Korištenje MongoDB u razvoju web aplikacija

Sobotinčić, Ana

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski jednopredmetni studij informatike

Ana Sobotinčić

Korištenje MongoDB u razvoju web aplikacija

Završni rad

Mentor: doc. dr. sc. Lucia Načinović Prskalo

Rijeka, rujan 2021.

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 22.4.2021.

Zadatak za završni rad

Pristupnik: Ana Sobotinčić

Naziv završnog rada: Korištenje MongoDB u razvoju web aplikacija

Naziv završnog rada na eng. jeziku: Using MongoDB in web application development

Sadržaj zadatka: Zadaci završnog rada su dati pregled korištenja baza podataka u web aplikacijama, njihovog razvoja i trenutnog stanja te opisati osnovne karakteristike baza podataka koje se koriste u web aplikacijama. Naglasak je pritom na MongoDB bazi podataka, koja će se koristiti prilikom izrade primjerne web aplikacije. U radu će se također opisati i demonstrirati svi važni elementi i funkcionalnosti izrađene web aplikacije.

Mentor

Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditelj za završne radove

doc. dr. sc. Miran Pobar

Zadatak preuzet: 22.4.2021.

Ana Sobotinčić

(potpis pristupnika)

Sažetak

U ovome završnom radu daje se pregled, i opis baza podataka koje se koriste u razvoju web aplikacija, njihov razvoj kroz povijesti te osvrt na sadašnje stanje i budućnost. Detaljnije su pojašnjena dva osnovna modela baza podataka: relacijski model podataka i nerelacijski model podataka.

U radu je naglasak stavljen na NoSQL bazu podataka MongoDB, koja je trenutno najpopularnija nerelacijska baza podataka u svijetu. Glavni je cilj rada izraditi web aplikaciju koja koristi MongoDB kao bazu podataka te pomoću nje prikazati kako se podaci (dokumenti) u bazi stvaraju, čitaju, uređuju i brišu.

Ključne riječi: web aplikacija, baze podataka, nerelacijske baze podataka, dokument baze podataka, MongoDB, dokument, kolekcija

Sadržaj

1. Uvod

Web aplikacije su aplikacije kojima pristupamo putem Interneta, dakle klijent ne treba aplikaciju instalirati na osobno računalo kako bi ju koristio. Sastoje se od klijentskog i poslužiteljskog dijela aplikacije. [1] Jedna od najbitnijih komponenata kod razvoja web aplikacije jest baza podataka. **Baza podataka** je organizirani skup podataka kojima se pristupa putem računala. Organizirane su na način da se podaci iz njih putem upita mogu čitati, dodavati, izmjenjivati i brisati [2]. **Sustav za upravljanje bazom podataka** (eng. *Database Management System*, u nastavku SUBP) je programska podrška kojom se baze podataka izrađuju te kontroliraju podaci u bazi. Baze podataka su vrlo značajne za web aplikaciju, one sadrže sve podatke aplikacije, te ako želimo kvalitetno izrađenu dinamičnu web aplikaciju koja je uvijek ažurna, koja je interaktivna, baza podataka je obavezna. U nekoj web aplikaciji, kada korisnik pošalje zahtjev sa klijentske strane za npr. prikaz nekog podatka, web poslužitelj taj zahtjev prosljeđuje na poslužiteljski dio aplikacije, koja preko upita iz baze podataka dohvaća traženu informaciju. Tu informaciju zatim poslužiteljski dio aplikacije šalje web poslužitelju koji je u konačnici prikaže klijentu na ekranu.

Kako se tehnologija danas razvija iz dana u dan, količina informacija se povećava, te je prilikom izrade web aplikacija važno izabrati pravi model baza podataka za nju. Zato je potrebno razumjeti na koji način rade različiti modeli. Već desetljećima **relacijske baze podataka**, koje se temelje na relacijskom modelu podataka i koriste upitni jezik SQL, dominiraju u području razvoja web aplikacija. No neki nedostaci poput loše skalabilnost, kompliciranosti SQL-a, te manjak fleksibilnosti, dovelo je do razvoja **nerelacijskih baza podataka** koje se ne temelje na relacijskom modelu podataka, te ne koriste nužno SQL za manipulaciju podacima. Njihova popularnost raste već desetak godina, te i najveće tvrtke poput Google-a, Facebook-a i Twitter-a koriste nerelacijske baze podataka. Najpopularniji nerelacijski SUBP trenutno je MongoDB, koja je prema njihovoj web stranici „baza podataka za moderne aplikacije“ [3].

Rad je organiziran tako da je u 2. poglavlju dan prikaz povijesti razvoja baza podataka te osvrt na trenutno stanje i budućnost. U 3. poglavlju objašnjene su relacijske baze podataka, ACID svojstva, te pregled dvaju SUBP – MySQL i Postgre SQL. U 4. poglavlju opisane su nerelacijske baze podataka (NoSQL baze podataka), četiri osnovne vrste NoSQL baza podataka: ključ-vrijednost baze podataka, graf baze podataka, stupčane baze podataka i dokument baze podataka, te je za svaku opisan primjer. U 5. poglavlju detaljnije je opisana MongoDB baza podataka, te je na primjeru web aplikacije za bilješke i podsjetnike prikazano kako se izrađuje web aplikacija koja koristi MongoDB kao bazu podataka. Naposljetku su u 6. poglavlju dani završni zaključci rada.

2. Baze podataka u razvoju web aplikacija

2.1. Povijest

Baze podataka su kao koncept postojale mnogo prije računala. Potreba za skladištenjem podataka je ključ za dobro funkcioniranje bilo kojeg sustava, pa su tako kroz povijest nastajale knjižnice, arhive koje je imala svaka organizacija, pa sve do danas, kada se to izvodi putem računala, koja su skladištenje te manipulaciju podacima neizmjerno olakšala.

Pojam „baza podataka“ prvi puta se počinje koristiti 1960-tih. Kako je računalima rasla popularnost, sve više privatnih kompanija je krenulo koristiti računala za pohranjivanje podataka. 1970. godine E.F. Codd objavljuje rad „A Relational Model of Data for Large Shared Data Banks“ u kojem predlaže korištenje relacijskog modela podataka [4]. Tijekom ovog razdoblja su stvorena i dva bitna sustava za upravljanjem racionalnom bazom podataka: Ingres, koji se kasnije razvio u Postgres, te System R tvrtke IBM. 1976., P. Chen stvara model Entiteta-Veza.

U 80-tima SQL (Structured Query Language) postaje standard, a relacijske baze komercijalni uspjeh, te IBM stvara DB2, a nastaju i mnoge druge tvrtke koje se bave razvojem, uglavnom relacijskih, baza podataka. 1985. je stvoren i koncept objektno-orijentirane baze podataka, koji je dobro funkcionirao s objektno-orijentiranim jezicima, te je podržavao mnogo više različitih tipova podataka. U prvoj polovici 1990.-tih, nastaje objektno relacijski model podataka, koji je stvoren da iskoristi najbolje od oba modela baza podataka. U istom razdoblju, najviše se razvijaju alati za razvoj aplikacija, kao što su PowerBulider (Sybase), Oracle Developer te VB (Microsoft), te se razvijaju alati poput Excel-a/Access-a i ODBC (Open Database Connectivity) [5].

Engleski znanstvenik Tim Berners-Lee 1989. stvara World Wide Web, koji nekoliko godina kasnije postaje dostupan cijelom svijetu. Pojavom WWW-a na globalnoj razini, kompanije kreću se razvijanjem alata koji će povezivati baze podataka s njim, kao što su Dream Weaver, Front Page, Active Aerver Pages, Java Servelets, Oracle Developer 2000, itd. Također se pojavljuju i open source rješenja kao što su Apache, GCC (GNU Compiler Collection), CGI (Computer Generated Imagery) i MySQL. 1995. godine Netscape Communications stvaraju JavaScript, a 1997. nastaje XML (Extensible Markup Language).

U 2000.-tima nastaju mnoge web aplikacije koje sve više postaju interaktivne, a vodeće kompanije za baze podataka su Microsoft, IBM i Oracle. Kreće i razvijanje baza podataka u oblaku kada je Amazon stvorio Amazon Web Services. Kako se tržište širilo, tako su se i zahtjevi tržišta mijenjali, te su se krenule razvijati baze podataka koje se nisu temeljile na relacijskom modelu podataka i za koje nije nužno znanje SQL-a. Tako u kasnim 2000-tim nastaju NoSQL baze podataka, točnije razvijaju se ključ-vrijednost baze podataka, dokument baze podataka, graf baze podataka itd., a u isto vrijeme se razvijaju i velike kompanije poput Google-a, Amazon-a, Twitter-a i Facebook-a.

2.2. Sadašnje stanje i budućnost

U zadnjih nekoliko godina, stvoreno je više podataka nego u cijeloj ljudskoj povijesti, te zbog toga postoji velika potreba za brzim i efektivnim skladištenjem podataka i njihovom manipulacijom [6]. Iako popularnost NoSQL baza podataka svake godine sve više raste, relacijske baze podataka i SQL su još uvijek najpopularniji kod razvoja web aplikacija i dominiraju tržištem, kao što prikazuje Tablica 1, te je čak 60.5% implementiranih baza podataka u svijetu spada u relacijske baze podataka. Globalna vrijednost SUBP-a je u 2020. procijenjena na gotovo 63.1 milijardu američkih dolara, a u 2026. se predviđa da će se ta brojka dignuti na 125.6 milijardu američkih dolara [6].

Poredak	**SUBP**	**Model baze podataka**
1.	Oracle	Relacijski
2.	MySQL	Relacijski
3.	Microsoft SQL Server	Relacijski
4.	PostgreSQL	Relacijski
5.	MongoDB	Dokument
6.	Redis	Ključ-vrijednost
7.	IBM Db2	Relacijski

Tablica 1 Najpopularniji sustavi za upravljanje bazom podataka (SUBP) u kolovozu 2021. *[7]*

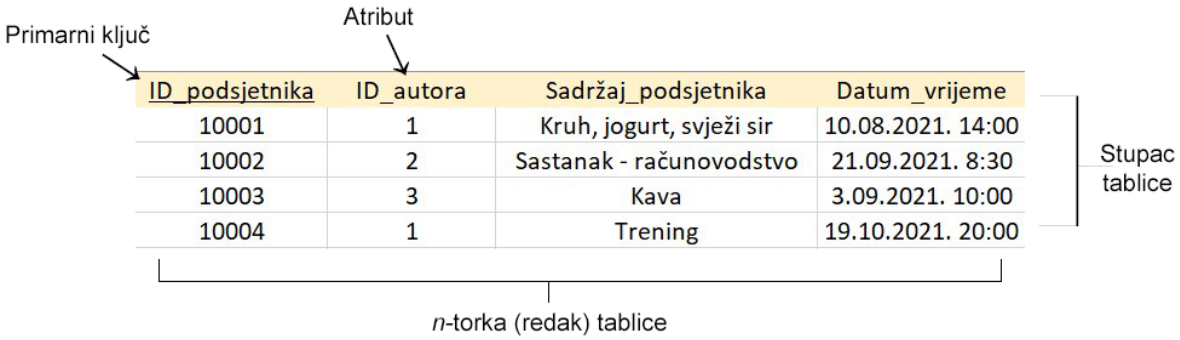
Velike kompanije se prilagođavaju novim standardima te unaprjeđivanju postojeće SUBP i stvaraju nove tako da su sve više SUBP-a dostupni i u oblaku, visoko skalabilni. Isto tako se okreću prema modelu više modela podataka (eng. *multimodel*), kao što npr. Oracle i MySQL osim racionalne baze podataka, podržavaju i dokument baze podataka, graf baze podataka, ključ-vrijednost baze podataka koje su distribuirane. U posljednje vrijeme pokušava se i implementirati i umjetna inteligencija te postupci strojnog učenja, koje bi se koristile u bazi podataka na način se predviđaju i stvaraju podatci u bazi na temelju dosadašnjih podataka, kao što funkcionira startup MindsDB, ili Oracle Autonomous Database i Microsoft SQL Server Machine Learning Services. Veliki fokus je i na sigurnosti baza podataka i zaštiti podataka. Pa je, tako, primjerice, Oracle integrirao *always-on* enkripciju, a Amazon ima *bulit-in firewall* [6].

3. Relacijske baze podataka

Relacijske baze podataka, poznate i kao SQL baze podataka, su baze podataka koje se temelje na relacijskom modelu podataka, što znači da su podaci prikazani u obliku relacija (tablica), te operacije koje možemo obavljati nad podacima u relaciji su: upisivanje, izmjenjivanje, čitanje i brisanje. Takve baze podataka koriste Structured Query Language (SQL) za manipulaciju podacima u relacijama. SQL je deklarativni programski jezik kojim se komunicira putem upita s relacijskom bazom podataka.

Relacijski model podataka je predstavio E.F. Codd u članku 1970. godine. Baziran je na matematičkoj teoriji, a Coddova definicija relacije je: *„Neka su dani skupovi $D_1, D_2, D_3, \dots, D_n$ (ne obavezno različiti). R je **relacija** nad tih n skupova $(n > 0)$ ako je to skup n-torki takav da za svaku n-torku vrijedi da je prvi element n-torke iz D_1, drugi iz D_2, …, n-ti iz D_n."* [8].

Relacijska baza podataka se sastoji od najmanje dvije relacije (tablice), koje sadrže stupce i retke. Svaki stupac tablice jest specifični atribut, a broj atributa n je jedan redak tablice. Svaka n-torka ima svoj jedinstveni ključ, te se preko ključa tablice povezuju u relacije.



Slika 1 Primjer relacije PODSJETNIK

Na Slici 1 je primjer jedne relacije PODSJETNIK. Primarni ključ (ID_podsjetnika) je atribut relacije koji osigurava svojstvo jedinstvenosti u relaciji, dakle nikada neće biti 2 ista retka u relaciji jer će primarni ključ uvijek biti unikatan. Primarni ključ može postati i **vanjski ključ** (ID_autora), što je zapravo primarni ključ neke druge tablice, npr. u tablici AUTOR, gdje bi se skladištili podaci o autoru, kao npr. Ime_autora, Prezime_autora, itd. Na taj način se u relacijskom modelu povezuju dvije tablice.

Prilikom stvaranja relacijske baze podataka, potrebno je dobro razmisliti o strukturi tablica, jer je strukturu kasnije vrlo teško izmijeniti.

3.1. ACID svojstva

Pouzdanost i dosljednost relacijske baze podataka se provjerava pomoću četiri ACID svojstva transakcije: atomičnost, konzistentnost, izolaciju i trajnost [9]. **Transakcija** je jedna ili više SQL operacija koje se promatraju kao jedna cjelina. ACID svojstva osiguravaju:

- **Atomičnost** (eng. **A**tomicity): ako je i samo jedan dio transakcije nedovršen, onda se cijela transakcija smatra nedovršenom, i transakcija se ne izvrši.

- **Konzistentnost** (eng. **C**onsistency): baza podataka mora uvijek biti stabilna, i prije i poslije svake transakcije.
- **Izolaciju** (eng. **I**solation): ovo svojstvo osigurava da ako se u isto vrijeme izvode različite transakcije, jedna drugu neće ometati, obje će biti izolirane.
- **Trajnost** (eng. **D**urability): kada je transakcija izvršena, ostat će u istom stanju, trajno pohranjena, i u slučaju grešaka, pada sustava ili bilo kakvog kvara [9].

Kraticu ACID su osmislili Andreas Reuter i Theo Harder 1983. godine, a prema njima, IBM Information Menagment System je već 1973. godine funkcionirao na način da je podržavao ACID svojstva [10].

U svim većim relacijskim SUBP-a transakcije se izvršavaju prema ACID svojstvima.

3.2. Pregled relacijskih SUBP-a

3.2.1. MySQL

MySQL je besplatan *open source* relacijski SUBP koji je stvorila švedska tvrtka MySQL AB 1995. godine. Koriste ga mnoge velike kompanije poput Adobe, Facebook i Google, te je jedna od najpopularnijih baza podataka, osobito pri razvoju web aplikacija, kao npr. WordPress, phpBB, Joomla. [11]

MySQL je napisana u C i C++, te koristi SQL. Dio je **LAMP** (**L**inux, **A**pache, **M**ySQL, **P**HP/**P**erl/**P**ython) platforme koju čine tehnologije napisane u zagradama, a LAMP je najčešća korištena platforma pri izradi web aplikacija visokih performansi u današnje vrijeme. Koristi se kada je potrebna jednostavna, brza i lako razumljiva baza podataka. MySQL je moguće koristiti na mnogo operacijskih sustava, brza je, a globalno je smatraju najsigurnijom bazom podataka. Trenutno je u vlasništvu Oracla, koji nudi i premium verzije MySQL baze podatka.

3.2.2. Postgre SQL

PostgreSQL je također besplatan i *open source* SUBP. Originalno ime mu je bilo POSTGRES, referenca na svog prethodnika, bazu podataka Ingres. Prvi puta je izdan 1996. godine, a razvio ga je tim sa Sveučilišta Berkeley u Kaliforniji.

PostgreSQL je objektno-relacijska baza podatka, što znači da ima relacijski model podataka, ali u tablice može smještati podatke kao objekte. Iz tog razloga PostgreSQL nudi puno više značajki od MySQL. Stvorena je za kompleksne upite, ogromne baze podataka (nema ograničenja veličine), te podržava razne tipove podataka. Za razvoj web aplikacija koriste ga Apple, Skype, Cisco, itd.

4. Nerelacijske baze podataka

Nerelacijske baze podataka ili NoSQL („Not only SQL“ ili „non SQL") su baze podataka čija je struktura dinamična, dakle ne temelje se na relacijskom modelu podataka. Nastale su kasnih 2000-ih, kada su se cijene memorija naglo snizile. Kako su se cijena snizile, tako se povećala količina podataka. Zato su nastale NoSQL baze podataka, koje su omogućile fleksibilnost, horizontalno skaliranje, bržu pohranu i obradu podataka širokog opsega, ali i jednostavnost korištenja. Ubrzo su postale vrlo popularne i kod web aplikacija.

NoSQL baze podataka možemo klasificirati po načinu na koji su podaci organizirani, a najosnovnije su:

- Ključ – vrijednost baze (eng. *Key – value database*)
- Dokument baze (eng. *Document database*)
- Graf baze (eng. *Graph database*)
- Stupčane baze (eng. *Columnar database*)

4.1. Ključ – vrijednost baze podataka

Ključ – vrijednost baze su najjednostavnije NoSQL baze podataka, temelje se na modelu podataka u kojem se podaci spremaju u obliku parova (ključ, vrijednost) gdje je jednom tekstualnom ključu pridružena jedna vrijednost bilo kojeg tipa. Takve baze podataka ne koriste SQL, a od manipulacije podacima omogućavaju čitanje, stvaranje i brisanje parova na osnovu ključa putem API, što su operacije:

1. GET (ključ) – koja vraća vrijednost pridruženu uz ključ,
2. PUT (ključ, vrijednost) – koja dodaje novi par ili novu vrijednost ključu,
3. DELETE (ključ) – koja briše par, dakle i ključ i vrijednost

Format ključa može biti bilo kakav znakovni niz, dok format vrijednosti može biti i tekst, ali i slika, zvuk, dokument, web stranica, itd. Za ove baze podataka jedina su pravila da svaki ključ mora bit jedinstven, te da upiti mogu ići jedino preko ključa. Prednost ovakvih baza su ta što imaju veliku skalabilnost. Zbog jednostavnih operacija, podaci se mogu modelirati na način da se u potpunosti iskoristi učinkovitost sustava.

Baze podataka koje su primjer ključ – vrijednost baze podataka su: AmazonDynamoDB, Riak, Redis, Oracle NoSQL.

4.1.1. Redis

Redis je jednostavna *open source* ključ-vrijednost baza podataka nastala 2009. godine čiji je autor Salvatore Sanfilippo. Cilj mu je bio da pomoću Redis baze poboljša skalabilnost vlastitog startup projekta. Redis radi na način da cijeli skup podatka smješta na memoriju, te ga u proizvoljnom vremenskom razdoblju i aksinkrono smješta na disk. Zbog takvog načina rada, Redis je jedna od najbržih ključ-vrijednost baza podataka, no zbog mogućnosti lakog gubitka podataka, nije pogodan za veliki broj web aplikacija.

4.2. Graf baze podataka

Kod graf baza podataka za pohranu podataka koristi se struktura grafa na način da je naglasak na vezi između čvorova grafa (entiteta). Čvorovi grafa predstavljaju nekakav objekt u aplikaciji, npr. radnik ili proizvod, a veze (lukovi) između čvorova su odnosi među njima. Specifičnost ovog modela podataka ja te da čak i lukovi mogu imati vlastita svojstva, te je i smjer lukova isto tako bitan.

Ove baze podataka koriste se najčešće u aplikacijama gdje su važne veze između podataka, kao na primjer društvene mreže. U odnosu na relacijski model podataka, veze između čvorova grafa su eksplicitno definirane i fleksibilne, dok se kod relacija to odvija implicitno putem ključa. Lukovi u grafu se dobivaju korištenjem operacije pridruživanja (*join*) [12].

Ne postoji univerzalni upitni jezik za graf baze podataka. 2019. godine je za to predložen GQL, a koriste se još i Gremlin, SPARQL i Cypher. Neke graf baze podataka komuniciraju s aplikacijom putem API-a.

Primjeri ove vrste baza podataka su Amazon Neptune, ArangoDB, Neo4j.

4.2.1. Neo4j

Neo4j je *open source* NoSQL SUBP izdan 2007. godine. Napisan je u Javi, te je jedan od NoSQL baza podataka koje podržavaju ACID svojstva transakcije. Koristi Cypher kao upitni jezik. U Neo4j, podaci mogu biti spremljeni u obliku luka (veze), čvora ili atributa. Svaki luk i čvor mogu imati neograničen broj atributa.

4.3. Stupčane baze podataka

Stupčane baze podataka spremaju podatke u stupcima, koje možemo zamisliti kao stupce tablice. No, u odnosu na stupce tablice u relacijskom modelu podataka, kod stupčanih baze podataka se pristupa pojedinačnom stupcu bez obzira na ostale stupce, te svaki stupac ima proizvoljni format. Svaki stupac ima naziv i pripadajuću vrijednost.

Primjeri ovakve nerelacijske baze podataka su: Apache Cassandra i Bigtable.

4.3.1. Apache Cassandra

Apache Cassandra je besplatni *open source* distrubuirani SUBP također napisan u Javi. Visoko je skalabilna, fleksibilna, te brza u obradi podataka, pa ju zato koriste i Apple, Netflix, GitHub itd. Napravio ju je Avinash Laskshman 2008. godine kako bi poboljšao Facebook-ovu tražilicu. Koristi vlastiti upitni jezik, Cassandra Query Language (CQL).

4.4. Dokumentne baze podataka

Dokumentne baze podataka su najpopularnije NoSQL baze podataka u svijetu. Ovaj model je intuitivan, te proširuje koncept ključ – vrijednost baze podataka, tako što se podaci spremaju u dokument. Dokument možemo opisati kao uređeni skup ključeva s pridruženim vrijednostima. Dokumenti se smještaju u kolekcije, kako bi se korisnik lakše snalazio u bazi, a kolekcije se nazivaju npr. FILM, što znači da će svaki dokument sadržavati podatke o filmu. U kolekcijama je shema nedefinirana, dakle svaki dokument u kolekciji može imati posebnu strukturu. Dokumenti mogu čak sadržavati i druge dokumente, što omogućava veću fleksibilnost te brzinu i lakoću pristupa svim povezanim dokumentima. Svaki dokument ima jedinstveni ključ.

Dokument baze podataka su stvorene s ciljem skalabilnosti, da mogu dobro funkcionirati s velikom količinom podataka distribuiranih na nekoliko servera. Ima nekoliko formata u kojima se spremaju dokumenti, ali najčešći su JSON, XML i YAML.

Najpoznatiji i trenutno najpopularniji primjer dokument baze podataka je MongoDB, o kojem ću pisati detaljnije u poglavlju 5. Ostali primjeri ovih baza su MarkLogic Server, OrientDB, InterSystems Cache, BaseX, Couchbase Server itd.

5. MongoDB

MongoDB je *open source* SUBP, kreiran 2007. godine. Osnovali su je Dwight Merriman, Eliot Horowitz i Kevin Ryan koji su njome htjeli riješiti probleme skalabilnosti i agilnosti kod kojih su se susretali prilikom korištenja drugih baza podataka [3]. MongoDB je trenutno najpopularnija *open source* NoSQL baza podataka, dok je 5. na ljestvici popularnosti baza podataka općenito, a za razloge njene popularnosti se često navode fleksibilnost i jednostavnost korištenja [7]. Napisana je u C, C++ i JavaScript-u. Iako je nerelacijska SUPB, podržava ACID svojstva.

5.1. Dokumenti

MongoDB radi na način da su podaci u bazi organizirani u dokumente, u BSON formatu, gdje svaki dokument može imati drugačiju strukturu. BSON, „Binary JSON“, je format baziran na JSON formatu. BSON format proširuje JSON format, kao što prikazuje Tablica 2, tako što podržava više tipova vrijednosti, te je optimiziran za pohranjivanje podataka u bazu, a podaci zapisani u BSON obliku u bazi u sebi sadrže i veličinu dokumenta te tip dokumenta [3].

	JSON	**BSON**
Kodiranje	UTF-8	Binarno
Podržani tipovi podataka	String, Boolean, Number, Array	String, Boolean, Number (Integer, float, long, decimal128,…), Array, Date, Raw Binary
Čitljivost	Čovjek i računalo	Računalo
Koristi se za	Slanje podataka putem mreže	Pohranjivanje u bazu

Tablica 2 Usporedba JSON i BSON *[3]*

U MongoDB, dokumenti se sastoje od parova ključa i njihovih pripadnih vrijednost. Ključevi su jedinstveni, a ulogu primarnog ključa ima ključ **_id**, te se on nalazi uvijek na vrhu dokumenta u bazi. Vrijednost primarnog ključa _id koji sadržava svaki dokument u MongoDB je posebnog tipa *ObjectId*. ObjectId je automatski stvoren prilikom unosa novog dokumenta u bazu ako korisnik sam ne definira vrijednost ključa _id, a ako korisnik pokuša stvoriti dokument s vrijednosti primarnog ključa koja već postoji, MongoDB javlja grešku i takav dokument nije moguće spremiti u bazu. ObjectId je duljine 12 bita, a čine ga:

- 4-bitna oznaka vremena u kojem je stvoren dokument (milisekunde od UNIX epohe),
- 5 bita zauzima nasumična vrijednost, te
- 3 bita zauzima brojač koji je inicijaliziran na slučajnu vrijednost [13].

Ostali ključevi u dokumentu su tipa *string*. Tip vrijednosti u dokumentima može biti bilo koji BSON tip, drugi dokument (*embedded/nested documents*), polja, te polja dokumenata. Slika 3 prikazuje primjer dokumenta *mydoc*, gdje možemo uočiti različite tipove podatka u dokumentu. Na vrhu je primarni ključ *_id* i on je tipa ObjectId, a ispod njega je ključ *name* koji u sebi sadrži drugi dokument koji se sastoji od dva ključa, *first* i *last*, koji su oboje tipa String. Sljedeća dva ključa *birth* i *death* su tipa Date, ključ *contribs* je tipa Array, a zadnji ključ *views* je tipa NumberLong [13].

```
var mydoc = {

    _id: ObjectId("5099803df3f4948bd2f98391"),
    name: { first: "Alan", last: "Turing" },
    birth: new Date('Jun 23, 1912'),
    death: new Date('Jun 07, 1954'),
    contribs: [ "Turing machine", "Turing test", "Turingery" ],
    views : NumberLong(1250000)

}
```

Slika 2 Primjer dokumenta u MongoDB s različitim tipovima vrijednosti *[13]*

5.2. Kolekcije i baze podataka

Kolekcija je skup dokumenata u bazi podataka, a možemo ju usporediti sa tablicom u relacijskom modelu podataka. Kolekcije postoje unutar jedne baze podataka u MongoDB. Mogu se stvoriti direktno u odabranoj bazi podataka putem MongoDB web aplikacije, ili preko vlastite web aplikacije. Ako neka kolekcija još ne postoji u bazi podataka, stvorit će se prilikom unosa novog podatka. Kolekcije služe kako bi se organizirali dokumenti u logičku cjelinu, pa se tako kolekcije imenuju na način da jasno odrede o kakvom tipu dokumenata je riječ. Na primjer, neka kolekcija će se zvati Mobitel, što znači da će svaki dokument u toj kolekciji sadržavati podatke o nekom mobitelu. Ali, naravno, u svakoj kolekciji struktura svakog dokumenta je proizvoljna [13].

U MongoDB, baza podataka sadrži jednu ili više kolekcija. Ako baza podataka ne postoji, stvorit će se automatski prilikom unosa novog podatka u bazu. Jedan korisnik može imati više baza podataka, a svaka baza ima i različite dozvole koje određuje administrator.

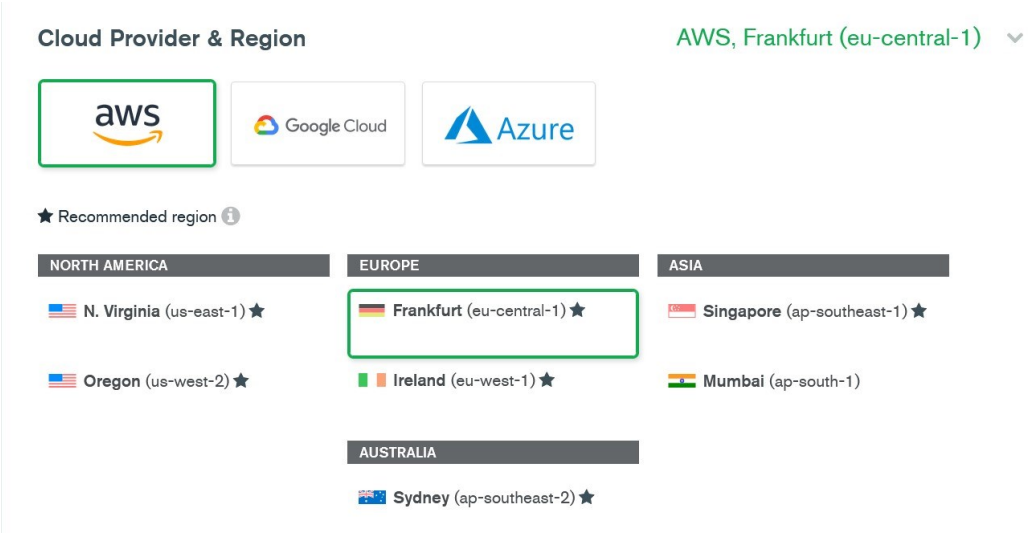
5.3. Primjer jednostavne web aplikacije

Za potrebe detaljnijeg pojašnjenja načina korištenja MongoDB u razvoju web aplikacija, izradila sam jednostavnu web aplikaciju za bilješke i podsjetnike. Aplikacija uključuje registraciju (signup) za nove korisnike i prijavu (login) za postojeće, zatim unos, čitanje, uređivanje i brisanje bilješki i podsjetnika. Aplikacija je izrađena koristeći Node.js, Express, Mongoose, te MongoDB Atlas.

5.3.1. MongoDB Atlas

MongoDB Atlas je *Database-as-a-Service* baza podataka. Nalazi se u oblaku (cloud) što znači da korisnik ne treba preuzimati i instalirati programsku podršku kako bi joj mogao pristupiti i koristiti ju. Prema njihovoj web stranici, MongoDB Atlas je najnapredniji sustav za upravljanje bazom podataka [3].

Kako bi se koristila ova SUBP, potrebna je registracija na njihovom web sjedištu, a registracija je moguća ili putem Gmail-a ili emaila. Nakon registracije u vrlo kratkom roku stiže i email za potvrdu email adrese, te nakon što se klikne na poveznicu, registracija je izvršena. MongoDB Atlas pruža dvije mogućnosti za postavljanje baze podataka na oblak: putem klastera (eng. *cluster*) i bez poslužitelja (eng. *serverless instances*). U primjeru web aplikacije koristila sam klastere. Kako je vidljivo na Slici 4, moguće je izabrati i između 3 dobavljača oblaka (eng. *cloud provider*): Amazon Web Services, Google Cloud i Microsoft Azure, što omogućava lakšu migraciju podataka.



Slika 3 Odabir dobavljača oblaka i regije na MongoDB Atlas računu

Također se u ovom koraku bira ime i rang klastera (eng. *cluster tier*), te s obzirom na niske potrebe za moju jednostavnu aplikaciju, odabrala sam besplatni i najniži rang, M0 Sandbox, u kojem je na raspolaganju 512 MB prostora, maksimalno 500 kolekcija i 100 baza podataka. Stvaranje klastera traje od 1-3 minute. Klaster se stvori u projektu kojeg smo prethodno proizvoljno imenovali, u ovom slučaju radi se o projektu naziva *Biljeske i podsjetnici*.

Kako bi tijekom razvoja web aplikacije povezali bazu podatka i aplikaciju, potrebno je odobriti koji korisnici mogu pristupiti, te IP adrese koje mogu pristupit bazi. Na taj način se ostvaruje sigurnost i očuvanje baze podataka.

5.3.2. Mongoose

Mongoose je biblioteka za objektno modeliranje podataka za Node.js koja se koristi za jednostavnije upravljanje objekata u MongoDB. Instalirati je možemo putem terminala naredbom

```
npm install mongoose
```

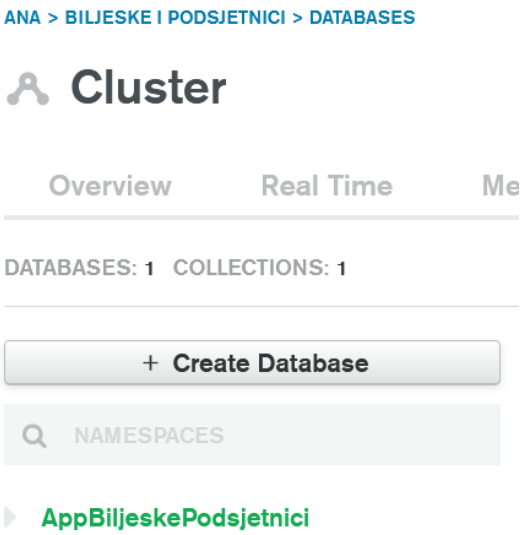
5.3.3. Povezivanje web aplikacije i MongoDB Atlas

U datoteci *app.js* nalazi se serverski dio aplikacije. Web aplikaciju povezujemo sa MongoDB Atlasom na način da pozovemo *connect* metodu, kao što je prikazano na Slici 5, te u zagradu zalijepimo *string* koji nam stvori MongoDB Atlas na vlastitom računu. Potrebno je samo izmijeniti lozinku, te naziv baze podataka ako to želimo. Možemo primijetiti da je naziv baze podataka **AppBiljeskePodsjetnici**. Nakon što se prvi put pokrene aplikacija, te se uspješno poveže s MongoDB, u klasteru možemo vidjeti da je stvorena baza podataka AppBiljeskePodsjetnici.

```
mongoose.connect('mongodb+srv://ana:zavrsniadmin@cluster.ossmw.mongodb.net/AppBiljeskePodsjetnici?retryWrites=true&w=majority',
{useNewUrlParser: true, useUnifiedTopology: true, useCreateIndex:true,});
const db = mongoose.connection;
db.on('error', console.error.bind(console, 'connection error:'));
db.once('open', function() {
    console.log('Connected to MongoDB!');
});
```

Slika 4 Povezivanje web aplikacije s MongoDB Atlas putem *connect* metode

Nakon što se prvi put pokrene aplikacija, te se uspješno poveže s MongoDB, u klasteru možemo vidjeti da je stvorena baza podataka AppBiljeskePodsjetnici, kao što prikazuje slika 6.



Slika 5 Stvorena nova baza podataka *AppBiljeskePodsjetnici* u klasteru *Cluster*

5.3.4. Registracija i prijava

Kako bi u stvorenu bazu podataka *AppBiljeskePodsjetnici* bilo moguće putem web aplikacije stvoriti kolekciju *user*, u mapi *models* je kreirana datoteku *User.js*. U *User.js* se nalazi shema *userSchema*. Shema definira strukturu dokumenta u kolekciji *user*. Na Slici 7 možemo vidjeti da definirana shema *userSchema* ima 3 ključa, a to su *email*, *userName* i *password*. Za svaki ključ možemo postaviti njegov tip, a u Mongoose je to ShemaType. Dozvoljeni tipovi su: String, Number, Date, Buffer,

Boolean, Mixed, ObjectId, Array, Decimal128 i Map. Također se može postaviti jedinstvenost ključa (*unique*), te je li ključ obavezan (*reqired*). U *userSchema* svi ključevi su tipa String, te se ne smiju ostaviti kao prazna polja, a *email* i *userName* moraju biti jedinstveni. Može se primijetiti kako u ovoj shemi nema primarnog ključa _id. On se može dodati, ali i ne mora. No prilikom registracije novog *user*-a, njegov _id će se automatski stvoriti na vrhu dokumenta u MongoDB Atlas.

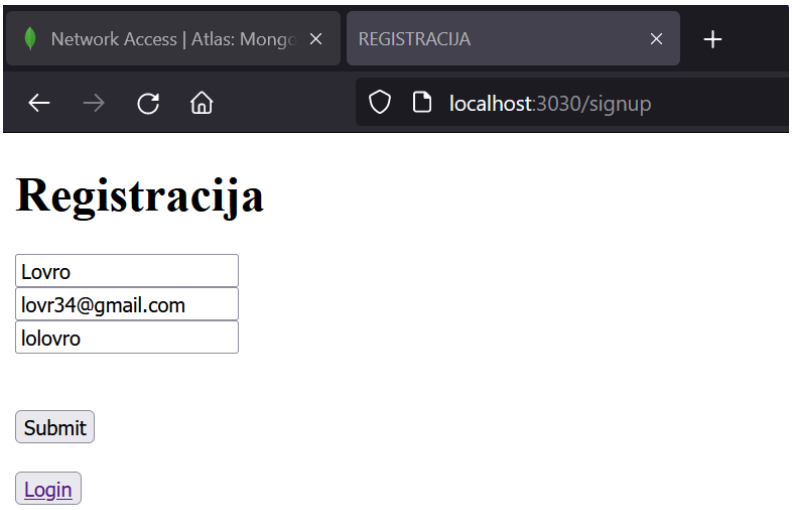
```
const mongoose = require('mongoose');

const userSchema = mongoose.Schema({
    email: {
        type: String,
        unique: true,
        required: true
    },
    userName: {
        type: String,
        unique: true,
        required: true
    },
    password: {
        type: String,
        required: true
    }
})
```

Slika 6 Mongoose shema dokumenta u kolekciji *user*

Na taj način se pomoću Mongoose stvaraju modeli koji odgovaraju kolekciji u MongoDB Atlas. Kako bi model *User* mogli koristiti u cijeloj aplikaciji, potrebno ga je izvesti (*export*), a to u Node.js radimo ovom linijom koda:

```
module.exports = mongoose.model('User', userSchema);
```

Sada kada imamo model *User*, u mapi *views* u datoteku *signup.html* napišemo jednostavnu HTML datoteku u kojoj je obrazac za registraciju korisnika u web aplikaciju, koji u web pregledniku izgleda kao na Slici 8. U obrazac korisnik upisuje korisničko ime, email i lozinku.



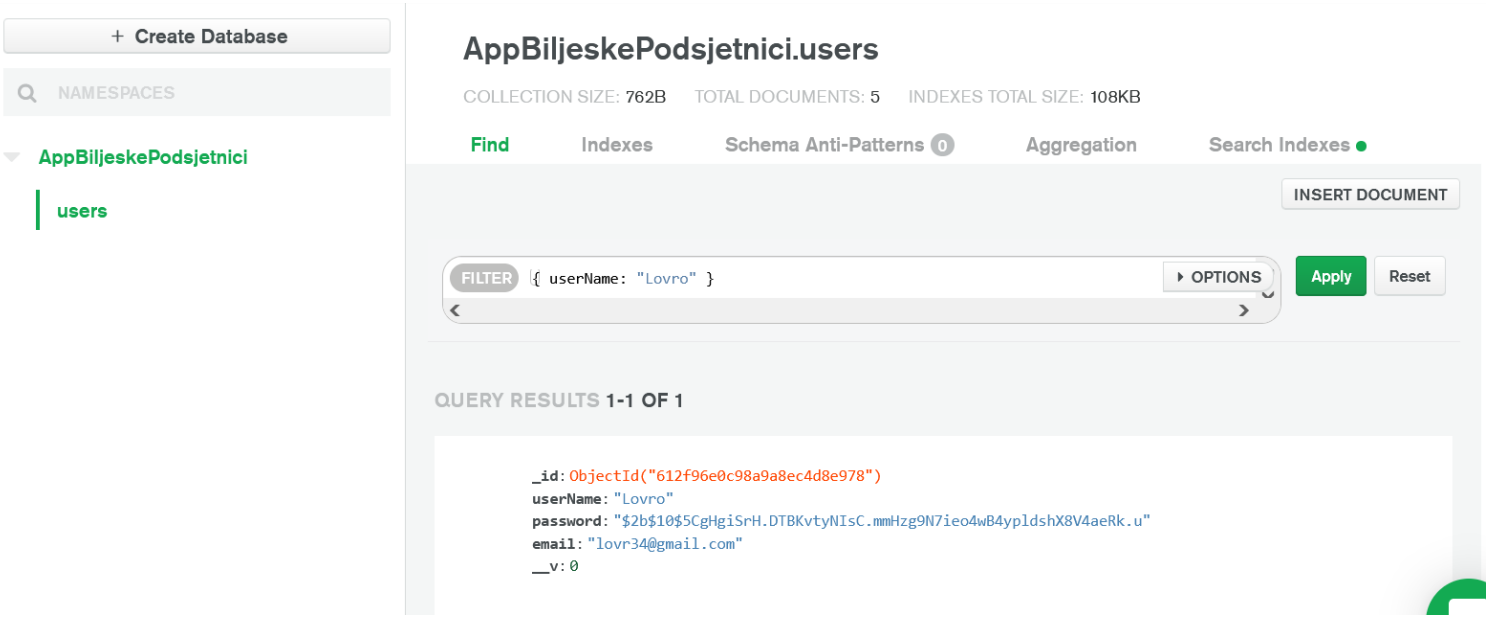
Slika 7 Obrazac za registraciju novih korisnika

Kako bi upisani podaci dospjeli u bazu podatka, potrebno je napisati API za stvaranje novog korisnika. Obrazac iz datoteke *signup.html* POST metodom šalje podatke na */user* u middleware funkciju, koja omogućuje objekte zahtjev (eng. *request*) i odgovor (eng. *response*), kao što možemo vidjeti na slici 9. Zato stvaramo varijablu *user* u koju se sprema novi User model. U ključeve se preko zahtjeva dohvaća tijelo html dokumenta te ono što smo upisali u obrazac. Kod API-a za registraciju korisnika koristit ćemo biblioteku *bcrypt* koja će haširati lozinke korisnika i takve ih zapisivati u dokument. Pozovemo metodu **save()** koja stvara novi dokument u kolekciji. Ako je dokument uspješno kreiran, u terminalu nas obavijesti o tome, te u web pregledniku otvara stranicu localhost:3030/, što je *login.html* stranica. Ako se dogodila greška prilikom unosa, u terminalu se pojavi obavijesti o greški te se ponovo otvori *signup.html*.

```
router.post('/user', (request, response) => {
    const user = new User({
        userName : request.body.userName,
        password : request.body.password,
        email : request.body.email
    });
    bcrypt.hash(user.password, 10, function (err, hash){
      if (err) {
          return next(err);
      }
      user.password = hash;
      user.save().then(data => {
          console.log("Successfully created a new User");
          response.redirect("/");
      }).catch(error => {
          console.log("Error - user not saved!");
          response.redirect("/signup");
      })
  })
})
```

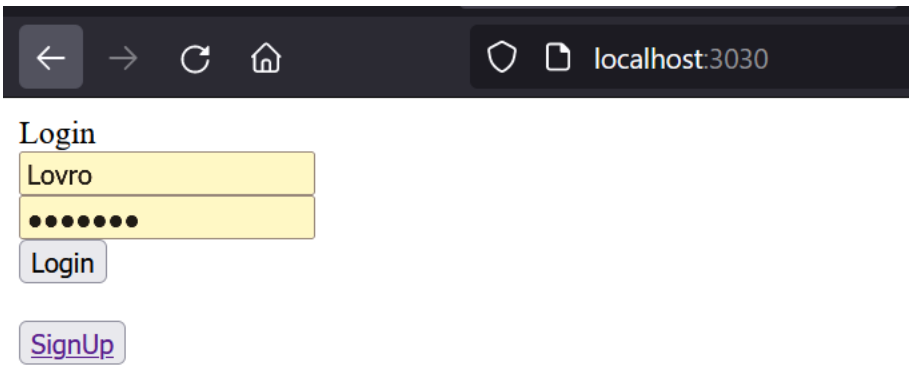
Slika 8 API za stvaranje (registraciju) novog korisnika

Ako sada otvorimo bazu podataka *AppBiljeskePodsjetnici*, možemo primijetiti da je stvorena kolekcija *users*, te da je stvoren i dokument s podacima koji se mogu vidjeti na Slici 10. Isto tako vidimo i da je automatski stvoren primarni ključ _id**.** U MongoDB Atlas možemo i pretraživati dokumente na način da upišemo u uglate zagrade ključ i vrijednost ključa koju tražimo te klikom na *Apply* nam se prikažu rezultati upita.



Slika 9 Kolekcija *user* i primjer jednog dokumenta

Nakon što se korisnik registrira, u pregledniku se otvori obrazac za login, koji se nalazi u mapi *view* u *login.html*. Korisnik upisuje korisničko ime i lozinku, a obrazac POST metodom šalje podatke na */login* middleware. Slika 11 prikazuje kako izgleda obrazac za prijavu u web pregledniku.



Slika 10 Login stranica

Za autentikaciju i autorizaciju korisnika koristi se Passport, middleware za Node.js koji nudi različite strategije za autentikaciju. U ovoj web aplikaciji poslužit će nam *Local Strategy*, strategija Passporta koja omogućuje autentikaciju putem korisničkog imena i lozinke. Prvo inicijaliziramo Passport middleware, te Local Strategy, prema uputama u Passport dokumentaciji [14]. Funkcija **authenticate()**

autentificira korisnika nakon što specificiramo strategiju 'local', te ako je korisnik uspješno autentificiran, prosljeđuje ga na početnu stranicu. Zatim se pozivaju funkcije serialiseUser, koja serijalizira korisnika i smješta određene podatke u sesiju, te deserialiseUser, koja sprema _id logiranog korisnika tako da se ga se može koristiti u nastavku koda putem *req.user.id* (prikazano na Slici 12).

```
router.use(passport.initialize());
router.use(passport.session());
const LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy(
   { usernameField: "userName" },
   (userName, password, done) => {
     User.findOne({userName: userName}, (err, userData) => {
      let passwordCheck = bcrypt.compareSync(password, userData.password);
      if (userName === userData.userName && passwordCheck) {
          return done(null, userData)
          }else{
              return done(null, false, { message: 'Korisnik ne postoji.'})
          }
        })
    }
));

router.post('/login', (req, res, next) => {
    passport.authenticate('local', (err, user, info) => {
        req.login(user, (err) => {
        res.redirect('/index');
        })
    })
    (req, res, next);
})

passport.serializeUser((user, done) => {_
    done(null, user.id);
    console.log("Serializing user: ", user);
});
passport.deserializeUser(function(id, done) {
    User.findById(id, function(err, user) {
        loggedInUser = user;
        done(err, user);
    });
});
```

Slika 11 Autentikacija i login korisnika

5.3.5. Bilješke i podsjetnici

Nakon što smo riješili registraciju, prijavu i autentikaciju korisnika, prelazimo na funkcije ove web aplikacije, odnosno CRUD (create, read, update, delete) operacije nad bilješkama i podsjetnicima. Svaki korisnik na vlastitom računu može stvarati nove te čitati, izmjenjivati i brisati postojeće bilješke i podsjetnike.

Kako bi se mogle stvarati nove bilješke i podsjetnici, potrebno je u mapi *models* stvoriti dvije datoteke: *Note.js*, u kojoj će se nalaziti shema za bilješke, te *Reminder.js* u kojoj će se definirati shema za podsjetnike. Na Slici 13 možemo vidjeti datoteku *Note.js*, shema dokumenta u kolekciji *notes*. U njoj se nalaze 3 ključa, *user* tipa String, u koji će biti proslijeđen id korisnika koji je prijavljen i koji stvori bilješku, zatim ključ *title* tipa String u kojem će biti naslov bilješke, te ključ *content* tipa String u koji će se spremiti sadržaj bilješke.

```
const mongoose = require('mongoose');

const noteSchema = mongoose.Schema({
    user: {
        type: String
    },
    title: {
        type: String
    },
    content: {
        type: String
    }
})

const Note = mongoose.model('Note', noteSchema);

module.exports = Note;
```

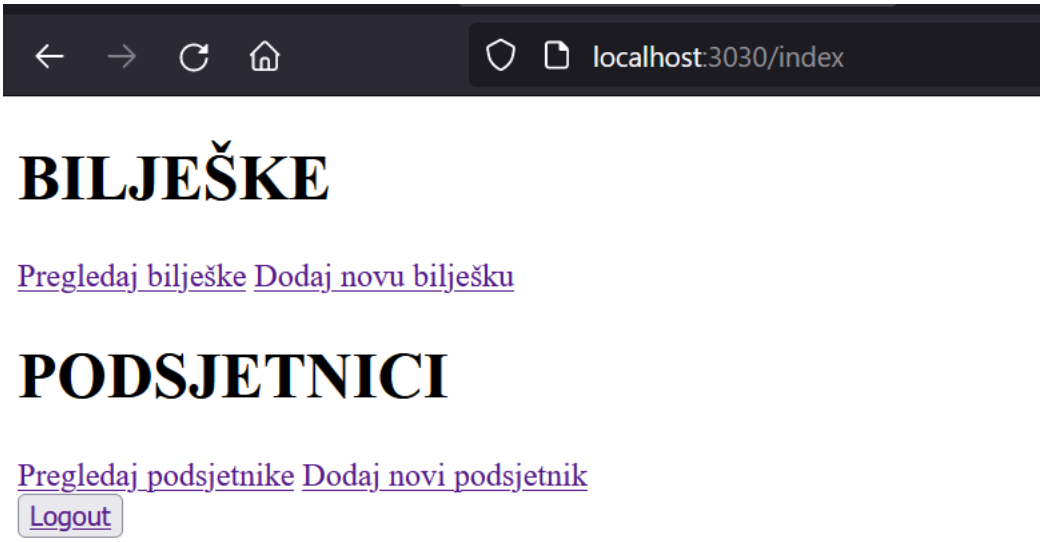
Slika 12 Mongoose shema dokumenta u kolekciji notes

Na Slici 14 je datoteka *Reminder.js* u kojoj je zapisana shema dokumenta kolekcije *reminders*. Definiran je dokument s 4 ključa: ključ *user* koji također sprema id korisnika koji je prijavljen i koji stvori podsjetnik, slijedi ključ *titleP* tipa String koji sprema naslov podsjetnika, pa ključ *contentP* tipa String koji sprema sadržaj podsjetnika i na kraju ključ *dateP* tipa Date.

```
const mongoose = require('mongoose');

const reminderSchema = mongoose.Schema({
    user: {
        type: String
    },
    titleP: {
        type: String
    },
    contentP: {
        type: String
    },
    dateP: {
        type: Date
    }
})

const Reminder = mongoose.model('Reminder', reminderSchema);
```

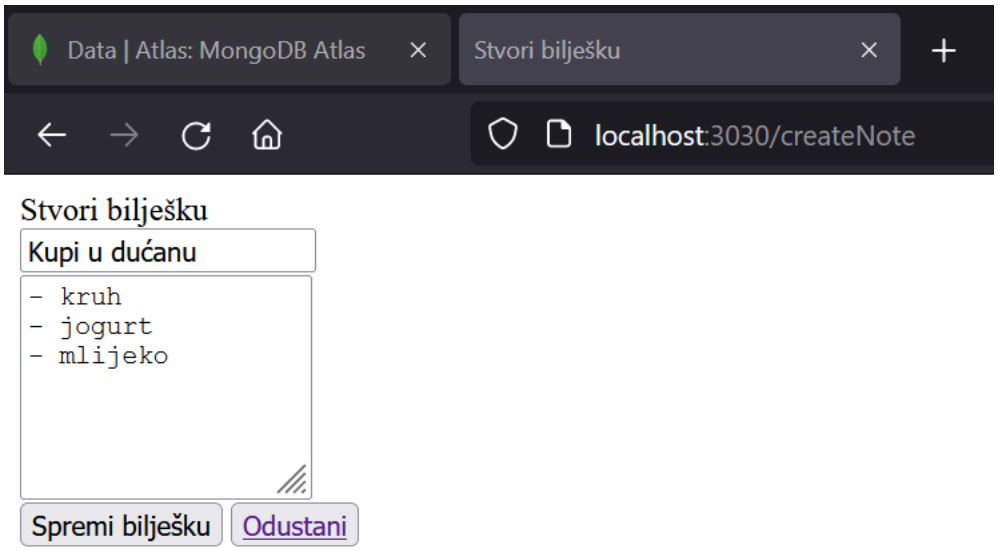
Slika 13 Mongoose shema dokumenta u kolekciji reminders

Kada se korisnik prijavi u aplikaciju, prikaže mu se početna stranica web aplikacije (prikazano na Slici 15). Početna stranica je u mapi *views* u datoteci *index.html*. Na početnoj stranici se nalaze dvije sekcije: BILJEŠKE i PODSJETNICI, u sekciji BILJEŠKE su poveznice na stranice za pregled bilješke, te dodavanje nove bilješke, a u sekciji PODSJETNICI su poveznice na stranice za pregled podsjetnika i dodavanje novog podsjetnika.



Slika 14 Početna stranica web aplikacije

Kada korisnik klikne na poveznicu *Dodaj novu bilješku*, prikaže se html datoteka *createNote.html* koja se nalazi u mapi *views*. To je jednostavna html datoteka u obliku obrazca, gdje korisnik upisuje naslov bilješke i sadržaj bilješke, kao što vidimo na Slici 16.



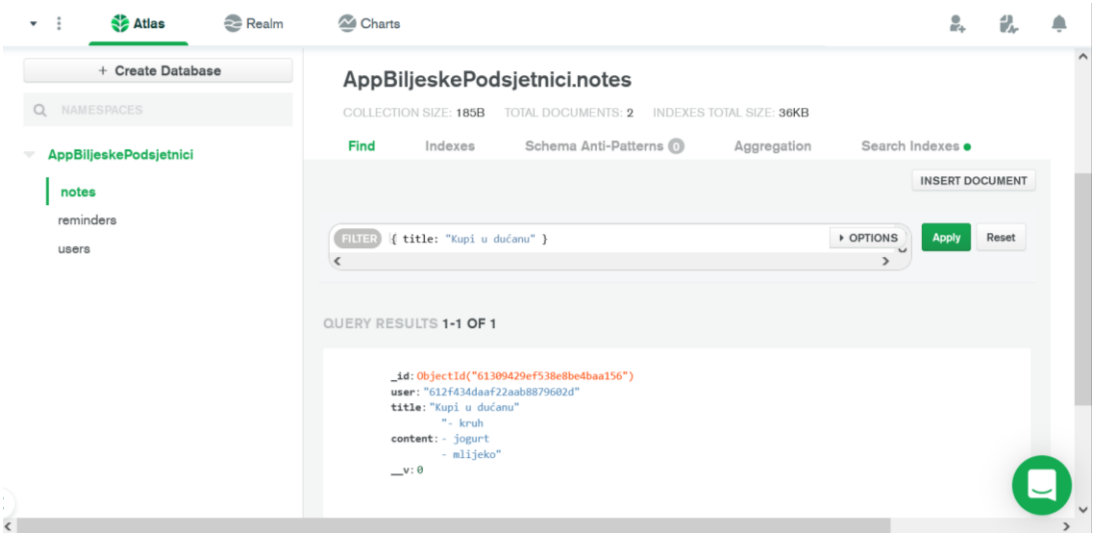
Slika 15 Web stranica za obrazac unosa nove bilješke

Obrazac se šalje POST metodom na putanju */createNote* gdje se pokreće middleware funkcija. Zahtjevima se dohvaća unos vrijednost ključa korisnika u html *body* sekciji, te se sprema u novo stvorenu bilješku. U ključ *user* spremamo id trenutno prijavljenog korisnika koji se nalazi u *req.user.id* kojeg imamo zbog Passport funkcije deserializeUser(). Metodom save() bilješka se stvori kao dokument u bazi s vrijednostima koje je unio korisnik u obrascu. Ako je bilješka uspješno spremljena, u terminalu se ispiše poruka o tome, u suprotnom nas obavijesti da je došlo do greške i bilješka nije spremljena. Zatim nas preusmjeri na putanju */notes* (prikazano na Slici 17).

```
router.post('/createNote', function(req, res) {
    const note = new Note({
            user : req.user.id,
            title : req.body.title,
            content : req.body.content
        });
        note.save().then(data => {
            console.log("Successfully created a new Note!");
        }).catch(error => {
            console.log("Error - note not saved!")
        })
    res.redirect('/notes');
})
```

Slika 16 API za stvaranje nove bilješke

Sada kada otvorimo bazu podataka *AppBiljeskePodsjetnici*, vidjet ćemo da je stvorena kolekcija *notes* i novi dokument u kolekciji (prikazano na Slici 18). Isto tako je automatski stvoren i primarni ključ _id dokumenta.



Slika 17 Kolekcija *notes* i stvoreni dokument

Analogno tomu se stvaraju i podsjetnici, te na Slici 18 možemo vidjeti da je stvorena i kolekcija *reminders*.

Putanja */notes* prikazuje datoteku *notes.ejs* iz mape *views* (prikazano na Slici 19). **EJS** je jezik za predložak (eng. *templating language*) koji spaja HTML i JavaScript [15], te pomoću njega ispisujemo na web preglednik sve bilješke iz kolekcije *notes* koje je stvorio prijavljeni korisnik, kako bi sve bilješke bile pregledno prikazane na jednom mjestu.

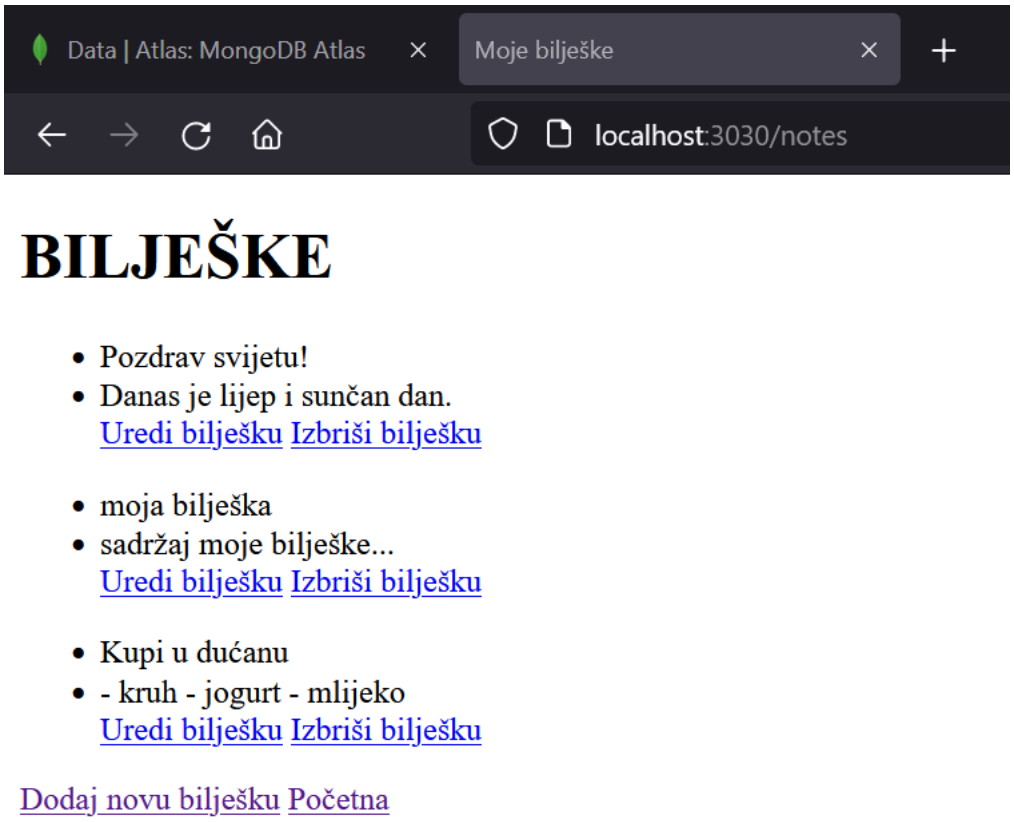
```
<!DOCTYPE html>
<html>
<head>
<title>Moje bilješke</title>
</head>
<body>
<div>
    <h1>BILJEŠKE</h1>
    <% notes.forEach(function (note) { %>
        <ul>
            <li><%= note.title %></li>
            <li><%= note.content %></li>
            <a href="/editNote/<%= note._id %>">Uredi bilješku</a>
            <a href="/deleteNote/<%= note._id %>">Izbriši bilješku</a>
        </ul>
    <% }) %>
    <a href="/createNote">Dodaj novu bilješku</a>
    <a href="/index">Početna</a>
</div>
</body>
</html>
```

Slika 18 Datoteka *notes.ejs*

```
router.get('/notes', function(req, res) {
    Note.find({ user: req.user.id }).exec(function(err, notes) {
        if (err) throw err;
        res.render('notes.ejs', { "notes": notes });
    });
});
```

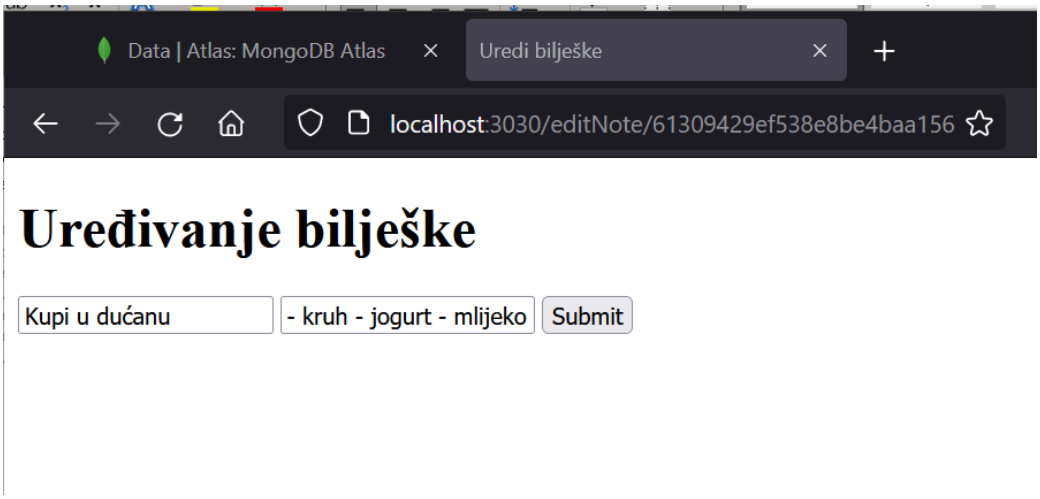
Slika 19 API za prikaz datoteke *notes.ejs*

Stranicu prikazujemo na način kako je prikazano na Slici 20. Preko Mongoose funkcije **find()** napravimo upit koji pronalazi sve bilješke (dokumente) gdje je vrijednost ključa *user* u kolekciji *notes* jednaka vrijednosti spremljenoj u *req.user.id* (id trenutno prijavljenog korisnika) kako bi mu se prikazale samo one bilješke koje je on stvorio.



Slika 20 Prikaz web stranice *Moje bilješke*

Ispod svake bilješke su poveznice *Uredi bilješku* i *Izbriši bilješku*. Promotrimo sada kako se uređuje jedna bilješka. Klikom na poveznicu *Uredi bilješku* otvori se stranica prikazana na Slici 22, koja se u projektu nalazi u mapi *views* pod imenom *editNote.ejs* (prikazana na Slici 23).



Slika 21 Web stranica za uređivanje jedne bilješke

```
<!DOCTYPE html>
<head>
    <title>Uredi bilješke</title>
</head>
<body>
    <h1>Uređivanje bilješke</h1>
    <form action="/updateNote/<%= note._id %>" method="POST">
        <input type="text" name="title" value="<%= note.title %>" />
        <input type="text" name="content" value="<%= note.content %>" />
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

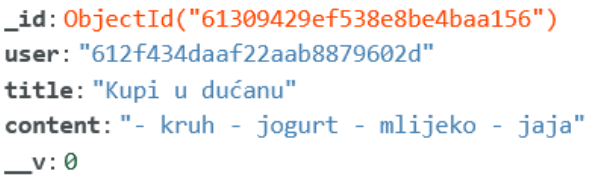
Slika 22 Datoteka *editNote.ejs*

Prikazuje se trenutna vrijednost ključeva *title* i *content*, ali se oni mogu ispravljati i uređivati. Kada se unese nova vrijednost u obrazac, metodom POST se poziva putanja */updateNote/:id* koja ažurira vrijednost samo dokumenta čiji je vrijednost ključa **_id** proslijeđen.

```
router.post('/updateNote/:id', function(req, res) {
    console.log("update note is in process...");
    Note.findByIdAndUpdate(req.params.id, {
        $set :{ title : req.body.title, content : req.body.content } }, {new: true}, function (err, note){
            if (err){
                console.log(err);
                res.render('/editNote', {note:req.body});
            }
            res.redirect("/notes");
        });
});
```

Slika 23 API za ažuriranje vrijednosti jednog dokumenta

Preko Mongoose funkcije **findbyIdAndUpdate()** u kolekciji *notes* se pretražuje dokument s traženom vrijednosti primarnog ključa, te se postavlja i ažurira nova vrijednost ključeva *title* i/ili *content* u dokumentu, ovisno o tome što je korisnik izmijenio. Nakon što se izvrši izmjenjivanje dokumenta, na MongoDB Atlasu možemo vidjeti ažurirane vrijednosti ključeva u dokumentu kolekcije *notes*, te da je korisnik izmijenio vrijednost ključa *content*.



Slika 24 Prikaz ažuriranog dokumenta u kolekciji *notes*

I na kraju, ostalo je prikazati kako se dokumenti brišu iz kolekcije. Ako smo na web stranici *Moje bilješke* ispod neke bilješke kliknuli na poveznicu *Izbriši bilješku*, primarni ključ odabranog dokumenta se prosljeđuje u putanju, te se bilješka izbriše putem API. U varijablu *note* putem zahtjeva spremamo tijelo, te koristeći Mongoose funkciju **findByIdAndRemove()**, dokument se briše iz kolekcije. Nakon toga smo preusmjereni na istu web stranicu, ali ovaj put ćemo vidjeti da dokumenta više nema, a ako pogledamo u bazu podataka, također dokumenta više neće biti.

```
router.get("/deleteNote/:id", (req,res,next)=>{
    var note = new Note(req.body);
    Note.findByIdAndRemove(req.params.id ,{useFindAndModify : false}, (err, note)=> {
        if(err) console.log(err)
        console.log("IZBRISANA BILJESKA: ", note);
    })
  res.redirect('/notes');
})
```

Slika 25 Brisanje dokumenta iz kolekcije putem API

Na isti se način izvršavaju CRUD operacije i za dokumente u kolekciji *reminders*.

5.3.6. Odjava

Odjava korisnika iz aplikacije se odvija vrlo jednostavno uz pomoć Passport-a. Klikom na gumb *Logout* koji je na početnoj stranici, poziva se putanja */logout*, poziva se Passport funkcija **logout()**, te se vraćamo ponovno na Login stranicu.

```
router.get('/logout', function(req, res){
    req.logout();
    res.redirect('/');
    console.log("logout", req.user);
});
```

Slika 26 Odjava korisnika

6. Zaključak

Relacijske baze podataka još će dugo godina dominirati u području razvoja web aplikacija, no nerelacijske baze podataka pokazale su se kao dobar izbor koji nadograđuje nedostatke relacijskih baza podataka, kao što su rad s podacima širokog opsega, skalabilnost, fleksibilnost baza podataka, horizontalno skaliranje, te rad na distribuiranim sustavima. Pritom valja imati na umu da i nerelacijske baze podataka imaju nedostatke u odnosu na relacijske baze podataka, pa tako ne podržavaju nužno ACID svojstva te ne postoji jedan upitni jezik koji koriste sve NoSQL baze podataka.

Pravilan odabir modela baze podataka ili SUBP-a , u konačnici zavisi o zahtjevima web aplikacije koja se razvija. Količina podataka, struktura podataka, broj korisnika, brzina obrade podataka, skalabilnost itd. - sve su to bitne komponente koje treba držati na umu prilikom odabira SUBP-a, kako bi web aplikacija funkcionirala dugoročno i kvalitetno. Prvo se treba opredijeliti za model baze podataka, dakle relacijski ili nerelacijski, a onda treba odabrati i koji točno SUBP će najbolje odgovarati aplikaciji.

U ovom je završnom radu izrađen primjer web aplikacije za pisanje, uređivanje i brisanje bilješki i podsjetnika koristeći MongoDB SUBP. MongoDB je vrlo jednostavna za korištenje i osobama bez mnogo iskustva u bavljenju s bazom podataka, brza je i ima dobro razvijenu Mongoose biblioteku koja olakšava programerima razvoj web aplikacija u JavaScript-u.

Popis slika

Popis tablica

Popis priloga

Reference

[1] »Riverbed,« [Mrežno]. Available: https://www.riverbed.com/faq/how-does-web-application-work.html. [Pokušaj pristupa 9 kolovoz 2021.].

[2] »Britannica,« 18 svibanj 2020.. [Mrežno]. Available: https://www.britannica.com/technology/database. [Pokušaj pristupa 5 kolovoz 2021.].

[3] »MongoDB,« MongoDB, 2021.. [Mrežno]. Available: https://www.mongodb.com/. [Pokušaj pristupa 12. kolovoz 2021.].

[4] »Wikipedia,« rujan 2021.. [Mrežno]. Available: https://en.wikipedia.org/wiki/Relational_database. [Pokušaj pristupa 9. rujan 2021.].

[5] K. Berg, T. Seymour i R. Goel, »History Of Databases,« *International Journal of Management & Information Systems,* prosinac 2012..

[6] L. Yen, »Datamation,« 15. kolovoz 2021.. [Mrežno]. Available: https://www.datamation.com/big-data/current-database-trends/. [Pokušaj pristupa 2. rujan 2021.].

[7] »Db-engines,« kolovoz 2021.. [Mrežno]. Available: https://db-engines.com/en/ranking. [Pokušaj pristupa 12. kolovoz 2021.].

[8] M. Pavlić, »Relacijska metoda,« u *Informacijski sustavi*, Zagreb, Školska knjiga, 2011., pp. 171-172.

[9] N. Jatana, S. Puri, M. Ahuja, I. Kathuria i D. Gosain, »A Survey and Comparison of Relational and Non-Relational Database,« *International Journal of Engineering Research & Technology,* kolovoz 2012..

[10] »Wikipedia,« Wikipedia, 30. kolovoz 2021.. [Mrežno]. Available: https://en.wikipedia.org/wiki/ACID. [Pokušaj pristupa 30. kolovoz 2021.].

[11] S. Srivastava, »Appinventiv,« 4. lipanj 2021.. [Mrežno]. Available: https://appinventiv.com/blog/top-web-app-database-list/. [Pokušaj pristupa 30. kolovoz 2021.].

[12] A. Stojanović, »Osvrt na NoSQL baze podataka - četiri osnovne tehnologije,« *Polytechnic and design,* 2016..

[13] »MongoDB Documentation,« MongoDB, [Mrežno]. Available: https://docs.mongodb.com/. [Pokušaj pristupa 27 kolovoz 2021.].

[14] J. Hansen, »Passportjs,« 2020.. [Mrežno]. Available: http://www.passportjs.org/. [Pokušaj pristupa 27. srpanj 2021.].

[15] »EJS,« EJS, 2021.. [Mrežno]. Available: https://ejs.co/. [Pokušaj pristupa 2 rujan 2021.].

[16] C. Győrödi, R. Győrödi i R. Sotoc, »A Comparative Study of Relational and Non-Relational Database Models in a Web- Based Application,« *International Journal of Advanced Computer Science and Applications,* 2015..