnika



Slika 13: Dodavanje dopuštenih IP adresa

Posljednji korak u integraciji je uspostava konekcije na bazu podataka. Za postavu konekcije potrebno je odabrati opciju "poveži" (engl. connect) nakon čega se otvara skočni prozor u kojem je potrebno odabrati način ili metodu spajanja na bazu podataka. U ovom slučaju odabran je način "poveži aplikaciju" (engl. connect your application). Na kraju je potrebno konekcijsku poveznicu kopirati u NodeJS aplikaciju za daljnje korištenje. Na slici 14 prikazan je ekran za povezivanje MongoDB Atlas-a i NodeJS-a.



Slika 14: Povezivanje MongoDB Atlas-a s NodeJS-om

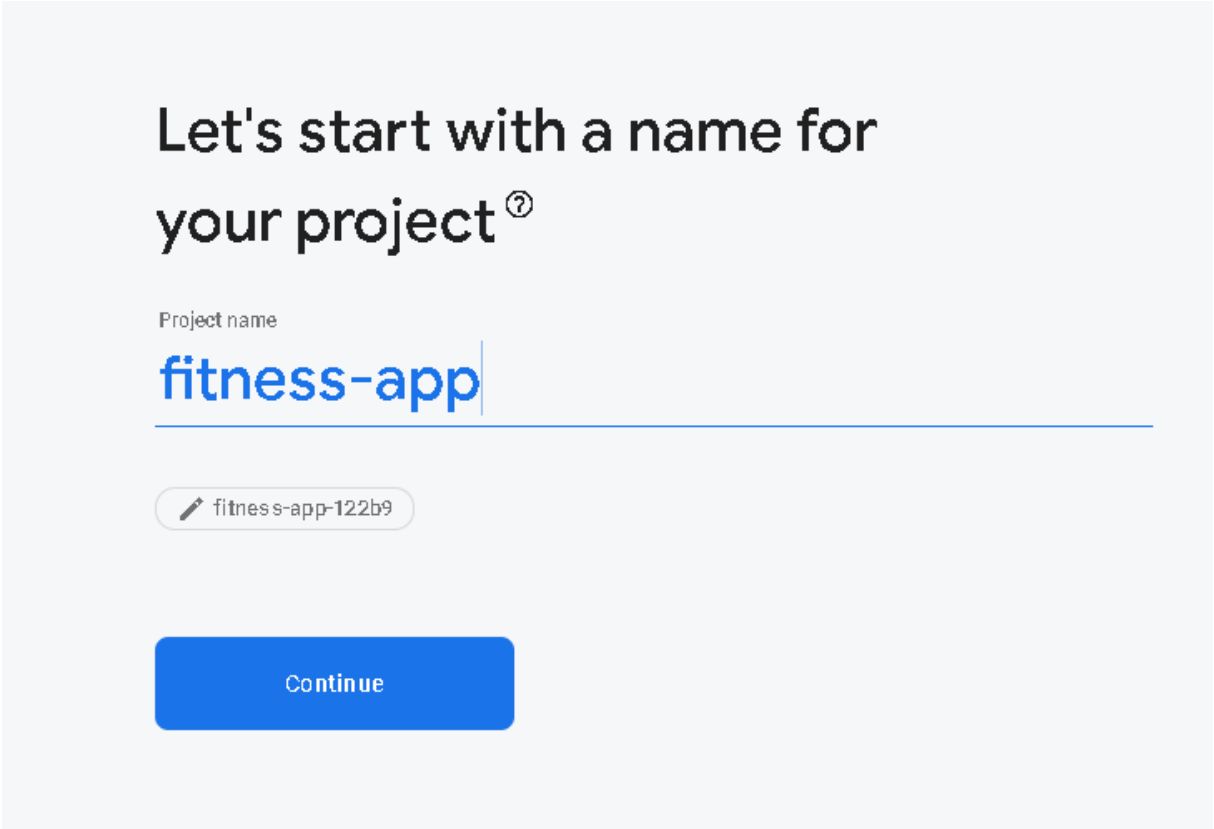
Nakon uspješnog postavljanja svih potrebnih postavki, potrebno je u NodeJS aplikaciji instalirati Mongoose biblioteku uz pomoć NodeJS upravitelja paketima (engl. Node package manager(NPM)) i naredbe `npm install mongoose`. Nakon izvršavanja instalacije, Mongoose biblioteka je spremna za korištenje te se uz pomoć nje ostvaruje konekcija na bazu podataka na sljedeći način 12:

Isječak kôda 12: Spanjanje na MongoDB

```
1  import mongoose from 'mongoose';
2
3  const connectionString = '###'
4
5  const connectDB = async () => {
6      try {
7          await mongoose.connect(connectionString, {
8              useNewUrlParser: true,
9              useUnifiedTopology: true
10         });
11         console.log("MongoDB connection SUCCEEDED");
12     } catch (error) {
13         console.log(error)
14         console.error("MongoDB connection FAILED.");
15         process.exit(1);
16     }
17 }
18
19 export default connectDB;
```

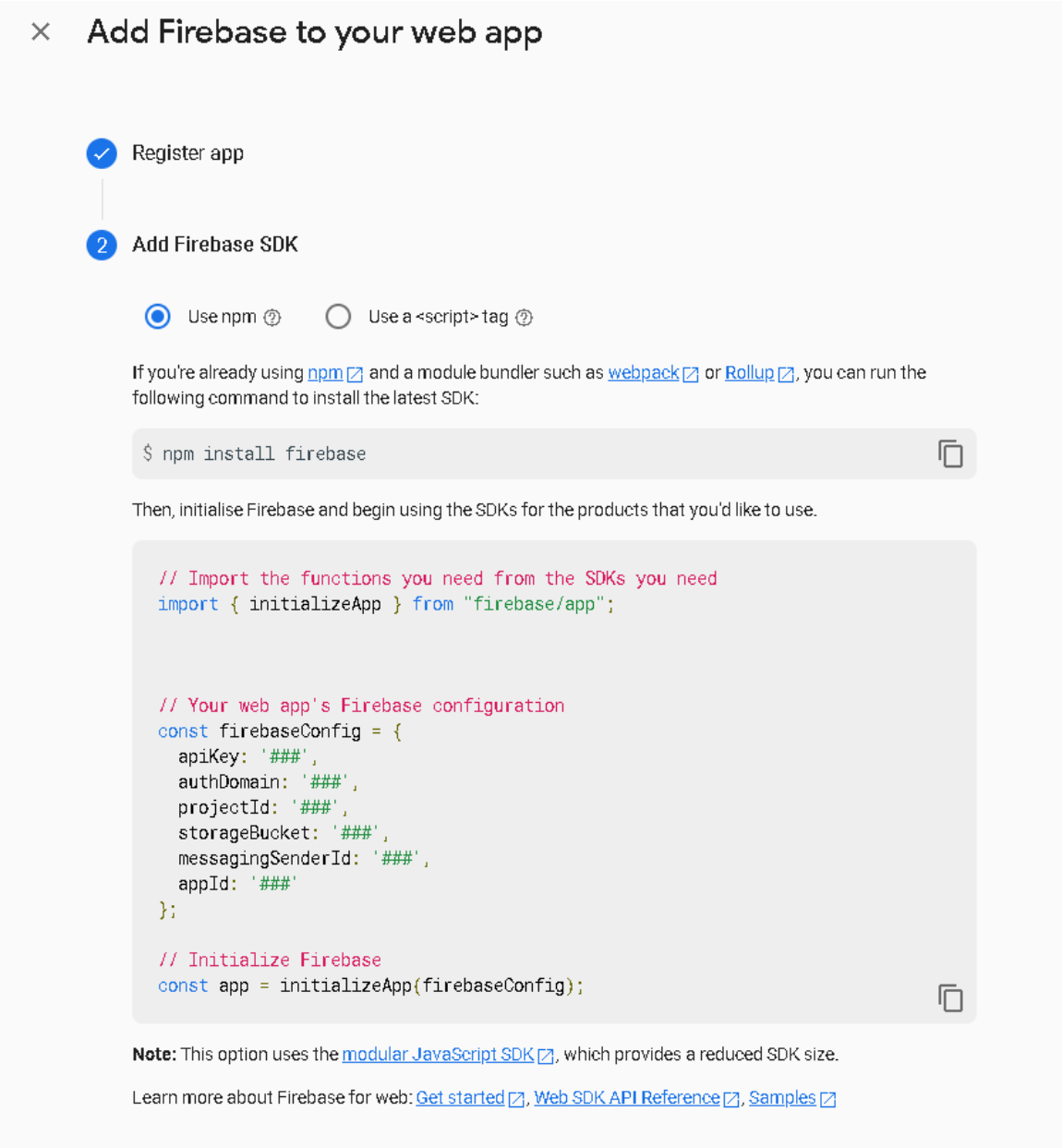
5.2. Integracija Firebase Firestore-a i NodeJS-a

Integracija Firebase Firestore-a i NodeJS-a također je vrlo jednostavan proces koji započinje kreiranjem novog Firebase projekta odabirom opcije "dodaj novi projekt". Nakon odabira opcije za kreiranje novog projekta, otvara se ekran na kojem je potrebno upisati ime projekta. Na slici 15 prikazan je ekran za kreiranje novog projekta.



Slika 15: Kreiranje novog Firebase projekta

Nakon uspješnog kreiranja novog projekta, potrebno je registrirati Firebase aplikaciju, na temelju čega se generira konfiguracija za inicijaliziranje Firebase aplikacije. Na slici 16 prikazan je ekran za registraciju Firebase aplikacije.



Slika 16: Povezivanje Firebase-a s web aplikacijom

Uz pomoć NPM-a i naredbe `npm install firebase` potrebno je instalirati firebase biblioteku. Nakon instalacije Firebase biblioteke, pozivom metode `initializeApp(config)` inicijalizira se zadana instanca Firebase aplikacije na temelju proslijeđene konfiguracije. Nakon inicijaliziranja Firebase aplikacije potrebno je dohvatiti postojeću Firestore i FirebaseAuth instancu povezanu s navedenom Firebase aplikacijom 13.

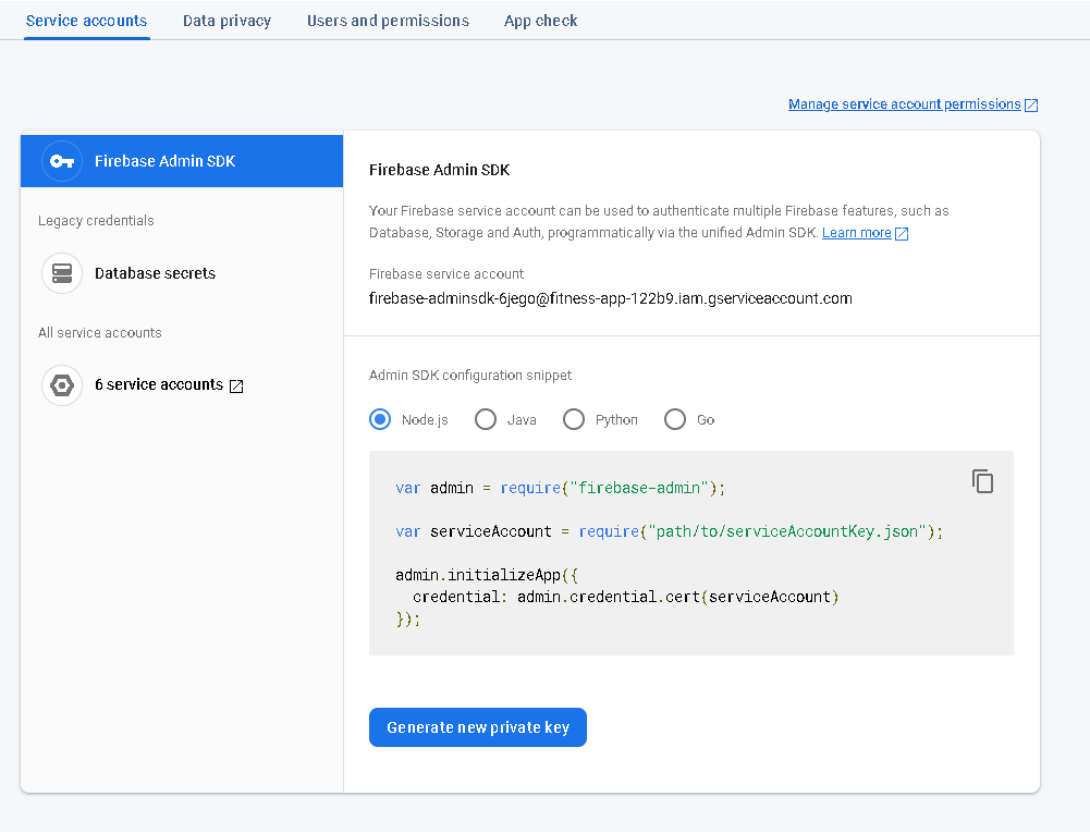
Isječak kôda 13: Inicijalizacija Firebase aplikacije

```
import { initializeApp } from "firebase/app";

import { getAuth } from "firebase/auth";

import { getFirestore } from "firebase/firestore";

const firebaseConfig = {
  apiKey: "###",
  authDomain: "###",
  projectId: "###",
  storageBucket: "###",
  messagingSenderId: "###",
  appId: "###"
};

const firebaseApp = initializeApp(firebaseConfig);

export const db = getFirestore(firebaseApp);
export const auth = getAuth(firebaseApp);
```

Osim Firebase biblioteke, postoji i Firebase-admin biblioteka koja omogućuje interakciju s Firebase-om na privilegiran način [37]. Za korištenje takvog načina rada, potrebno je na početnoj stranici Firebase projekta odabrati opciju "postavke projekta". Klikom na "generiraj novi privatni ključ" započinje preuzimanje JSON datoteke u kojoj se nalazi generirani privatni ključ. Na slici 17 prikazan je ekran za generiranje privatnog ključa. Nakon generiranja privatnog ključa potrebno je u NodeJS aplikaciji instalirati Firebase-admin biblioteku pokretanjem naredbe `npm install firebase-admin`, nakon čega slijedi inicijalizacija Firebase-admin aplikacije 14.

Isječak kôda 14: Inicijalizacija Firebase-admin aplikacije

```
import admin from "firebase-admin";

admin.initializeApp({
  credential: admin.credential.cert({
    "type": "service_account",
    "project_id": "###",
    "private_key_id": "###",
    "private_key": "###",
    "client_email": "###",
    "client_id": "###",
    "auth_uri": "###",
    "token_uri": "###",
    "auth_provider_x509_cert_url": "###",
    "client_x509_cert_url": "###"
  })
});
```



Slika 17: Firebase-admin-generiranje privatnog ključa

5.3. Modeliranje podataka

Kao što je u sekciji 3.2 navedeno, dokumentno-orijentirane baze podataka se vrlo jednostavno prilagođavaju aplikaciji što ih čini vrlo fleksibilnim i jednostavnim za korištenje. Stoga je nakon planiranja svih funkcionalnost potrebno definirati modele podataka potrebne za ispravan rad aplikacije. U nastavku slijedi opis korištenih modela podataka i kratak opis njihovih atributa.

5.3.1. Model podataka "mišićne skupine"

Isječak kôda 15: Model podataka "mišićne skupine"

```
1 {
2   items: ["back", "cardio", "chest", "lower arms", "lower legs" "neck", "shoulders",
      "upper arms", "upper legs", "waist"]
3 }
```

Model podataka "mišićne skupine" 15 sadrži samo jedno polje "items". Polje "items" je niz tekstualnih vrijednosti, a svaki element niza predstavlja jednu mišićnu skupinu.

5.3.2. Model podataka "vježba"

Isječak kôda 16: Model podataka "vježba"

```
{
  bodyPart: "waist",
  equipment: "body weight",
  gifUrl: "http://d205bpvrqc9yn1.cloudfront.net/0001.gif",
  id: "0001",
  name: "3/4 sit-up",
  target: "abs"
}
```

Prilikom dohvata vježbi na temelju odabrane mišićne skupine, potrebno je prikazati ime vježbe, podatak o dijelu tijela kojeg vježba "pogađa", potrebnoj opremi za izvođenje vježbe, način izvođenja vježbe (u "gif" formatu) te pripadajuću mišićnu skupinu. Isječak iz koda 16 prikazuje primjer takvog modela.

5.3.3. Model podataka "varijacija vježbe"

Isječak kôda 17: Model podataka "varijacija vježbe"

```
{
  exerciseName: "Squat (Barbell)",
  reps: 6,
  setOrder: 7
  weight: 315,
  workoutName: "Squat 1",
}
```

Model podataka "varijacija vježbe" 17 pohranjuje podatke o nazivu vježbe, nazivu varijacije vježbe, broju ponavljanja, broju serija i težini s kojom se vježba izvodi. Ovaj model podataka služi za kreiranje planova treninga na način da korisnik pretražuje određenu vježbu prema imenu, a rezltat pretrage vraća različite varijante izvođenja tražene vježbe.

5.3.4. Model podataka "nutrijent"

Isječak kôda 18: Model podataka "nutrijent"

```
{
  calories: 235,
  carbs: 26,
  category: "Daily products",
  fat: 11
  fiber: 0
  food: "Cocoa"
  grams: 252
  measure: "1 cup",
  protein: 8
}
```

Na ekranu nutrijenata korisnik ima mogućnost pretraživanja nutrijenata/namirnica prema željenoj kategoriji i/ili nazivu namirnice. Upravo zato model podataka "nutrijent" 18 sadrži podatak o nazivu i kategoriji nutrijenta. Također je neophodno prikazati i sastav nutrijenta koji se odnosi na količinu kalorija, proteina, masti, ugljikohidrata i vlakana po jedinici doziranja.

5.3.5. Model podataka "korisnik"

Isječak kôda 19: Model podataka "korisnik"

```
{
  name: 'Dominik Lunko',
  email: 'dominiklunko@gmail.com',
  password: '$2a$10$qTyQ4L9ATtlSPkO39TRRn.aExHrWNBqvo51pqkHjbZYVzq53sxwfe',
  activity_level:"level_4"
  age: 24
  gender: "male"
  height: 189
  weight: 93
}
```

Kao što se vidi na 19, model podataka "korisnik" se sastoji od polja: ime i prezime, email, lozinka (koristi se samo kod implementacije s MongoDB Atlasom jer Firebase pruža implementaciju gotove autentifikacije), razina aktivnosti, godina starosti, spol, visina i težina. Svi ostali podaci o korisniku pohranjeni su u modelu "korisnička analitika" 20, zato što prijavom u aplikaciju nije potrebno dohvaćati sve podatke o korisniku, već samo one osnovne. Podaci poput idealne težine, optimalnog unosa kalorija, dnevnog unosa kalorija, itd. dohvaćaju se odabirom na opciju "Personal data" o čemu će biti više riječi u sekciji 5.6.

5.3.6. Model podataka "korisnička analitika"

Isječak kôda 20: Model podataka "korisnička analitika"

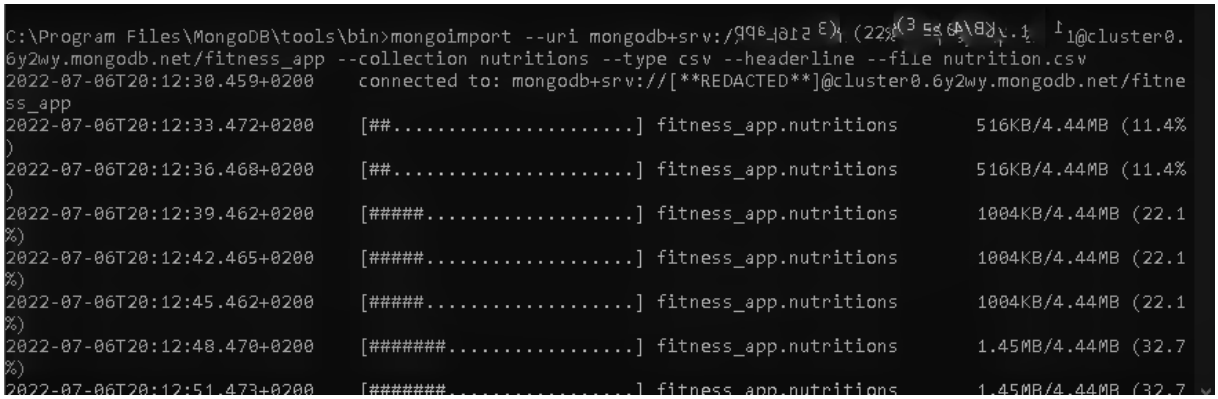
```
{
  "userId": "62e6b100b9b0a936c0413cac",
  "favourite_nutrients": ["62e2fd2a65e45582399274b4",
                          "62e2fd2a65e45582399274a3",
                          "62e2fd2a65e45582399274bf",
  ],

  "bmi": "26.04",
  "bmr": "1996.25",
  "health": "Overweight",
  "healthy_bmi_range": "18.5 - 25",
  "ideal_weight": "81.83000000000001",
  "weight_goals": {
    "extreme_weight_gain": {
      "calory": "4094.1875",
      "weight": "1 kg"
    },
    "extreme_weight_loss": {
      "calory": "2094.1875",
      "weight": "1 kg"
    },
    "mild_weight_loss": {
      "calory": "2844.1875",
      "weight": "0.25 kg"
    },
    "mild_weight_gain": {
      "calory": "3344.1875",
      "weight": "0.25 kg"
    },
    "weight_loss": {
      "calory": "2594.1875",
      "weight": "0.50 kg"
    },
    "weight_gain": {
      "calory": "3594.1875",
      "weight": "0.50 kg"
    },
    "maintain_weight": "3094.1875"
  },
  "daily_calory_intake": [
    {
    "_id": "id5568accs31241",
      "date": "1659139200000",
      "calories": "599"
    },
    {
    "_id": "id5568ac7810r5a",
      "date": "1659052800000",
      "calories": "2294"
```

```
    }
  ],
  "workout_plans": [
    {
      "workoutDays": [
        {
          "title": "Monday",
          "rows": [
            {
              "exerciseName": "Incline Bench Press",
              "reps":  "8",
              "setOrder":  "1",
              "weight": "95",
              "workoutName": "Chest"
            }
          ]
        },
        .
        .
        .
      ],
      "_id": "id5568ac78105b6",
      "title": "Pull Up"
    }
  ]
}
```


Model podataka "korisnička analitika" 20 je referentan modelu "korisnik", a referenca između njih je jedinstvani identifikator korisnika, "userId". Podaci o idelanoj kilaži (engl. weight goals), dnevnom unosu kalorija (engl. daily calory intake) i planovima treninga (engl. workout plans) su ugniježđeni unutar modela "korisnička analitika" zato što je prikaz tih podataka uvijek potreban prilikom dohvata korisničke analitike, odnosno oni se gledaju u kontekstu korisničke analitike. Ugnježđivanjem se izbjegava potreba za izvedbom kompleksnih upita prilikom dohvata korisničke analitike. U polju omiljenih nutrijenata (engl. "favourite nutrients") pohranjuju se jedinstveni identifikatori namirnica/nutrijenata koji predstavljaju referencu na model podataka "nutrijent", a zatim se pomoću reference dohvaćaju svi ostali podaci o omiljenim nutrijentima. Optimalni dnevni unosu kalorija, idealna tjelesna težina, indeks tjelesne mase i razina metabolizma (engl. basal metabolic rate (BMR)) dohvaćaju se na temelju popunjenih osnovnih podataka (težina, visina, razina aktivnosti, spol i godine starosti) pomoću Rapid API fitnes kalkulatora. *"RapidAPI je najveće svjetsko API tržište s preko 10.000 dostupnih API-ja."* [38].

5.4. Uvoz podataka

Skupovi podataka o nutrijentima, mišićnim skupinama, vježbama i varijacijama vježbi preuzeti su s Kaggle-a u obliku .csv datoteke te ih je potrebno uvesti u bazu podataka. MongoDB omogućuje vrlo jednostavan uvoz (engl. import) podataka u bazu podataka putem naredbe `mongoimport`. U nastavnku naredbe potrebno je navesti putanju do klastera, korisničko ime i lozinku, ime baze podataka, ime kolekcije te tip i naziv datoteke koja se uvozi. Na slici 18 prikazan je uvoz podataka o nutrijentima iz datoteke nutrition.csv". Prilikom kreiranja svakog dokumenta, MongoDB automatski kreira polje `_id` koje predstavlja jedinstveni identifikator tog dokumenta.



Slika 18: Primjer učitavanja .csv datoteke u MongoDB Atlas pomoću Command Prompt-a

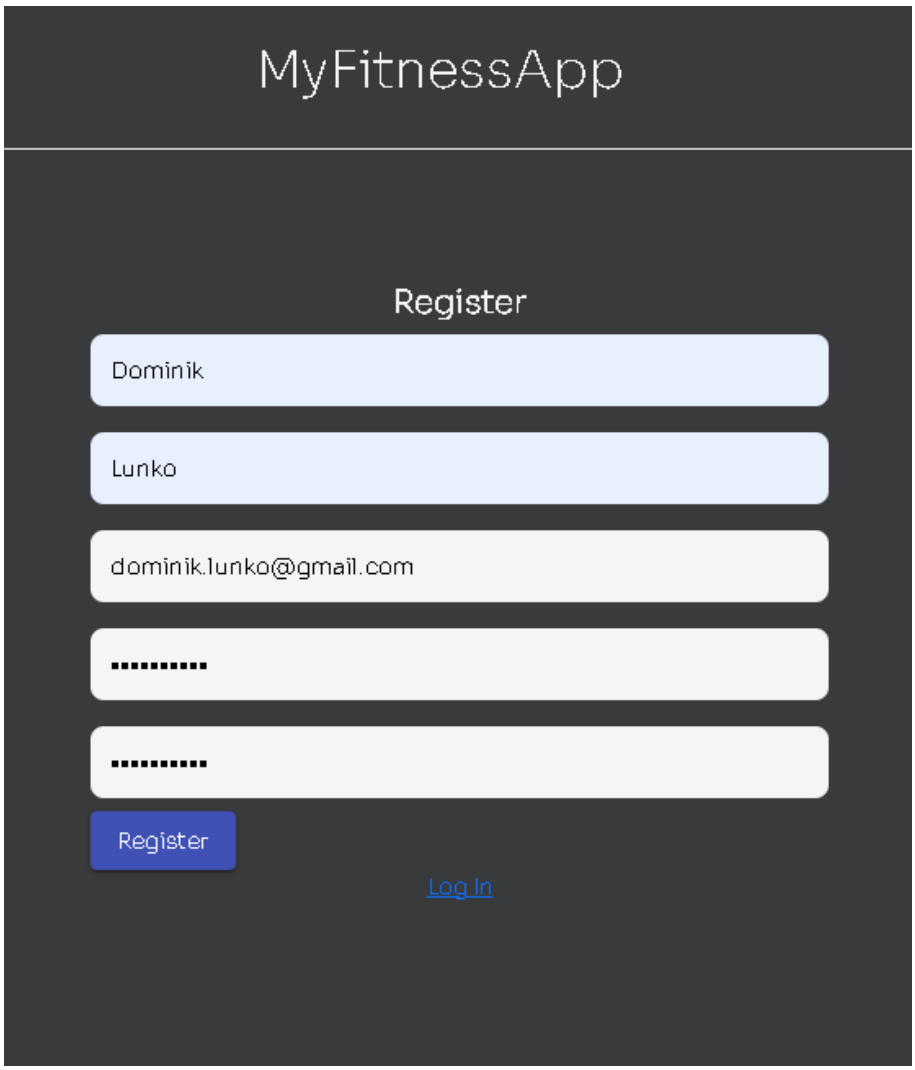
U radu sa Firebase-om taj postupak je malo zahtjevniji zato što izvoz i uvoz podataka u Firestore nije besplatan. Također, Firestore ima postavljenje dnevne limite za čitanje i pisanje podataka u bazu te zato nisu uvezeni cijeli skupovi podataka već samo manji dijelovi (za razliku od MongoDB-a gdje su uvezeni cijeli skupovi sa nekoliko tisuća zapisa). Isječak koda 21 prikazuje dva načina punjenja Firestore-a pomoću vlastite skripte napisane u JavaScript-u. Prvi način pomoću metode `addDoc()` zapisuje zasebno svaki element (JSON objekt) iz liste `data` u kolekciju koja je navedena u metodi `collection()`, a drugi način prikazuje zapisivanje serije (engl. batch) dokumenata u kolekciju. Pozivom metode `writeBatch()` kreira se instanca objekta "WriteBatch" koja pruža metode za kreiranje serijskog zapisa dokumenata u kolekciju [39], a pisanje se izvršava pozivom metode `commit()`. Također je bitno naglasiti da Firestore prilikom kreiranja novog dokumenta ne stvara dodano specifično polje unutar dokumenta kao MongoDB (polje `_id`), već kreira referencu dokumenta (engl. DocumentReference) koji sadrži identifikator dokumenta (id) unutar kolekcije. Slika 20 prikazuje "id" dokumenta.

Isječak kôda 21: Primjer dodavanja dokumenata u Firestore kolekciju

```
 // Zasebno zapisivanje svakog dokumenta u kolekciju
{
  let data = [{...}, {...}, {...}]

import { initializeApp } from "firebase/app";
import { getFirestore, collection, query, where, getDocs, addDoc } from "firebase/
 firestore";

import firebaseConfig from "./config.js";

const firebaseApp = initializeApp(firebaseConfig);
const db = getFirestore(firebaseApp);

const collectionRef = collection(db, "nutrients");

data.forEach(item => {
  addDoc(collectionRef, item)
  .then(collectionRef => {
      console.log("Document added with ID: ", docRef.id);
  })
  .catch(error => {
      console.log(error);
  })
})
}

 // Zapisivanje više dokumenata u kolekciju odjednom
const batch = writeBatch(db);
data.forEach((item) => {
  let docRef = doc(collection(db, "nutrients"));
  batch.set(docRef, item);
});

batch.commit()
```

5.5. Autentifikacija

Autentifikacija je vrlo bitna stavka svake aplikacije jer upravo ona određuje koje radnje neautentificirani korisnik može, odnosno ne može obavljati. U ovom slučaju neautentificirani korisnik ima samo mogućnost pregleda vježbi prema mišićnoj skupini i pretraživanje nutrijenata, dok autentificirani korisnik uz to ima mogućnost pregleda korisničke analitike, dodavanje nutrijenata u favorite, praćenje dnevnog unosa kalorija i izradu vlastitih planova treninga. Na slici 19 prikazan je ekran za registraciju korisnika.



Slika 19: Ekran za registraciju korisnika

U implementaciji aplikacije s MongoDB-om napravljena je vlastita autentifikacija uz pomoć "bcryptjs" i "jsonwebtoken" biblioteka. Biblioteka "bcryptjs" omogućuje pohranjivanje kriptiranih lozinki, dok "jsonwebtoken" omogućuje dijeljenje sigurnosnih informacija između klijenta i poslužitelja [40]. Na sljedećem isječku koda 22 prikazan je dio metode za registraciju korisnika.

Isječak kôda 22: Metoda za registraciju korisnika s MongoDB-om

```
const existingUser = await User.findOne({ email });

if (existingUser)
  return res.status(200).json({
    message: "User already exists",
    success: false,
  });

if (password !== confirmPassword)
  return res.status(200).json({
    message: "Passwords dont match.",
    success: false,
  });

const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);

const result = await User.create({
  email,
  password: hashedPassword,
  name: `${firstName} ${lastName}`,
});
const newAnalytics = new User_analytics({ userId: result._id });
await newAnalytics.save();

const token = jwt.sign({ _id: result._id }, "test", { expiresIn: "1h" });
```

Prilikom registracije se provjerava postoji li već korisnik s tom adresom u bazi podataka. Ukoliko ne postoji lozinka se kriptira i podaci o korisniku se pohranjuju u bazu podataka pomoću metode `create()` koja kao parametar prima JSON objekt. Također, prilikom registracije korisnika, kreira se novi zapis u kolekciji "user_analytics", a u polje "userId" se postavlja vrijednost polja "_id" koje je nastalo kreiranjem dokumenta o novom korisniku u bazu podataka. Polje "userId" je referenca između korisnika i korisničke analitike.

Prijava u sustav 23 radi na način da se na temelju email adrese pretražuje baza podataka metodom `findOne()`. Metoda `findOne()` može pretraživati kolekciju baze podataka na temelju bilo kojeg navedenog polja dokumenta. Ukoliko baza podataka vrati rezultat, tj. ukoliko korisnik s navedenim email-om postoji u bazi podataka, izvršava se provjera ispravnosti lozinke koju je korinisk unio.

Isječak kôda 23: Prijava u aplikaciju s MongoDB-om

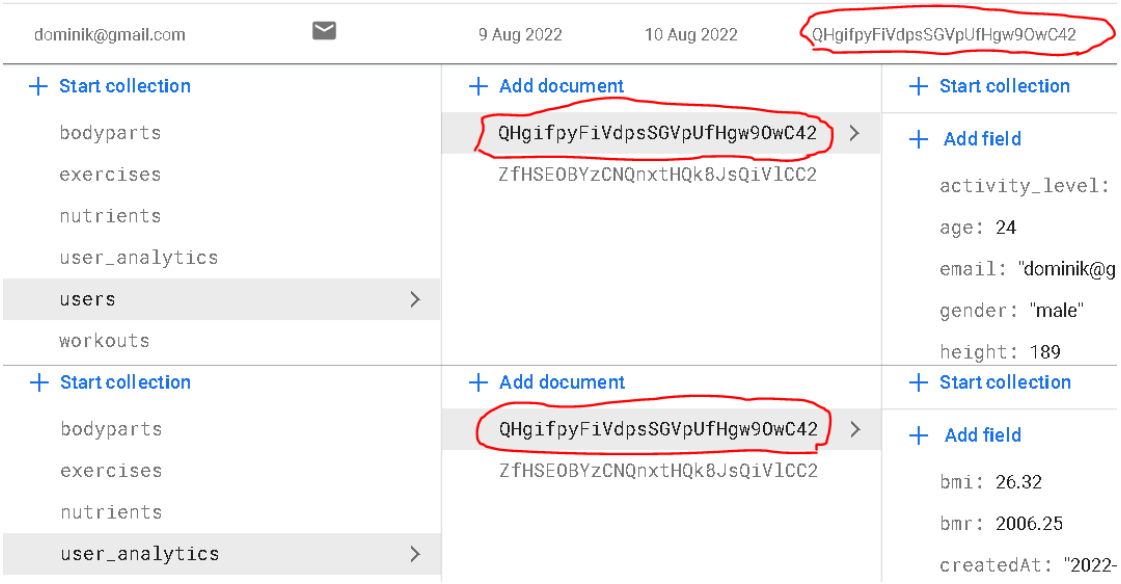
```
const { email, password } = req.body;

const existingUser = await User.findOne({ email });

if (!existingUser)
 return res.status(200).json({...});

const isPasswordCorrect = await bcrypt.compare(
    password,
    existingUser.password
    );

if (!isPasswordCorrect)
    return res.status(200).json({...});

const token = jwt.sign({ _id: existingUser._id }, "test", {
      expiresIn: "1h",
});
```

Implementacija autentifikacije s Firebase-om relativno je jednostavna zato što Firebase nudi svoju gotovu autentifikaciju. Prije same implementacije potrebno je u postavkama projekta omogućiti željeni način autentifikacije. Za potrebe ove aplikacije korištena je autentifikacija pomoću email-a i lozinke. Isječak koda 24 prikazuje implementaciju metode za registraciju korisnika.

Isječak kôda 24: Registracija s Firestore-om

```
createUserWithEmailAndPassword(auth, email, password)
    .then(async (result) => {
      let user = {
        name: firstName + " " + lastName,
        email: email
      };
      await setDoc(doc(db, "users", result.user.uid), {
        ...user,
      })
        .then(async () => {
          await setDoc(doc(db, "user_analytics", result.user.uid), {
            createdAt: new Date()
          }).then(() => {
            ...
        })
        .catch((error) => {
          ...
        });
    })
```

Metoda `createUserWithEmailAndPassword()` kao argumente prima FirebaseAuth instancu, email i lozinku, a kao rezultat vraća objekt u kojem se nalazi jedinstveni indetifikator "user.uid". Ukoliko su email i lozinka validni, metoda `setDoc()` kreira nove dokumente u kolekciji "users", odnosno u kolekciji "user_analytics". Metoda `setDoc()` kao treći argument prima vrijednost koju postavlja kao "id" novokreiranog dokumenta. U ovom slučaju novokreirani dokument u kolekciji "users" i novokreirani dokument u kolekciji "user_analytics" imat će vrijednost "id" dokumenta jednak kao "user.uid". Na slici 20 prikazani su "id"-evi novokreiranih dokumenata.



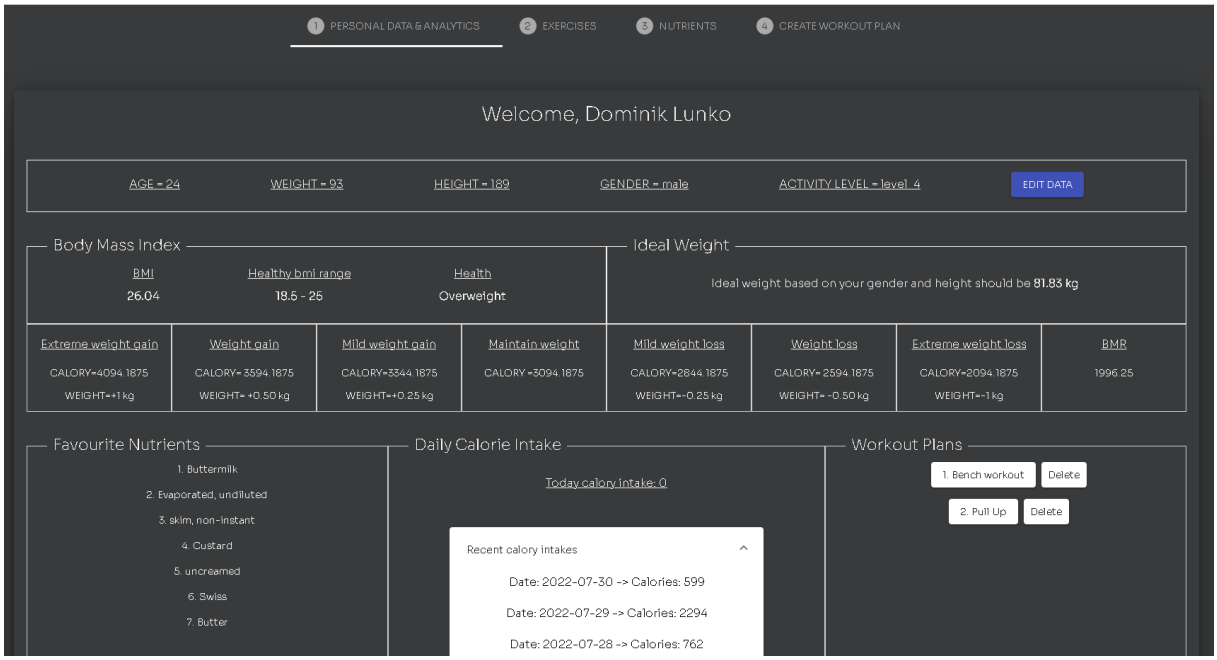
Slika 20: Prikaz id dokumenta

Metoda `signInWithEmailAndPassword()` kao argumente također prima FirebaseAuth instacnu te email i lozinku. Ukoliko su email i lozinka ispravni, metoda kao rezultat vraća objekt u kojem se nalazi jedinstvani identifikator "user.uid". Pomoću tog identifikatora i metode `doc()` dohvaća se referenca dokuemnta. Nakon toga se poziva metoda `getDoc()` koja kao agrument prima referencu dokumenta, a kao rezultat vraća snimak (engl. snapshot) dokumenta. Pozivom metode `data()` nad snimkom dokumenta, ekstrahiraju se podaci pohranjeni u dokumentu. Na sljedećem primjeru 5.5 prikazana je metoda prijave u aplikaciju.

```
signInWithEmailAndPassword(auth, email, password)
.then((result) => {
  const userDocRef = doc(db, "users", `${result.user.uid}`);
  const userDocSnap = await getDoc(userDocRef);
  if (userDocSnap.exists()) {
    res.status(200).json({
      ...
      user: userDocSnap.data(),
    })
  } else {
      ...
  }})
```

5.6. Pregled korisničke analitike

Odabirom na opciju "Personal data & Analytics" dohvaćaju se svi podaci o korisniku iz kolekcije "user_analytics", a osim toga dohvaćaju se i podaci o nutrijentima na temelju liste omiljenih nutrijenata. Opći podaci o korisniku kao što su: ime i prezime, godine, razina aktivnosti, spol, težina i visina dohvaćaju se prilikom prijave korisnika u sustav. Slika 21 prikazuje ekran za prikaz korisničkih podataka.



Slika 21: Prikaz korisničkih podataka

U implementaciji s MongoDB-om dohvaćanje korisničkih podataka 25 izvršava se pomoću agregacije sa klauzulama ili fazama: $match, $lookup i $project. Klauzula $match koristi se za dohvaćanje točno određenog dokumenta iz baze podataka na temelju polja "userId". $lookup klauzula izvršava lijevo vanjsko spajanje (engl. left outer join) [41] na način da dodaje novo polje u svaki ulazni dokument. Novo polje moguće je nazvati po želji (ovom slučaju "joined_favourite_nutrients"), a sadrži podudarne dokumente iz pridružene kolekcije "nutrients". Zadnja klauzula ili faza ove agregacije je klauzula $project unutar koje se uz pomoć biarnih vrijednosti (0 i 1) definiraju polja dokumenta koja će biti vraćena kao rezultat.

Isječak kôda 25: Dohvaćanje korisničke analitike s MongoDB-om

```
const analytics = await User_analytics.aggregate([
  {
    $match: { userId: req.userId },
  },
  {
    $lookup: {
      from: "nutrients",
      localField: "favourite_nutrients",
      foreignField: "_id",
      as: "joined_favourite_nutrients",
```

```
    },
  },
  {
    $project: {
      ideal_weight: 1,
      bmr: 1,
      bmi: 1,
      health: 1,
      daily_calory_intake: 1,
      workout_plans: 1,
      healthy_bmi_range: 1,
      weight_goals: 1,
      createdAt: 1,
      joined_favourite_nutrients: {
        food: 1,
        _id: 1,
      },
    },
}])
```

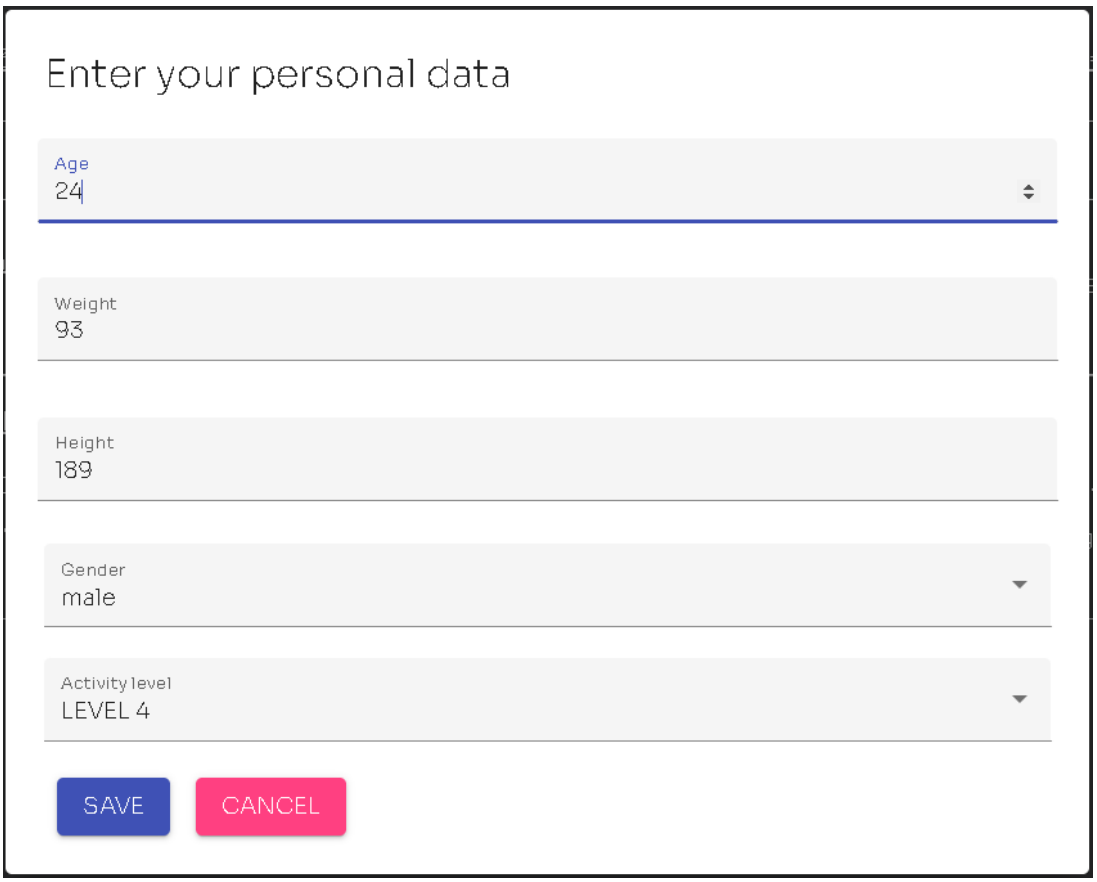
Firebase Firestore ne sadrži metodu za spajanje podataka iz više različitih kolekcija kao MongoDB. Zato je dohvaćanje korisničkih podataka izvršeno na način da se prvo dohvati dokument iz kolekcije "user_analytics" na temelju "id"-a trenutno prijavljenog korisnika, a zatim se nad listom omiljenih nutrijenata poziva metoda `map()` unutar koje se za svaki "id" iz liste dohvaća dokument iz kolekcije nutrijenata 26.

Isječak kôda 26: Dohvaćanje korisničke analitike s Firestore-om

```
const userAnalyticsDocRef = doc(db, "user_analytics", `${currentUserId}`);
  await getDoc(userAnalyticsDocRef)
    .then(async (userAnalyticsSnap) => {
      if (userAnalyticsSnap.exists()) {
        nutrientDocs = await Promise.all(userAnalyticsSnap.data().
         favourite_nutrients
          .map(nutrientId => getDoc(doc(db, "nutrients", `${nutrientId}`)).then(
           nutrientDoc => {
            return nutrientDoc.data();
          })))
      }
      ...
    })
    .catch((error) => {
      ...
    });
```

5.7. Ažuriranje korisničkih podataka

Odabirom opcije "promjena podataka" (engl. edit data), otvara se skočni prozor sa općim podacima o korisniku. Prilikom odabira opcije "spremi", poziva se Rapid API koji na temelju unešenih podataka računa postotak tjelesne masti, idealnu kilažu, potreban unos kalorija, itd. Nakon dohvaćanja izračuna, podaci se spremaju u bazu podataka. Na slici 22 prikazan je ekran za ažuriranje korisnika.



Slika 22: Prikaz ažuriranja korisnika

Ažuriranje podataka s MongoDB-om, odnosno Mongoose-om se može izvršiti pomoću metode `findByIdAndUpdate()` ili `updateOne()`, ali razlika je u tome što metoda `findByIdAndUpdate()` kao prvi argument prima isključivo automatski generirano polje "_id", dok metoda `updateOne()` može pretraživati dokumente iz kolekcije na temelju bilo kojeg polja dokumenta koje sadrži jedinstvenu vrijednost. Obje metode kao drugi argument primaju objekt s podacima za ažuriranje dokumenta. Isječak koda 27 prikazuje spremanje korisničkih podataka s MongoDB-om .

Isječak kôda 27: Spremanje korisničkih podataka s MongoDB-om

```
1
2  // spremanje općenitih korisničkih podataka
3  try {
4    if (!mongoose.Types.ObjectId.isValid(userId)) {
5      ...
6    }
7    const updatedUser = await User.findByIdAndUpdate(userId, user, {
8      new: true,
9    });
10 } catch (err) {
11 }
12
13
14 // spremanje podataka o korisničkoj analitici
15 user_analytics.userId = req.userId;
16 try {
17   const updatedAnalytics = await User_analytics.updateOne(
18     { userId: req.userId },
19     user_analytics
20   );
21   ...
22 } catch (err) {
23   ...
24 }
```

Spremanje korisničkih podataka s Firestore-om izvodi se na način prikazan u isječku koda 28. Za ažuriranje podataka korištena je metoda `updateDoc()` koja će se za razliku od `setDoc()` izvršiti uspješno samo ukoliko dokument prethodno postoji u kolekciji baze podataka.

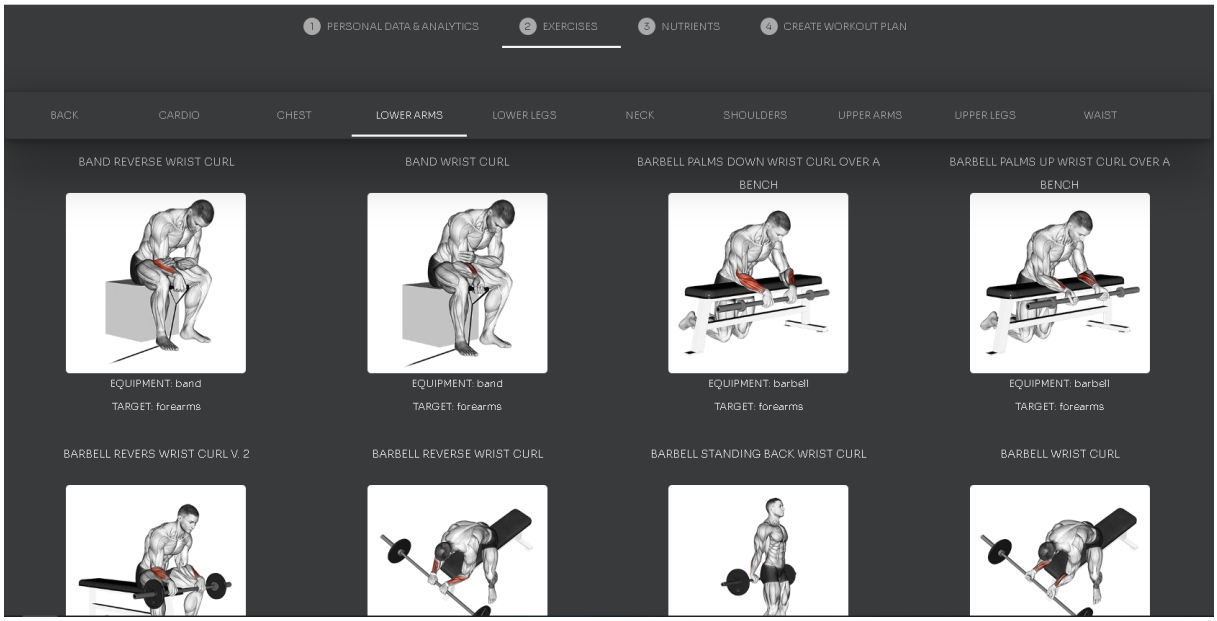
Isječak kôda 28: Spremanje korisničkih podataka s Firestore-om

```
1
2  // spremanje općenitih korisničkih podataka
3  const currentUserId = auth.currentUser.uid;
4  const userRef = doc(db, "users", `${currentUserId}`);
5
6  await updateDoc(userRef, {
7    age: user.age,
8    weight: user.weight,
9    height: user.height,
10   gender: user.gender,
11   activity_level: user.activity_level,
12 })
13   .then(() => {
14   })
15   .catch((error) => {
16   });
17
18
19 // spremanje podataka o korisničkoj analitici
20 let user_analytics = req.body;
21 let currentUserId = auth.currentUser.uid;
```

```
22
23  const userAnalyticsRef = doc(db, "user_analytics", `${currentUserId}`);
24
25  await updateDoc(userAnalyticsRef, {
26    ...user_analytics,
27  })
28    .then(() => {
29    })
30    .catch((error) => {
31    });
```

5.8. Pregled vježbi prema mišićnoj skupini

Odabirom na opciju "vježbe" (engl. exercises) otvara se ekran s popisom svih mišićnih skupina te popisom vježbi ovisno o odabranoj mišićnoj skupini. Prilikom otvaranja ekrana dohvaćaju se mišićne skupine te popis vježbi za prvu mišićnu skupinu u listi. Na slici 23 prikazan je ekran za pregled vježbi prema mišićnoj skupini.



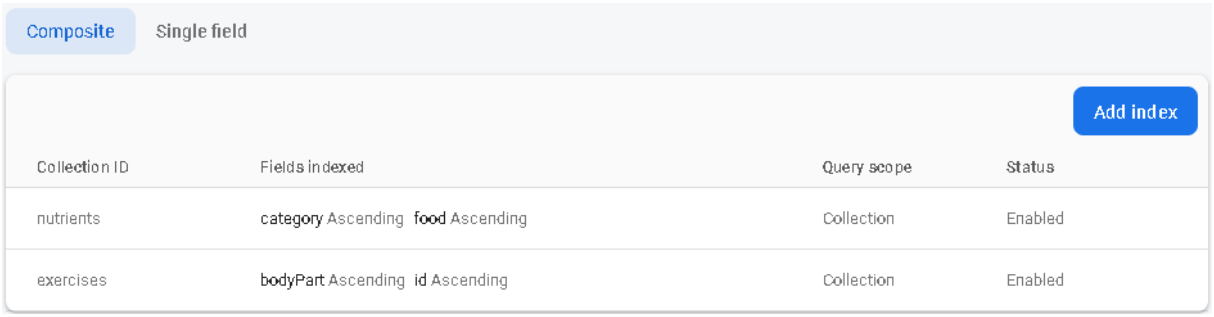
Slika 23: Pregled vježbi prema odabranoj mišićnoj skupini

Svaka mišićna skupina sadrži velik broj vježbi te nema smisla odjednom dohvaćati sve vježbe za neku mišićnu skupinu. Iz tog razloga potrebno je implementirati paginaciju pomoću koje će se dohvaćati određen broj zapisa. S MongoDB-om paginacija je implementirana pomoću agregacije s klauzulom ili fazom `$facet`. `$facet` obrađuje više cjevovoda (engl. pipeline) agregacije unutar jedne faze na istom skupu ulaznih dokumenata [42]. Za svaki cjevovod kreira se novo polje u izlaznom dokumentu (u ovom slučaju polje "metadata" i polje "exercisesList"). Za izradu paginacije potrebne su i klauzule `$skip` pomoću koje se određuje broj dokumenata koje je potrebno "preskočiti" te klauzula `$limit` koja ograničava broj vraćenih dokumenata. Isječak koda 29 prikazuje implementaciju dohvata vježbi s paginacijom.

Isječak kôda 29: Dohvaćanje popisa vježbi s MongoDB-om

```
1
2  try {
3    const exercisesData = await Exercise.aggregate([
4      {
5        $facet: {
6          metadata: [
7            { $match: { bodyPart: { $eq: bodyPart } } },
8            { $count: "total" },
9          ],
10
11         exercisesList: [
12           { $match: { bodyPart: { $eq: bodyPart } } },
13           { $skip: skip * 8 },
14           { $limit: 8 },
15           {
16             $project: {
17               bodyPart: 1,
18               equipment: 1,
19               gifUrl: 1,
20               name: 1,
21               target: 1,
22             },
23           },
24         ],
25       },
26     },
27     { $sort: { name: 1 } },
28   ]);
29 } catch (error) {
30   ...
31 }
```

Implementacija paginacije pomoću Firestore-a 30 napravljena je na način da se kreiraju dva upita, jedan koji će dohvatiti ukupan broj vježbi prema mišićnoj skupini, a drugi u kojem se koriste klauzule `limit`, `orderBy` i `startAfter`. Klauzula `startAfter` mora se korisiti u kombinaciji s `orderBy` iz razloga što `startAfter` kao argument prima vrijednost polja određenog klauzulom `orderby`. Ukoliko se koristi kaluzula `orderBy` u kombinaciji s klauzulom `where`, potrebno je kreirati složeni indeks nad bazom. Slika 24 prikazuje složeni indeks.



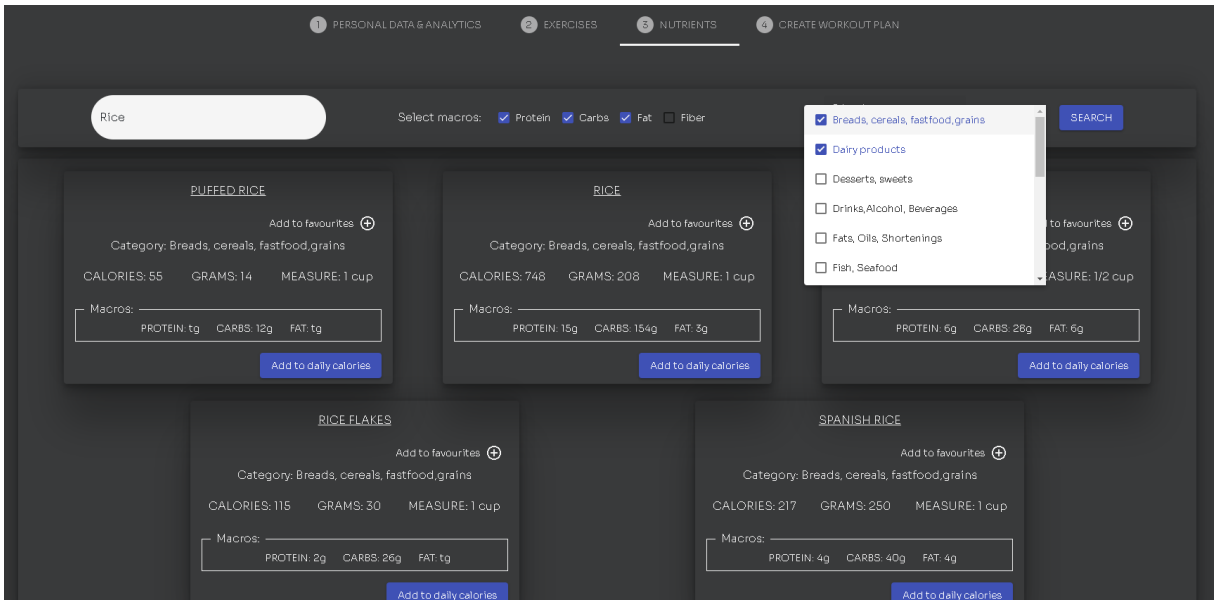
Slika 24: Primjer složenog indeksa

Isječak kôda 30: Dohvaćanje popisa vježbi s Firestore-om

```
let exercises = [];
let queryCount = 0;
const countQuery = query(collection(db, "exercises")
    ,where("bodyPart", "==", bodyPart),orderBy("id"));
await getDocs(countQuery).then((querySnap) => {
  queryCount = querySnap.docs.length;
});
let q = query(collection(db, "exercises")
    ,where("bodyPart", "==", bodyPart),orderBy("id"),limit(8));
if (lastItemId) {
  q = query(q, startAfter(lastItemId));
}
await getDocs(q)
  .then((querySnapshot) => {
    querySnapshot.forEach((doc) => {
      exercises.push(doc.data());
    });
  })
  .catch((error) => {
  });
```

5.9. Pretraživanje namirnica/nutrijenata

Odabirom na opciju "nutrients" otvara se ekran za pretragu namirnica/nutrijenata. Prilikom otvaranja poziva se metoda koja dohvaća kategorije nutrijenata 31. Korisnik ima mogućnost pretrage prema nazivu i prema kateogriji nutrijenta. Također, korisnik može odabrati samo one makronutrijente koje je potrebno prikazati na ekranu, na primjer proteine, masti i ugljikohidrate. Osim pretrage nutrijenata, autentificirani korisnici imaju mogućnost dodavanja nutrijenata u dnevni unos kalorija te dodavanja na listu omiljenih nutrijenata. Na slici 25 prikazan je ekran za pretragu nutrijenata.



Slika 25: Prikaz ekrana za pretragu nutrijenata

Isječak kôda 31: Dohvaćanje kategorija nutrijenata

```
// Dohvat kategorija s MongoDB-om
const categories = await Nutrient.find().distinct("category");

// Dohvat kategorija s Firebase-om
const q = query(collection(db, "nutrients"));
await getDocs(q)
  .then((querySnapshot) => {
    querySnapshot.forEach((doc) => {
      categories.push(doc.data().category);
    });
    categories = [...new Set(categories)];
  })
  .catch((error) => {
    ...
  });
```

Dohvaćanje željenih nutrijenata pomoću MongoDB-a 32 izvršava se pomoću agregacije. Kao i kod dohvata vježbi, koriste se klauzule `$facet`, `$skip` i `$limit` u svrhu postizanja paginacije. Zbog funkcionalnosti odabira samo određenih makronutrijenata, potrebna je uvjetovana projekcija koja se postiže klauzulom `$cond`.

Isječak kôda 32: Dohvaćanje nutrijenata s MongoDB-om

```
const foundFoods = await Nutrient.aggregate([
  {
    $facet: {
      metadata: [
        {
          $match: {
            $and: [
              { food: { $regex: regex } },
              { category: { $exists: true, $in: categories } },
            ],
          },
        },
        { $count: "total" },
      ],
      nutritionList: [
        {
          $match: {
            $and: [
              { food: { $regex: regex } },
              { category: { $exists: true, $in: categories } },
            ],
          },
        },
        { $skip: skip * 12 },
        { $limit: 12 },
        {
          $project: {
            protein: {
              $cond: {
                if: { $eq: [false, macros.protein] },
                then: "$$REMOVE",
                else: "$protein",
              },
            },
            carbs: {
              $cond: {
                if: { $eq: [false, macros.carbohydrate] },
                then: "$$REMOVE",
                else: "$carbs",
              },
            },
            fat: {
              $cond: {
                if: { $eq: [false, macros.fat] },
                then: "$$REMOVE",
                else: "$fat",
              },
            },
            fiber: {
              $cond: {
                if: { $eq: [false, macros.fiber] },
                then: "$$REMOVE",
```

```
                  else: "$fiber",
                },
              },
              calories: 1,
              food: 1,
              grams: 1,
              measure: 1,
              category: 1,
            },
          },
        ],
      },
    },
]);
```

Kod dohvaćanja nutrijenata s Firebase-om paginacija se postiže na isti način kao kod dohvaćanja vježbi. Pretraga nutrijenata po imenu postiže se klauzulama `startAt` i `endAt` u kombinaciji sa klauzulom `orderBy`. Firestore ne pruža mogućnost odabira (engl. select) samo određenih polja, već se dohvaćaju sva polja dokumenta. Zato je potrebno "ručno" izbaciti polja koja nisu potrebna 33.

Isječak kôda 33: Dohvaćanje nutrijenata s Firestore-om

```
let nutrientList = [];
let queryCount = 0;
const countQuery = query(
  collection(db, "nutrients"),
  where("category", "in", categories),
  orderBy("food"),
  startAt(foodName),
  endAt(foodName + "\uf8ff")
);
await getDocs(countQuery).then((querySnap) => {
  queryCount = querySnap.docs.length;
});
let q = query(
  collection(db, "nutrients"),
  where("category", "in", categories),
  orderBy("food"),
  startAt(foodName),
  endAt(foodName + "\uf8ff"),
  limit(12)
);
if (lastFoodName) {
  q = query(q, startAfter(lastFoodName));
}
await getDocs(q)
  .then((querySnapshot) => {
    querySnapshot.forEach((doc) => {
      let newObj = {
        _id: doc.id,
        protein: macros.protein ? doc.data().protein : null,
        carbs: macros.carbs ? doc.data().carbs : null,
```

```
31          fat: macros.fat ? doc.data().fat : null,
32          fiber: macros.fiber ? doc.data().fiber : null,
33          calories: doc.data().calories,
34          category: doc.data().category,
35          food: doc.data().food,
36          grams: doc.data().grams,
37          measure: doc.data().measure,
38        };
39        Object.keys(newObj).filter((key) => {
40          newObj[key] == null ? delete newObj[key] : key;
41        });
42        nutrientList.push(newObj);
43      });
44    })
45    .catch((error) => {
46    });
```

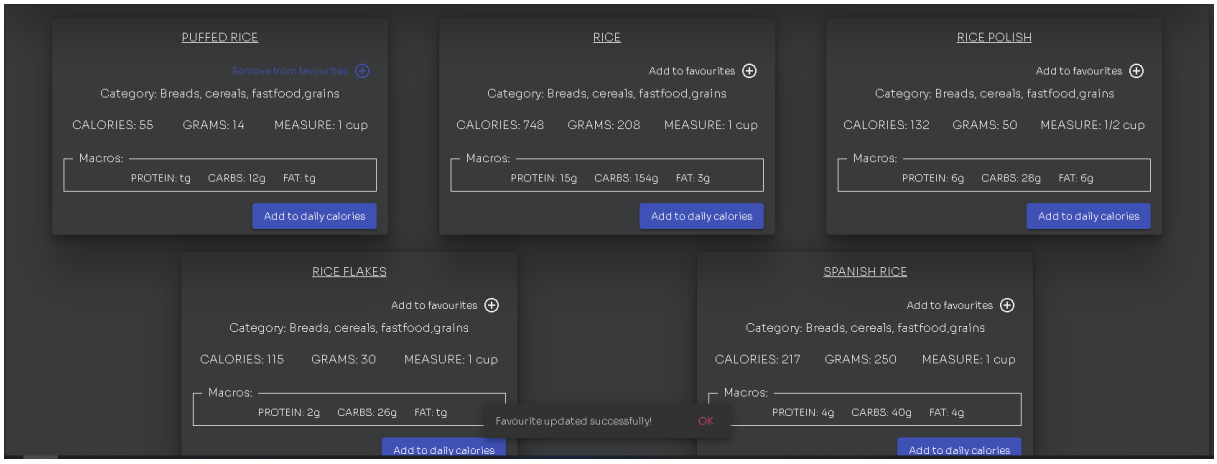
Firebase-admin omogućuje odabir samo određenih polja uz pomoć metode `select()`. Na primjeru 34 prikazano je korištenje Firebase-admin select() metode.

Isječak kôda 34: Primjer korištenja Firebase-admin select metode

```
1  adminFirestore
2    .collection("nutrients")
3    .select("category")
4    .get()
5    .then((querysnapshot) => {
6      ...
7      });
8      ...
9    });
```

5.9.1. Dodavanje nutrijenata na listu favorita

Odabirom na opciju "dodaj u favorite" (engl. add to favourite), "id" odabranog nutrijenta se dodaje na listu omiljenih nutrijenata (favorita). Ova funkcionalnost vidljiva je samo autentificiranim korisnicima. Na slici 26 prikazano je dodavanje u favorite.



Slika 26: Prikaz dodavanja nutrienta na listu omiljenih nutrijenata

Dodavanje nutrijenata u favorite s MongoDB-om 35 implementirano je na način da se provjerava postoji li id nutrijenta u listi favorita. Ukoliko postoji, potrebno ga je obrisati, u suprotnom potrebno ga je dodati na listu favorita.

Isječak kôda 35: Dodavanje nutrijenata u favorite s MongoDB-om

```
const userAnalytics = await User_analytics.find({ userId: req.userId });

let favouriteNutrients = [];

if (userAnalytics[0].favourite_nutrients.length > 0) {
  favouriteNutrients = userAnalytics[0].favourite_nutrients;
}
const index = favouriteNutrients.indexOf(mongoose.Types.ObjectId(nutrientId));

if (index == -1) {
  favouriteNutrients.push(nutrientId);
} else {
  favouriteNutrients = favouriteNutrients.filter(
    (id) => String(id) != nutrientId
  );
}
const updatedUserAnalytics = await User_analytics.updateOne(
  { userId: req.userId },
  { favourite_nutrients: favouriteNutrients }
);
```

Prilikom implementacije funkcionalnosti dodavanja u favorite s Firebase-om 36, korištena je metoda `runTransaction()`. Firebase transakcije služe za grupiranje više operacija, a korisne su prilikom ažuriranja vrijednosti polja na temelju trenutne vrijednosti ili vrijednosti nekog drugog polja [43]. Kao i kod implementacije s MongoDB-om, provjerava se postoji li trenutni id nutrienta u listi favorita i na temelju toga se id nutrienta briše, odnosno dodaje u listu.

Isječak kôda 36: Dodavanje nutrijenata u favorite s Firestore-om

```
const currentUserId = auth.currentUser.uid;
const { nutrientId } = req.params;
try {
  const userAnalyticsRef = doc(db, "user_analytics", `${currentUserId}`);
  await runTransaction(db, async (transaction) => {
    const userAnalyticsSnap = await transaction.get(userAnalyticsRef);
    if (!userAnalyticsSnap.exists()) {
      throw "Document does not exist!";
    }
    let newFavouriteNutritions = userAnalyticsSnap.data().favourite_nutrients;
    const index = newFavouriteNutritions.indexOf(nutrientId);

    if (index == -1) {
      newFavouriteNutritions.push(nutrientId);
    } else {
      newFavouriteNutritions = newFavouriteNutritions.filter(
        (id) => id != nutrientId
      );
    }
    transaction.update(userAnalyticsRef, {
      favourite_nutrients: newFavouriteNutritions,
    });
  });
} catch (error) {
}
```

5.9.2. Ažuriranje dnevnog unosa kalorija

Prilikom pretraživanja nutrijenata, autentificirani korisnici imaju mogućnost dodavanja nutrijenata u dnevni unos kalorija kako bi jednostavnije mogli pratiti koliko kalorija su unijeli toga dana. Na slici 21 vidi se dnevni kalorijski unos korisnika. Implementacija funkcionlanosti za ažuriranje dnevnog unosa kalorija 37 gotovo je identična implementaciji dodavanja u favorite. Razlika je u tome što se uspoređuje trenutni datum sa datumima pohranjenim u ugniježđenom dokumentu "daily_calory_intake" i na temelju toga se zbraja dnevni unos kalorija.

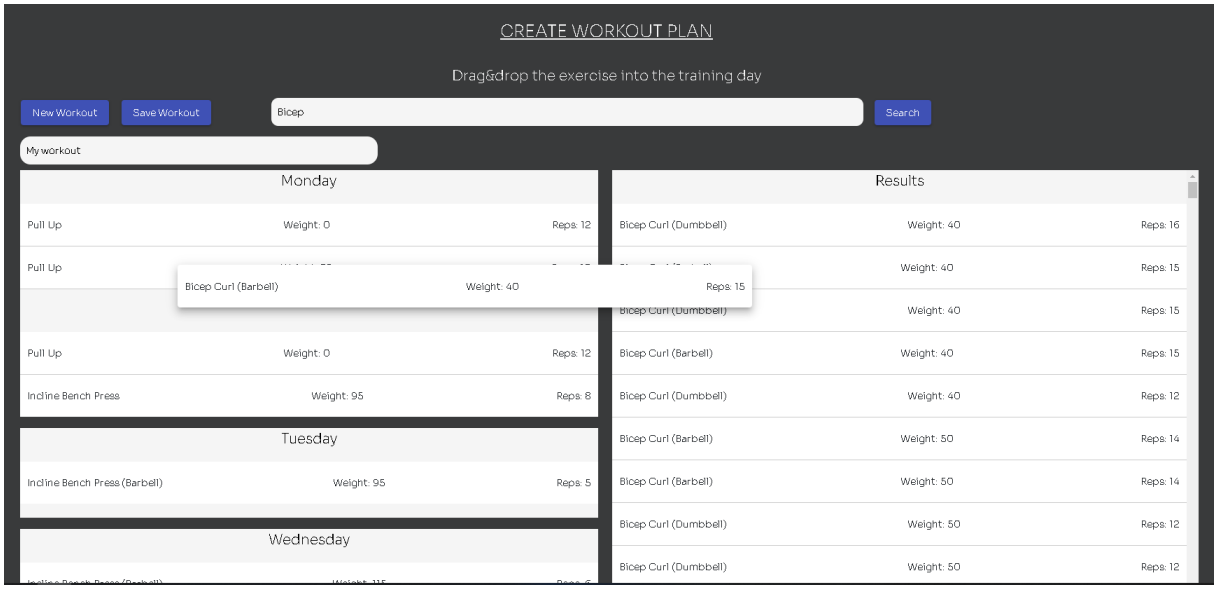
Isječak kôda 37: Ažuriranje dnevnog unosa kalorija

```

// MongoDB implementacija
const analytics = await User_analytics.find({ userId: req.userId });
let dailyCaloryIntake = [];
if (analytics[0].daily_calory_intake.length > 0) {
  dailyCaloryIntake = analytics[0].daily_calory_intake;
}
let todayDailyCalories = dailyCaloryIntake.find(
  (item) => item.date.toISOString().split("T")[0] == todayDate.split("T")[0]
);
if (todayDailyCalories) {
  todayDailyCalories.calories = todayDailyCalories.calories + calories;
} else {
  dailyCaloryIntake.push({
    date: todayDate.split("T")[0],
    calories: calories,
  });
}
const updatedUserAnalytics = await User_analytics.updateOne(
  { userId: req.userId },
  { daily_calory_intake: dailyCaloryIntake }
);

// Firebase implementacija
try {
  const userAnalyticsRef = doc(db, "user_analytics", `${currentUserId}`);
  await runTransaction(db, async (transaction) => {
    const userAnalyticsSnap = await transaction.get(userAnalyticsRef);
    if (!userAnalyticsSnap.exists()) {
      throw "Document does not exist!";
    }
    let dailyIntakeArray = userAnalyticsSnap.data().daily_calory_intake;

    let todayDailyCalories = dailyIntakeArray.find(
      (item) => item.date == todayDate.split("T")[0]
    );
    if (todayDailyCalories) {
      todayDailyCalories.calories = todayDailyCalories.calories + calories;
    } else {
      dailyIntakeArray.push({
        date: todayDate.split("T")[0],
```

```
          calories: calories,
        });
      }
      transaction.update(userAnalyticsRef, {
        daily_calory_intake: dailyIntakeArray,
      });
    });
} catch (error) {
}
```

5.10. Izrada vlastitog plana treninga

Kvalitetan plan treninga vrlo je važna stavka svakog vježbača, stoga ova aplikacija omogućava autentificiranim korisnicima izradu vlastitih planova treninga. Odabirom opcije "kreiraj plan treninga" (engl. create workout plan) otvara se ekran na kojem korisnik može pretraživati vježbe po nazivu. U bazi podataka za svaku vježbu postoje i različite varijacije vježbe ovisno o broju ponavljanja, kilaži s kojom se vježba izvodi, itd. Nakon dohvaćanja rezultata na temelju pretrage, korisnik povlačenjem i ispuštanjem (engl. drag and drop) dodaje željenu varijaciju vježbe određenom danu u tjednu. Na slici 27 prikazan je ekran za izradu plana treninga. Odabirom postojećeg plana treninga na ekranu korisničke analitike otvara se ekran za izradu planova treninga, ali u tom slučaju se pretpopunjavaju podaci o planu treninga i odabirom opcije "spremi plan" (engl. save workout) izvršava se ažuriranje postojećeg plana treninga.



Slika 27: Prikaz ekrana za izradu plana treninga

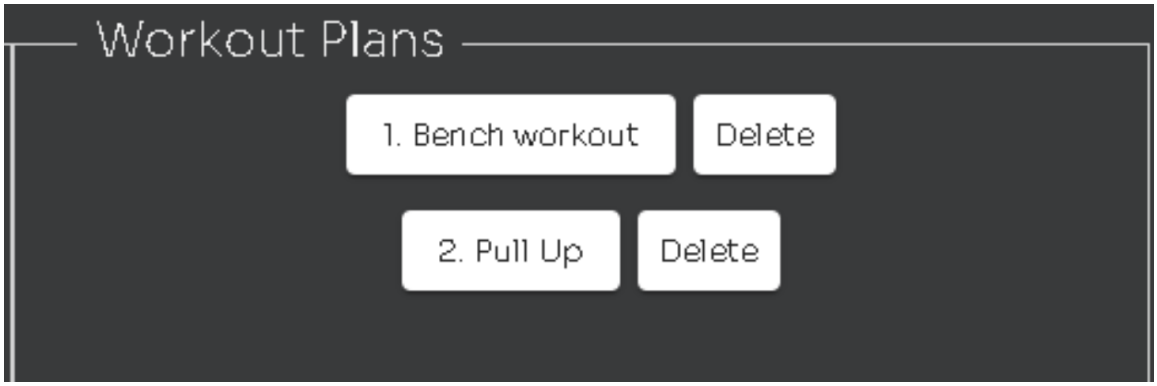
Implementacija izrade plana treninga radi na način da se dohvaća lista planova treninga, a zatim se provjerava postoji li već plan treninga s proslijeđenim identifikatorom. Ukoliko postoji, njegova vrijednost se ažurira, a inače se kreira potpuno novi plan treninga.

Isječak kôda 38: Kreiranje/ažuriranje plana treninga

```
// MongoDB implementacija
const { workout } = req.body;
  try {
     const result = await User_analytics.aggregate([
      {$match: {userId: req.userId}},
      {$project: {
        _id: 0,
        workout_plans: 1
      }}
     ])
     let workoutPlans = result[0].workout_plans
     const index = workoutPlans.findIndex(item => item._id == workout._id);
      if (index == -1) {
        workoutPlans.push(workout);
      } else {
        workoutPlans[index] = JSON.parse(JSON.stringify(workout));
      }
     await User_analytics.updateOne(
      { userId: req.userId },
      {workout_plans: workoutPlans}
    );
  } catch (error) {}

// Firebase implementacija
const { workout } = req.body;
try {
  const userAnalyticsRef = doc(db, "user_analytics", `${currentUserId}`);
  await runTransaction(db, async (transaction) => {
    const userAnalyticsSnap = await transaction.get(userAnalyticsRef);
    if (!userAnalyticsSnap.exists()) {
      throw "Document does not exist!";
    }
    let workoutPlans = userAnalyticsSnap.data().workout_plans;

    const index = workoutPlans.findIndex(
      (item) => item._id == workout._id
    );
    if (index == -1) {
      workoutPlans.push(workout);
    } else {
      workoutPlans[index] = JSON.parse(JSON.stringify(workout));
    }
    transaction.update(userAnalyticsRef, {
      workout_plans: workoutPlans,
    })})};
```

5.11. Brisanje plana treninga

Odabirom opcije "obriši" (engl. delete) plan treninga na ekranu korisničke analitike, šalje se id plana treninga kojeg je potrebno izbrisati. Na slici 28 prikazana je opcija za brisanje plana treninga. Brisanje jednog elementa iz liste se s MongoDB-om postiže uz pomoć klauzule `$pull`. Potrebno je samo navesti ime polja i jedinstveni identifikator kojeg je potrebno obrisati 39. S Firebase-om se brisanje jednog elementa iz liste postiže upotrebom transakcije 40.



Slika 28: Prikaz opcije za brisanje plana treninga

Isječak kôda 39: Brisanje plana treninga s MongoDB-om

```
 // MongoDB implementacija
const updatedUserAnalytics = await User_analytics.updateOne(
  { userId: req.userId },
  { $pull: { workout_plans: { _id: workoutId } } }
);
```

Isječak kôda 40: Brisanje plana treninga s Firestore-om

```
 // Firebase implementacija
try {
  const userAnalyticsRef = doc(db, "user_analytics", `${currentUserId}`);
  await runTransaction(db, async (transaction) => {
    const userAnalyticsSnap = await transaction.get(userAnalyticsRef);
    if (!userAnalyticsSnap.exists()) {
      throw "Document does not exist!";
    }
    let workoutPlans = userAnalyticsSnap
      .data()
      .workout_plans.filter((item) => item._id != workoutId);

    transaction.update(userAnalyticsRef, {
      workout_plans: workoutPlans,
    });
  });
} catch (error) {}
```

6. Zaključak

Tema ovog diplomskog rada je bila upotreba dokumentnih baza podataka prilikom izrade web-aplikacije, unutar koje je bio cilj prikazati što je to dokumentna baza podataka, koje su njezine glavne značajke, koje su sličnosti i razlike između dokumentne baze podataka MongoDB i sustava za upravljanje relacijskim bazama podataka MySQL, a na kraju, kao glavni dio rada, opisane su implementacije web-aplikacije s MongoDB-om i Firebase-om, na temelju kojih možemo zaključiti da se dokumentne baze podataka MongoDB i Firebase (Firestore) vrlo lako integriraju s modernim tehnogijama poput NodeJS-a, a rad s njima je također jednostavan zbog načina na koji se dokumenti pohranjuju u bazu podataka (JSON ili BSON format). Kroz implementaciju je prikazano korištenje osnovnih operacija umetanja, čitanja, ažuriranja i brisanja (CRUD), agregacija s MongoDB-om te transakcija s Firestore-om.

Na temelju detaljnog opisa dokumentno-orijentiranih baza podataka, zaključujemo da one postaju sve popularniji izbor baze podataka za aplikacije koje zahtijevaju rad sa velikom količinom nestrukturiranih podataka zahvaljujući svojoj fleksibilnosti, skalabilnosti i jednostavnosti korištenja. Osim toga, visoka horizntalna skalabilnost dokumentno-orijentiranih baza podataka (npr. MongoDB) omogućuje povećanje kapaciteta (broja korisnika i povećanje skupova podataka) i dostupnosti baze podataka po nižoj cijeni, za razliku od sustava za upravljanje relacijskim bazama podataka koji nisu horizontalno skalabilni. Međutim neke aplikacije ne zahtijevaju takva svojstva, već im priotitet mogu biti integirtet, dosljednost i sigurnost podataka, pogotovo u doba kada su kibernetički napadi dio svakidašnjice.

Odabir baze podataka temelji se na zahtjevima aplikacije unutar koje će baza podataka biti korištena. Shodno tome, zaključujemo da ne postoji loša baza podataka, već postoji loš odabir baze podataka. Na primjer, ukoliko aplikacija ne zahtjeva strogu shemu, integirtet i doljednost, već mogućnost rada s velikom količinom podataka u realnom vremenu, logičan potez je odabir dokumentno-orijenitrane baze podataka koja se savršeno uklapa u nadevene zahtijeve aplikacije.

Popis literature

[1] M. Madison, M. Barnhill, C. Napier i J. Godin, „NoSQL Database Technologies," *Journal of International Technology and Information Management*, sv. 24, br. 1, 2015., ISSN: 1941-6679.

[2] MongoDB. „MongoDB Atlas Tutorial." (2022.), adresa: https://www.mongodb.com/basics/mongodb-atlas-tutorial (pogledano 13. 8. 2022.).

[3] Firebase. „Cloud Firestore." (2022.), adresa: https://firebase.google.com/docs/firestore (pogledano 13. 8. 2022.).

[4] Firebase. „DocumentReference." (2022.), adresa: https://firebase.google.com/docs/reference/node/firebase.firestore.DocumentReference (pogledano 13. 8. 2022.).

[5] D. Chinmayee. „What is Angular?: Architecture, Features, and Advantages." (8. 8. 2022.), adresa: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular#what_is_angular (pogledano 13. 8. 2022.).

[6] NodeJS. „Introduction to Node.js." (), adresa: https://nodejs.dev/learn (pogledano 13. 8. 2022.).

[7] Studio by UXPin. „What is NPM (Node Package Manager)?" (), adresa: https://www.uxpin.com/studio/blog/what-is-npm/ (pogledano 13. 8. 2022.).

[8] Simplilearn. „What Is Express JS In Node JS?" (7. 7. 2022.), adresa: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js (pogledano 13. 8. 2022.).

[9] Mongoose. „Elegant mongodb object modeling for node.js." (), adresa: https://mongoosejs.com/ (pogledano 10. 8. 2022.).

[10] U. Çağlar. „What is Kaggle?" (3. 2022.), adresa: https://www.datacamp.com/blog/what-is-kaggle (pogledano 15. 8. 2022.).

[11] K. Karamjit i R. Rinkle, „Modeling and Querying Data in NoSQL Databases," *2013 IEEE International Conference on Big Data*, Patiala, Punjab, India: IEEE Xplore, 2013., ISBN: 978-1-4799-1293-3/13.

[12] Academy 3T. „The MongoDB Basics: Databases, Collections Documents." (2022.), adresa: https://studio3t.com/academy/lessons/mongodb-basics/ (pogledano 16. 8. 2022.).

[13] „Introducing JSON.” (), adresa: https://www.json.org/json-en.html (pogledano 28. 6. 2022.).

[14] L. Peter, L. David i G. Maxine, „XML (Extensible Markup Language),” *TechTarget*, 6. 2015.

[15] MongoDB. „Data Modeling Introduction.” (2022.), adresa: https://www.mongodb.com/docs/manual/core/data-modeling-introduction/ (pogledano 16. 8. 2022.).

[16] V. Harley, B. Wagner, H. Maristela, G. Valeria i H. Fernanda, „Data Modeling for NoSQL Document-Oriented Databases,” *2013 IEEE International Conference on Big Data*, Brasília, Brasil.: SIMBig, 2015.

[17] G. Yulia, „Data Modeling with MongoDB,” *Speaker Deck*, 7. 4. 2021.

[18] MongoDB. „Data Model Design.” (2022.), adresa: https://www.mongodb.com/docs/manual/core/data-model-design/#std-label-data-modeling-embedding (pogledano 16. 7. 2022.).

[19] MongoDB. „Indexes.” (2022.), adresa: https://www.mongodb.com/docs/manual/indexes/ (pogledano 16. 8. 2022.).

[20] MongoDB. „Add Secondary Indexes to Time Series Collections.” (2022.), adresa: https://www.mongodb.com/docs/manual/core/timeseries/timeseries-secondary-index/ (pogledano 18. 8. 2022.).

[21] MongoDB. „Single Field Indexes.” (2022.), adresa: https://www.mongodb.com/docs/manual/core/index-single/ (pogledano 18. 8. 2022.).

[22] MongoDB. „Compound Indexes.” (2022.), adresa: https://www.mongodb.com/docs/manual/core/index-compound/ (pogledano 18. 8. 2022.).

[23] MongoDB. „What is Scaling in MongoDB?” (2022.), adresa: https://www.mongodb.com/basics/scaling (pogledano 16. 8. 2022.).

[24] Developers. „Scaling Horizontally vs. Scaling Vertically.” (24. 7. 2020.), adresa: https://www.section.io/blog/scaling-horizontally-vs-vertically/ (pogledano 22. 8. 2022.).

[25] M. William, „Operational Big Data,” *Information Management*, ScienceDirect, 2014., str. 97–109.

[26] „MongoDB Replication.” (2022.), adresa: https://www.mongodb.com/basics/replication (pogledano 22. 8. 2022.).

[27] C. Abhinav. „Understanding NoSQL Data Replication.” (25. 5. 2022.), adresa: https://hevodata.com/learn/nosql-data-replication/#Methods (pogledano 22. 8. 2022.).

[28] Inapps. „MongoDB vs MySQL – Which is a better database?” (13. 6. 2022.), adresa: https://www.inapps.net/mongodb-vs-mysql-which-is-a-better-database/ (pogledano 18. 8. 2022.).

[29] MongoDB. „MongoDB Use Cases.” (2022.), adresa: https://www.mongodb.com/use-cases (pogledano 16. 8. 2022.).

[30] MongoDB. „MongoDB vs MySQL Differences.” (2022.), adresa: https://www.mongodb.com/compare/mongodb-mysql (pogledano 9. 7. 2022.).

[31] T. David. „MongoDB vs. MySQL: What's the difference?” (29. 8. 2022.), adresa: https://www.guru99.com/mongodb-vs-mysql.html#7 (pogledano 9. 9. 2022.).

[32] T. Ciprian-Octavian, R. Florin, B. Alexandru i B. Ion, „Performance evaluation for CRUD operations in asynchronously replicated document oriented database,” *20th International Conference on Control Systems and Science*, Bucharest, Romania: IEEE Xplore, 2015., ISBN: 978-1-4799-1780-8/15. DOI: DOI10.1109/CSCS.2015.32.

[33] A. Durga, „MongoDB vs MySQL: Which Is the Better Database Management System?” *Kinsta*, 2. 2022.

[34] MySQL. „MySQL Cluster CGE.” (2022.), adresa: https://www.mysql.com/products/cluster/ (pogledano 29. 6. 2022.).

[35] O. Lior, A. Jenny, G.-O. Nurit i G. Yaron, „Security Issues in NoSQL Databases,” *2011 International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11*, Beer-Sheva, Israel, 2011., ISBN: 978-0-7695-4600-1/11. DOI: DOI10.1109/TrustCom.2011.70.

[36] MongoDB. „What is MongoDB Atlas Cluster?” (2022.), adresa: https://www.mongodb.com/basics/clusters (pogledano 22. 8. 2022.).

[37] Firebase. „Add the Firebase Admin SDK to your server.” (2022.), adresa: https://firebase.google.com/docs/admin/setup (pogledano 10. 8. 2022.).

[38] StackShare. „What is RapidAPI?” (2022.), adresa: https://stackshare.io/RapidAPI (pogledano 13. 8. 2022.).

[39] Firebase. „Firebase.Firestore.WriteBatch.” (2022.), adresa: https://firebase.google.com/docs/reference/unity/class/firebase/firestore/write-batch (pogledano 18. 8. 2022.).

[40] JWT. „What is JSON Web Token?” (), adresa: https://jwt.io/introduction (pogledano 10. 8. 2022.).

[41] MongoDB. „Lookup (aggregation).” (2022.), adresa: https://www.mongodb.com/docs/manual/reference/operator/aggregation/lookup/ (pogledano 12. 8. 2022.).

[42] MongoDB. „Facet (aggregation).” (2022.), adresa: https://www.mongodb.com/docs/manual/reference/operator/aggregation/facet/ (pogledano 12. 8. 2022.).

[43] Firebase. „Transactions and batched writes.” (2022.), adresa: https://firebase.google.com/docs/firestore/manage-data/transactions (pogledano 12. 8. 2022.).

Popis slika

Popis tablica

Popis isječaka koda