



UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET



Marko Đorđević

Sistemi za upravljanje bazama podataka – Seminarski rad

Tema: Cloud baze podataka Google Cloud Firestore

Profesori:

Doc. dr Aleksandar Stanimirović

Student:

Marko Đorđević, br. ind. 1168

Niš, Jun 2021.

Sadržaj

1. Uvod	3
2. SQL baze podataka i NoSQL baze podataka	4
3. Google Cloud Firestore.....	6
3.1. Modeli podataka kod Google Cloud Firestore baze podataka	6
3.1.1. Dokumenti	6
3.1.2. Kolekcije	8
3.1.3. Način struktuiranja podataka	10
3.2. Google Cloud Firestore lokacije	13
3.3. Lokacije regiona u kojima možemo patiti podatke u Google Cloud Firestore bazi podataka.....	13
3.4. Pretraživanje Cloud Firestore baze podataka	14
3.4.1. Primer kreiranja kolekcije i pretraživanje dokumenata unutar kolekcije	15
3.4.2. Primer kreiranja upita za pretraživanje podataka u Google Cloud Firestore bazi podataka	18
3.4.3. Ažuriranje dokumenata kod Google Cloud Firestore baze podataka	22
3.5. Indeksi kod Google Cloud Firestore baze podataka.....	24
3.5.1. Kompozitni indeksi	25
3.5.2. Veličina indeksa i limiti	27
3.6. Offline režim rada Cloud Firestore baze podataka	28
3.6.1. Praktični primer offline režima rada Cloud Firestore Baze podataka.....	30
3.7. Kreiranje i autentifikacija korisnika na Google Cloud Firestore bazu podataka	36
3.7.1. Autentifikacija korisnika korišćenjem email i lozinke	37
3.7.2. Kreiranje korisničkih naloga korišćenjem email i lozinke	38
3.8. Cena korišćenja.....	40
3.9. Najbolje prakse korišćenja.....	41
4. Zaključak	42

1. Uvod

Živimo u svetu u kojem se količina podataka uveliko povećala u raznim oblastima. Web sajtovi, prate aktivnost korisnika, prodavci prikupljaju istoriju kupovine kupaca, pametni uređaji prikupljaju podatke o korisničkim aktivnostima, na primer otkucaji srca, obrasce spavanja, i tako dalje. Sa većim brojem podataka javlja se problem skladištenja takvih podataka. Podaci su često raznoliki i ne mogu se skladištiti u relacionim bazama podataka. Sa većim i bržim razvojem cloud tehnologija, sve više kompanije i pojedinci okreću se čuvanju podataka u oblaku, cloud-u. Ideja se zasniva na tome da svi podaci koji su preko potrebni budu dostupni u svakom trenutku, naravno uz prisustvu internet konekcije.

Cloud baze podataka su baze koje se nalaze na nekoj od cloud platformi i čiji je pristup obezbeđen kao servis. Cloud baze podataka imaju iste funkcionalnosti kao i obične baze, samo što je implementacija kao i fizička lokacija baze transparentna za korisnika. Prednost ovih baza nad običnim bazama jeste pre svega lakoća korišćenja, jer svaka cloud baza ima obezbeđen API ili web interfejs (konzola) od strane kompanije čija je cloud platforma. Takođe pošto se koriste resursi kompanije, cloud baze podataka nude veliku skalabilnost, jer se prostor koji je obezbeđen za bazu može povećavati u runtime-u, a korisnici plaćaju samo onoliko koliko baza stvarno zauzima. Još jedna prednost jeste i mogućnost oporavka, jer je svaka baza backup-ovana na više različitih servera, a backup se automatski radi od strane cloud platforme.

Kao i obične baze podataka tako i Cloud baze podataka mogu biti relacione i ne relacione odnosno SQL i NoSQL. Relacione baze inicijalno nisu zamišljene i dizajnirane da se koriste na distribuiranim sistemima, kao i to da su moderne relacione baze pokazale loše performanse kod veoma zahtevnih sistema, tako su NoSQL baze usvojene kao podrazumevani deo svake cloud platforme. NoSQL baze podataka su dizajnirane da podrže veća opterećenja kod čitanja i upisa, kao i to da se lako mogu skalirati, čime prirodno odgovaraju cloud sistemima.

U ovom seminarskom radu kao primer cloud baze podataka koristićemo Google Cloud Firestore NoSQL bazu podataka. Obradićemo teme kao što su način skladištenja podataka, obradićemo indekse kod ove baze podataka, takođe obradićemo offline režim rada Cloud Firestore baze podataka, ali pre svega toga govorićemo u razlici između SQL i NoSQL baze podataka.

2. SQL baze podataka i NoSQL baze podataka

Baza podataka je softver koji nam omogućava lak pristup, upravljanje, modifikovanje, ažuriranje, kontrolu i organizovanje podataka. Način na koji želimo da skladištimo informacije može uticati na to koju vrstu baze podataka odaberemo. Postoje dve glavne kategorije baza podataka, relacione, SQL baze podataka i ne-relacione, NoSQL, baze podataka.

Relaciona baza podataka je poseban tip baze podataka kod kojeg se organizacija podataka zasniva na relacionom modelu. Podaci se u ovakvim bazama organizuju u skup relacija između kojih se definišu određene veze. Relacija se definiše kao skup n-torki sa istim atributima, definisanih nad istim domenima iz kojih se mogu uzimati vrednosti. U relacionim bazama podataka, svaka relacija mora da ima definisan primarni ključ, koji predstavlja atribut pomoću kojeg se jedinstveno identifikuje svaka n-torka. Svaka relacija se može predstaviti u tabelarnom obliku, što je i prva asocijacija na relacione baze podataka. Svaki red tabele predstavlja n-torku relacije, svaka kolona, naziv kolone, vrednost kolone se kod relacija prevodi kao atribut, naziv atributa, vrednost atributa. Tabela je u stvari bazna relacija, skup naziva kolona predstavlja relacionu šemu, dok pogled, rezultati upita predstavljaju izvedene relacije.

Da bismo videli kako izgleda jedna relacija, odnosno tabela kod Relacione baze podataka, iskoristićemo primer iz prethodnog seminarskog rada¹, kada smo govorili o Oracle bazi podataka, jednoj od najpopularnije i najkorišćenije relacione baze podataka (slika 1.).

ID	ACTIVE	WEBACTIVE	ACTIVE_DOC	BARCODE	CODE	INPUTPRICE	INPUTPRI...	MANUF...	MANUFNA...	NAME	ALTERNAM...	PRICE	PRICECU...	PRICEVIS...	TAXID	TYPE	SUPPLIER...	UNITNAME
1	181 y	y	n	(null)	1252529921	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
2	182 y	y	n	(null)	1252529922	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
3	183 y	y	n	(null)	1252529923	0	(null)	(null)	Makita	Akumula... probe!!!!2	(null)	0	(null)	a	2 x	n	Kom.	
4	184 y	y	n	(null)	1252529924	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
5	185 y	y	n	(null)	1252529925	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
6	186 y	y	n	(null)	1252529926	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
7	187 y	y	n	(null)	1252529927	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
8	188 y	y	n	(null)	1252529928	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
9	189 y	y	n	(null)	1252529929	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
10	190 y	y	n	(null)	1252529930	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
11	191 y	y	n	(null)	1252529931	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
12	192 y	y	n	(null)	1252529932	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
13	193 y	y	n	(null)	1252529933	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
14	194 y	y	n	(null)	1252529934	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
15	195 y	y	n	(null)	1252529935	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
16	196 y	y	n	(null)	1252529936	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
17	197 y	y	n	(null)	1252529937	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
18	198 y	y	n	(null)	1252529938	0	(null)	(null)	Akumula...	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
19	199 y	y	n	(null)	1252529939	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
20	200 y	y	n	(null)	1252529940	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
21	201 y	y	n	(null)	1252529941	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
22	202 y	y	n	(null)	1252529942	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
23	203 y	y	n	(null)	1252529943	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
24	204 y	y	n	(null)	1252529944	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
25	205 y	y	n	(null)	1252529945	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
26	206 y	y	n	(null)	1252529946	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
27	207 y	y	n	(null)	1252529947	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
28	208 y	y	n	(null)	1252529948	0	(null)	(null)	GARDEN...	Elektri...	(null)	0	(null)	a	2 x	n	Kom.	
29	209 y	y	n	(null)	1252529950	0	(null)	(null)	GARDEN...	Pneumat...	(null)	0	(null)	a	2 x	n	Kom.	
30	210 y	y	n	(null)	1252529951	0	(null)	(null)	GARDEN...	GT450 E...	(null)	0	(null)	a	2 x	n	Kom.	
31	211 y	y	n	(null)	1252529952	0	(null)	(null)	GARDEN...	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
32	212 y	y	n	(null)	1252529953	0	(null)	(null)	GARDEN...	SDS-Plo...	(null)	0	(null)	a	2 x	n	Kom.	
33	213 y	y	n	(null)	1252529954	0	(null)	(null)	GARDEN...	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	
34	214 y	y	n	(null)	4326	0	(null)	4326	Makita	Uvodna ...	(null)	0	(null)	a	4 x	n	Kom.	
35	215 y	y	n	(null)	1252529955	0	(null)	(null)	Makita	Akumula...	(null)	0	(null)	a	2 x	n	Kom.	

Slika 1: Primer pamćenja informacija o proizvodu u Oracle bazi podataka

Za proizvod je potrebno pamtit koji je identifikacioni broj proizvoda, da li je aktivan proizvod (kao primer uzet je realni slučaj web aplikacije za prodaju mašinskih alata), zatim da li je vidljiv proizvod u web aplikaciji, barcode proizvoda, naziv, tip proizvoda i tako dalje. Ukoliko postoji potreba da dodamo informacije o jednom proizvodu na primer za proizvod čiji je id 183 želimo da pamtimo i informaciju o taksi za taj proizvod (taksa proizvoda predstavlja „pdv“ proizvoda). Da bismo to omogućili neophodno je dodati potpunu novu kolonu u kojoj ćemo čuvati informacije o taksama proizvoda i prilikom unosa, jedino polje će biti upisano za proizvod čiji

¹ “Interna struktura i organizacija indeksa kod Oracle baze podataka“ Marko Đorđević

je id 183, za sve ostale proizvode biće prazno polje. Ovakva pojava kod relacionih baza podataka je takva zbog toga što relaciona baza podataka ima specifičnu strukturu podataka koja se naziva šema. Šemom definišemo koji tip podataka ćemo pamtiti u bazi podataka, odnosno zahteva strogu strukturu. Stroga struktura relacione baze podataka omogućava našoj aplikaciji da zna kakvi podaci postoje, da zna koji je tip podataka i da primenjuje pravila kao što su zahtevi da podaci budu jedinstveni ili nametanje tipa podataka koji se čuvaju i tako dalje. Šema, dizajn, prisiljava podatke da u svakom redu imaju iste karakteristike, što znači da nisu baš fleksibilni osim ako ne promenimo šemu baze podataka.

Promena šema relacione baze podataka može biti vrlo „ometajući“ proces (cela baza podataka je nedostupna korisnicima), posebno ukoliko je baza podataka koja sadrži ogromnu količinu podataka, jer zahteva pokretanje skripti za promenu šeme i njenu pažljivu koordinaciju sa promenama koda u aplikacijama.

Kao rešenje ovakvih problema javile su se nerelacione baze podataka dokumenata kao što je Firestore gde ne moramo da brinemo o promenama šeme u bazi podataka ili zastoju kao rezultat toga. Kod ovakvog tipa baze podataka, dopuštena je velika fleksibilnost korisnika da pristupaju, dodaju i skladište samo potrebne podatke bez velikog rasipanja podataka.

Uopšteno govoreći, nerelaciona baza podataka čuva informacije u drugačijem formatu od relacione baze podataka. Postoje 4 glavne kategorije nerelacionih baza podataka koje se najčešće koriste a to su :

1. Column-Family
2. Document(Firestore)
3. Key-value
4. Graph

Budući da se u ovom seminarskom fokusiramo na Google Cloud Firestore u ovom odeljku ćemo istražiti način skladištenja podataka i bitne karakteristike Google Cloud Firestore baze podataka.

3. Google Cloud Firestore

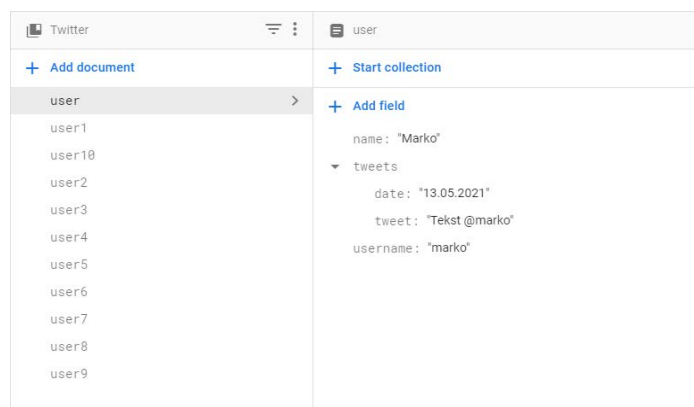
Google Cloud Firestore je fleksibilna, skalabilna NoSQL baza podataka kreirana od strane Googlea. Firestore baza podataka je deo većeg sistema Google Firebase-a. Nastanak ove baze podataka kreće u oktobru 2017. godine, preteća ovog sistema jeste Real Time Database. Uvođenjem Cloud Firestore baze podataka Google je svetu ponudio najbolje rešenje za baze podataka tako da pokrije neka ograničenja Firebase baze podataka u pogledu performansi, funkcionalnosti i skalabilnosti. Firestore baza omogućava da podaci budu sinhronizovani između svih klijentskih aplikacija pomoću realtime listeners-a i obezbeđuje offline podršku za mobilne i web aplikacije tako da aplikacije mogu da rade nezavisno od kašnjenja na mreži ili internet konekcije. Pošto Firestore baza kešira podatke koje naša aplikacija aktivno koristi, ona omogućava da aplikacija upisuje i čita podatke iako aplikacija nije povezana na internet. Kada se aplikacija ponovo poveže na internet sve lokalne promene se sinhronizuju sa bazom.

3.1. Modeli podataka kod Google Cloud Firestore baze podataka

Firestore je NoSQL baza podataka kod kojih se podaci čuvaju u okviru dokumenata. Za razliku od SQL baze podataka, ne postoje tabele ili redovi. Umesto toga, podaci se skladište u dokumentima koji su organizovani u kolekcije. Svaki dokument sadrži skup parova ključ/vrednost. Firestore je optimizovan za čuvanje velikih kolekcija malih dokumenata. Svi dokumenti se moraju čuvati u zbirkama odnosno kolekcijama. Dokumenti mogu da sadrže podkolekcije i ugnježdene objekte, a oba mogu sadržati primitivna polja poput nizova ili složene objekte poput lista. Kolekcije i dokumenti se implicitno kreiraju u Firestore-u. Jednostavno dodelimo podatke dokumentu u kolekciji. Ukoliko kolekcija ili dokument ne postoji, Firestore će ih kreirati.

3.1.1. Dokumenti

Kod Firestore baze podataka, jedinica za skladištenje podataka je dokument. Dokumenti se identifikuju nazivom dokumenta. Primer dokumenta data je na slici 2.



Slika 2: Prikaz dokumenta *user* skladištenog u Firestore bazi podataka

U ovom primeru dokumenti su organizovani kao korisnici Twitter platforme (naziv kolekcije) gde se u svakom dokumentu pamte polja, odnosno vrednosti koju pamti dokument. Identifikacija dokumenta kao što mozemo videti jeste *user* dokument, pa zatim dokument *user/* i tako redom.

Tipovi podataka koji su podržani za pamćenje informacija kod Firestore baze podataka dati su u tabeli ispod.

Tip podatka	Način sortiranja zapamćenih podataka	Informacije
Array	Prema vrednostima elemenata	<ul style="list-style-type: none"> *Nije dozvoljeno pamćenje drugih nizova unutar nizova. *Unutar niza, elementi zadržavaju navedeni položaj nakon dodele *Upoređivanje niza se vrši tako što se prvo poredi prvi element niza, ukoliko su jednaki poredi se sledeći element i tako redom.
Boolean	False<true	-
Bytes	Na osnovu redosleda bajtova	<ul style="list-style-type: none"> *Pamćenje do 1Mib podataka *Upiti uzimaju u obzir samo prvih 1500 bajtova
Date and time	Hronološki	*Pamti informacije o vremenu sa preciznošću do mikrosekunde
Floating-point number	Numerički	*Pamćenje 64b reči sa dvostrukom tačnošću
Geographical point	Prema geografskoj širini, pa zatim po dužini	*Trenutno se ne preporučuje korišćenje ovog tipa podatka zbog ograničenja upita i bolje je pamtiti kao numeričke vrednosti
Integer	Numerički	*64b označen broj
Map	Na osnovu ključeva, pa zatim na osnovu vrednosti	<ul style="list-style-type: none"> *Predstavlja objekat ugrađen u dokument.Prilikom indeksiranja u upitima je moguće navesti samo imena potpolja u upitu. *Ukoliko, prilikom poređenja, objekata ukoliko su identični objekti ali im je broj polja različita kao mera sortiranja biće parametar broj podpolja unutar objekata
Null	Ne sortira elemente	-
Reference	Sortira se prema kolekciji i po dokument id-u	-

Text string	UTF-8 encoded uređen na osnovu bajtova	*Pamćenje do 1Mib podataka *Upiti uzimaju u obzir samo prvih 1500 bajtova UTF-8 reprezentacije
-------------	--	---

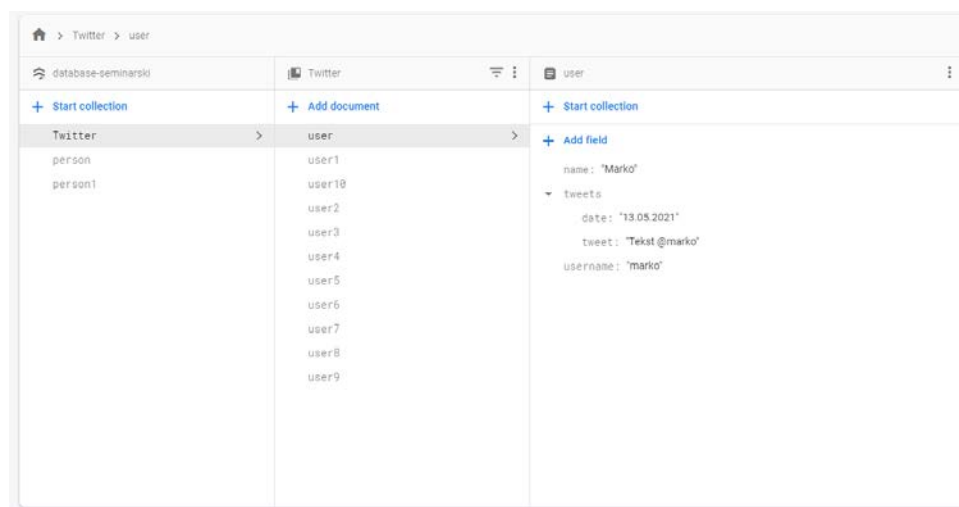
Firestore prilikom izvršavanja upita ima prioritete tipove koje prvo pretražuje:

1. Null vrednost
2. Boolean vrednosti
3. Integer i Double vrednosti sortirane numerički
4. Date vrednosti
5. Text String vrednosti
6. Byte vrednosti
7. Cloud Firestore reference
8. Geographical point vrednosti
9. Array vrednosti
10. Map vrednosti

Sami dokumenti kod Firestore baze podataka su u stvari JSON fajlovi i tako se mogu posmatrati. Jedina razlika je ta što dokumenti podržavaju neke dodatne tipove podataka i ograničeni su na 1MB, tako da se mogu tretirati kao jednostavniji JSON fajlovi.

3.1.2. Kolekcije

Dokumenti žive u kolekcijama, koje su jednostavno kontejneri za dokumente. Prikaz kolekcije Twitter, o kome je u pređašnjoj temi bilo reči, data je na slici 3.

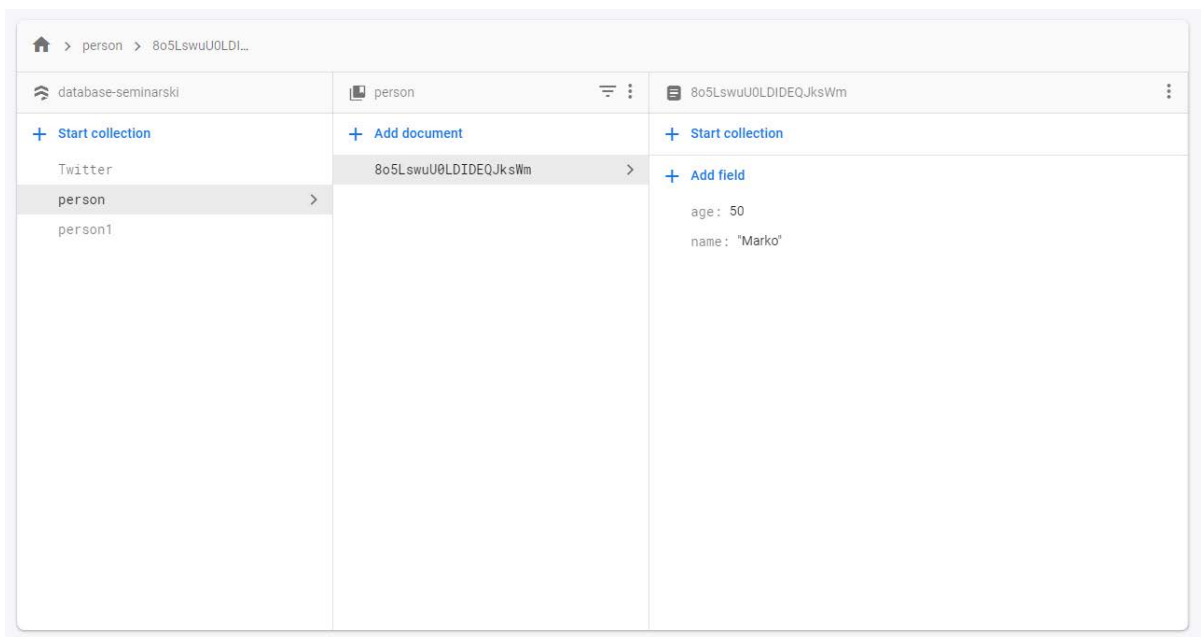


Slika 3: Prikaz korisnički kreirane kolekcije Twitter

Na slici 3. prikazana je jedna kolekcija koja se zove Twitter u kome pamtimo dokumente koja sadrže informacije o više različitih korisnika.

Za razliku od relacione baze podataka, Firestore je baza podataka koja ne poseduje šemu baze podataka, tako da korisnici imaju potpunu slobodu nad poljima u dokumentima kao i tipove podataka koje pamtimo u dokumentima. Na primer ukoliko za ovaj naš primer želimo da promenimo polje *tweets*, mi možemo promeniti to polje da bude niz objekata tipa *tweets*, kao što je i prikazano na primeru. Ovo „ograničenje“ nekada nije baš i pogodno zbog pretraživanja same baze, zato je dobra praksa koristiti ista polja i tipove podataka u više dokumenata, tako da ih možemo lakše pretražiti. Kolekcija sadrži samo dokumente, ne može direktno sadržati podatke i nije moguće kreirati kolekciju unutar kolekcije.

Imena dokumenata u kolekciji su jedinstvena. Možemo obezbediti sopstvene ključeve, identifikacije dokumenata, kao što se može videti na slici 3, gde smo kao ključ koristili ključnu reč *user* a u nastavku dodali id korisnika. Pored sopstvenog dodeljivanja ključeva dokumenata ukoliko ne navedemo naziv dokumenta u kom se pamte podaci, Firestore će automatski kreirati nasumične ID-ove. Primer je prikazan na slici 4.



Slika 4: Primer automatskog dodeljivanje identifikatora dokumenta kod Firebase baze podataka

Kolekcija postoji ukoliko postoje dokumenti u kolekciji, ukoliko obrišemo sva dokumenta jedne kolekcije, kolekcija više ne postoji.

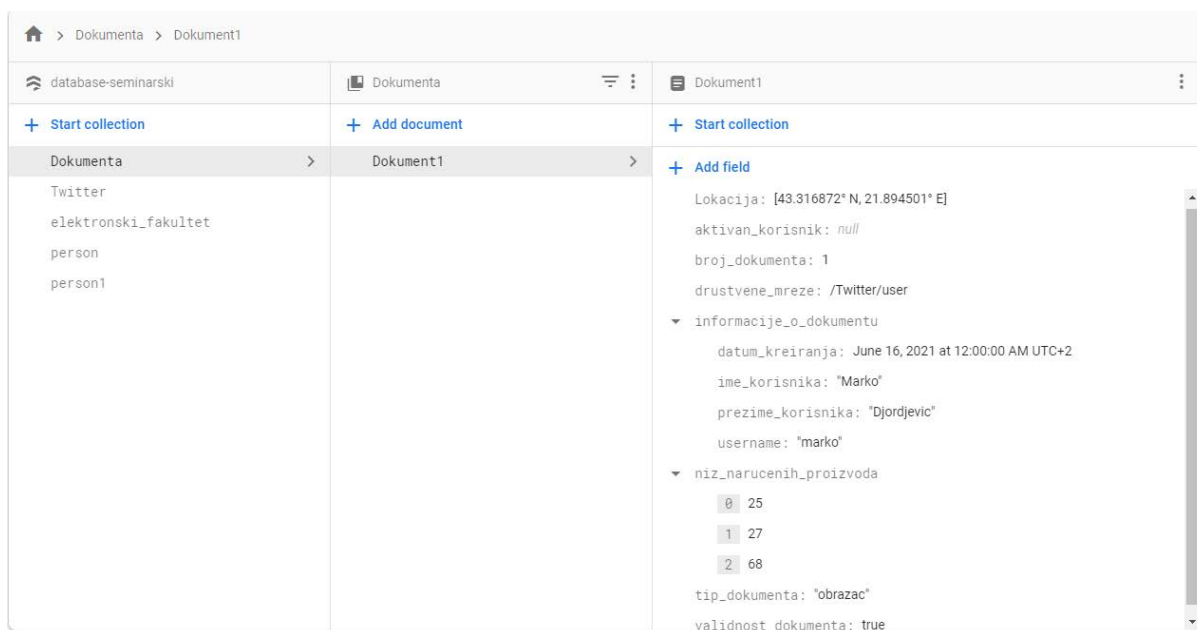
3.1.3. Način struktuiranja podataka

Kod Cloude Firestore baze podataka podatke možemo strukturirati na 3 načina:

1. Pamćenje podataka u okviru dokumenta
2. Pamćenje podataka u okviru više kolekcija
3. Pamćenje podataka u potkolekcijama u dokumentima

3.1.3.1. Struktuiranje podataka u okviru dokumenta

Kod ovog načina pamćenja podataka, pravilnije je reći struktuiranja, korisnik može kreirati poseban dokument za pamćenje informacija bez vođenja računa o samoj strukturi baze podataka. Primer ovakvog načina pamćenja data je na slici 5.



Slika 5: Primer pamćenja „sirovih“ podataka u okviru dokument

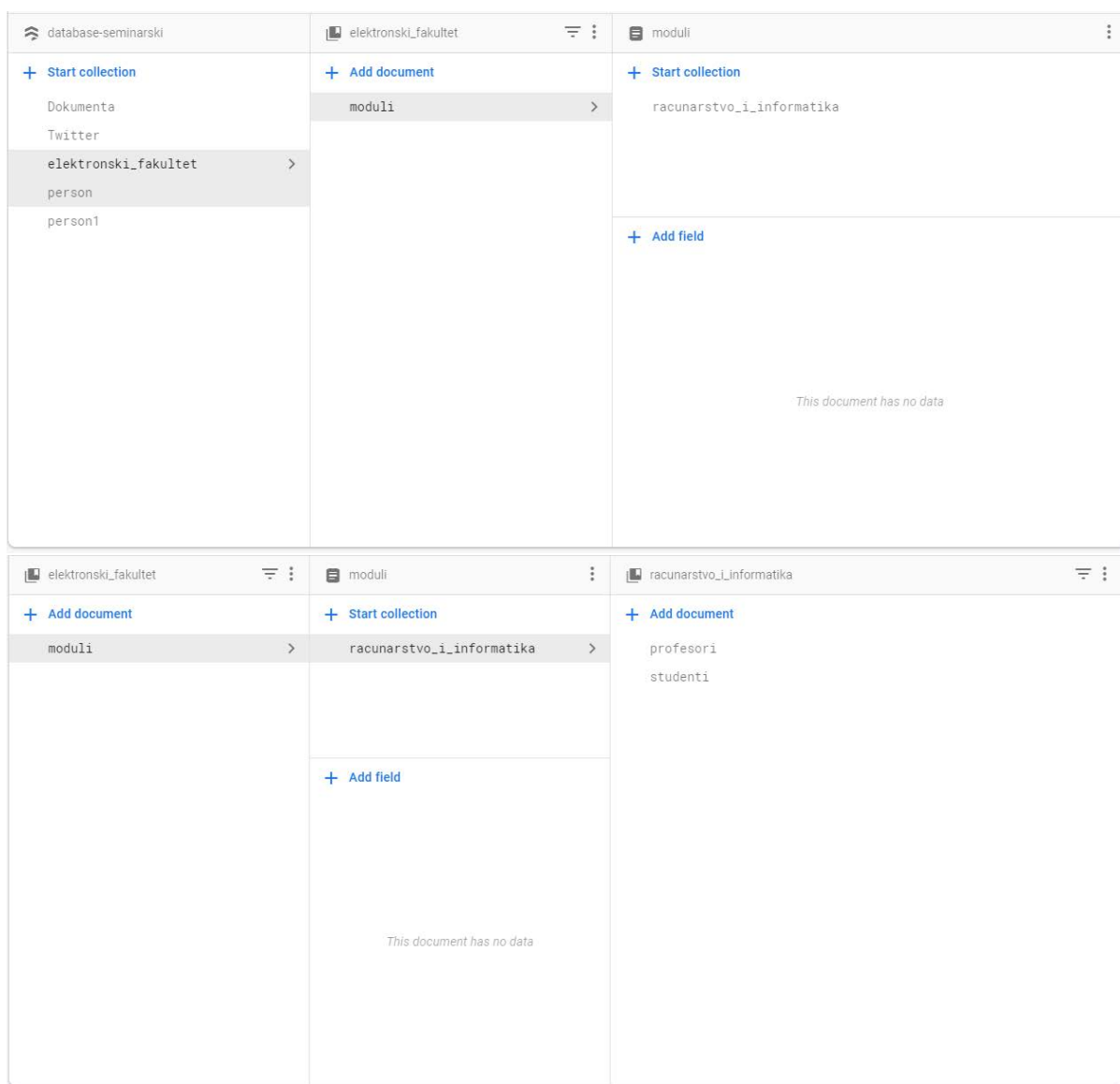
Na slici 5 je prikazan način pamćenja „sirovih“ podataka u dokument. U ovom primeru kreirali smo dokument pod nazivom *Dokument1* u kome pamtimo informacije o porudžbenicama. Dokument se sastoji iz svih tipova podataka koje je moguće skladištiti u Firestore bazi podataka. Podaci koje se pamte jesu Lokacija kada je dokument kreiran, da li je korisnik aktivan, broj dokumenta, referencu ukoliko postoji u kolekciji društvene mreže, zatim imamo niz naručenih proizvoda, zatim tip podatka, i da li je dokument validan.

Prednost ovakvog skladištenja podataka je u tome ukoliko imamo jednostavne, fiksne liste podataka koje želimo da sačuvamo u svojim dokumentima, ovim lako postavljamo i pojednostavimo strukturu podataka. Tipičan primer korišćenja ovakvog načina pamćenja podataka jeste prikazan na slici 5. Struktura dokumenata je za svaku narudžbinu relativno ista.

Mana ovakvog pamćenja podataka jeste ta što narušavamo skalabilnost baze podataka, posebno ako se potreba za pamćenjem podataka povećava. Sa većim ili rastućim podacima, dokument takođe raste što može dovesti do sporijeg vremena preuzimanja dokumenata.

3.1.3.2. Strukturiranje podataka u okviru više podataka

Ovakav način struktuiranja podataka je pogodan za kreiranje kolekcije u dokumentima kada posedujemo podatke koji bi se vremenom mogli proširiti. Primer ovakog pamćenja podataka je dat na slici 6.



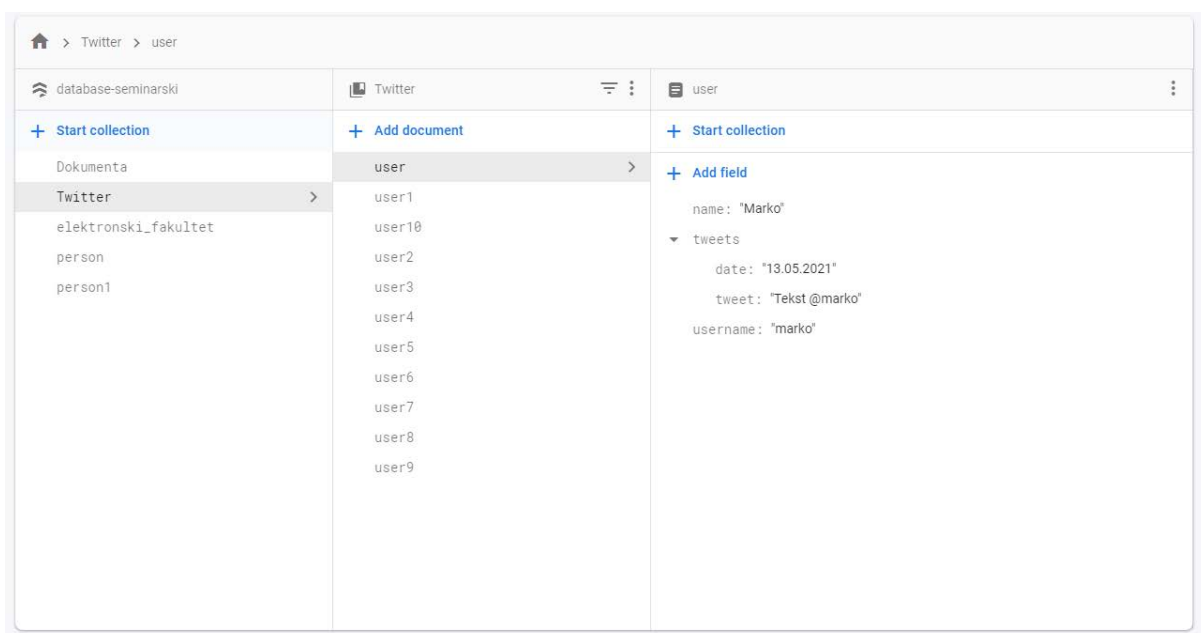
Slika 6: Strukturiranje podataka u okviru više dokumenata

U okviru kolekcije *elektronski_fakultet*, mi pamtim o i informacije o *modulima* na elektronskom fakultetu, pa zatim za svaki modul na fakultetu pamtim dokumente o *profesorima* i *studentima* određenog modula. Ovim smo stvorili hijerarhijsku strukturu dokumenata, pravilnije je reći kolekcija.

Prednost ovakvog struktuiranja podataka je ta da ukoliko veličina lista podataka raste, veličina samog dokumenta se ne menja. Ovim dobijamo mogućnost upita za podkolekcije i možemo kreirati upite za kolekcije unutar podkolekcije. Jedino ograničenje kod ovakvog tipa pamćenja informacija je ta što nije jednostavno brisanje kolekcija. Najpre je potrebno obrisati dokumente pa kolekcije u najdubljem delu stabla, pa zatim brisati kolekcije na nivo iznad. U našem slučaju potrebno bi bilo obrisati kolekciju *racunatstvo i informatika* pa zatim kolekciju *moduli*, u suprotnom neće biti pravilno obrisane kolekcije.

3.1.3.3. Strukturiranje podataka u potkolekcijama u dokumentima

Ovakav način pamćenja podataka unutar kolekcija nudi korisnicima mogućnost organizacije različitih skupova podataka. Primer je prikazan na slici 7.



Slika 7: Strukturiranje podataka u potkolekcijama u dokumentima

U ovom primeru imamo kolekciju dokumenata u kojima ćemo pamtit i informacije vezane za porudžbine klijenata (kolekcija *dokumenta*) i imamo kolekciju *Twitter* koja je imitacija društvene mreže. Podsećanja radi, u dokumentima imamo polje koje se odnosi na društvene mreže i u tom polju imamo podatak *username* koji se odnosi na korisničko ime na društvenoj mreži. Skupovi podataka su odvojeni i time pravimo bolju strukturu skladištenja. Kao prednost ovakvog načina pamćenja podataka jeste pamćenja relacija many-to-many. Ograničenje ovakvog pristupa jeste što pretraga podataka može biti složenije kako baza podataka raste.

3.2. Google Cloud Firestore lokacije

U poglavlju 3.1 obradili smo temu skladištenja podataka, kako se pamte podaci kod Google Cloud Firestore baze podataka, ali postavlja se glavno pitanje gde se u stvari smeštaju ti podaci.

Da bimo koristiti Firestore bazu podataka, pre svega je potrebno kreirati projekat u okviru Google Firebase platforme, a nakon toga kreirati i samu bazu podataka. Prilikom kreiranja baze podataka neophodno je navesti ime baze podataka (projekta) i zatim izabrati lokaciju na kojoj će se pamtit i baza podataka. Odabir lokacije na kojoj će se skladištiti baze podataka su od ključne važnosti, da bismo smanjili kašnjenja i povećali dostupnost, podatke treba pamtit i u blizini korisnika i usluga koje su nam potrebne. Prilikom kreiranja baze podataka, Firestore svojim korisnicima dodeli lokaciju za smeštanje baze podataka u zavisnosti od lokacije gde se trenutno nalazi korisnik. Menjanje lokacije baze podataka nije moguće nakon kreiranja same baze podataka pa je zato neophodno dobro odlučiti na kojoj lokaciji će se pamtit i podaci.

Podatke u Cloud Firestore bazi podatka možemo skladištiti u više regiona ili na lokacijama u regionu. Firebase podržava lokacije resursa date u tabeli 1. Treba imati na umu da aplikacija koja koristi Cloud Firestore bazu podataka i skladišti podatke u više regiona će smatrati u smislu da već postoji App Engine sa lokacijom ili u *us-central* ili *europa-west*.

Naziv lokacije više regiona(eng. Multi-Region)	Opis lokacije više regiona(eng. Multi-Region)	Sastavni regioni
Eur3	Evropa	Europe-west1, europa-west4
Nam5	United States	Us-central1, Us-central2(Oklahoma-private GCP region)

Tabela 1: Prikaz mogućih lokacija baza podataka kod Google Cloud Firestore baze podataka

Firestore svojim korisnicima takođe nudi i mogućnost izbora više regiona i time postizemo dostupnost i trajnost baze podataka. Lokacije sa više regiona mogu izdržati gubitak čitavih regiona i održati dostupnost bez gubitka podataka.

3.3. Lokacije regiona u kojima možemo patiti podatke u Google Cloud Firestore bazi podataka

Regionalna lokacija je određeno geografsko područje poput na primer Juzne Karoline. Podaci na regionalnoj lokaciji se kopiraju u više zona unutar regiona. Sve regionalne lokacije odvojene su od ostalih regionalnih lokacija najmanje 100 milja. Pravilnim izborom regionalne lokacije smanjujemo latenciju upisa u bazu podataka, kašnjenje ili smanjujemo kašnjenje ukoliko posedujemo multiregionalni odabir lokacije sa drugim resursima.

U Evropi postoje 4 glavne lokacije u kojima se čuvaju podaci i to su europa-west2(London), europa-west3(Frankfurt), europa-central12(Warsawa) i europa-west6(Zurich).

Vrlo bitno je napomenuti da u zavisnosti od odabira regiona zavisi i cena usluga koje korisnicima nude, tako da uvek prilikom pravljenja plana skladištenja povesiti racuna o regionima i cenama unutar regiona o kojima će kasnije biti više reči.

3.4. Pretraživanje Cloud Firestore baze podataka

Pretraživanje podataka kod NoSQL baze podataka razlikuje se u odnosu na SQL odnosno relacione baze podataka. Razlika je ta što SQL baze podataka koriste strukturirani jezik upita i poseduju unapred definisanu šemu. NoSQL baze podataka imaju dinamičke šeme za nestruktuirane podatke.

Relaciona baza podataka koriste strukturirani jezik upita (eng. Structured Query Language – SQL), kao što smo i rekli u prethodnom pasusu, za definisanje i manipulisanje podacima. SQL je jedna od najsvestranijih i široko korišćenih dostupnih opcija. SQL takođe zahteva da koristimo prethodno definisane šeme za određivanje strukture podataka pre nego što sa njima radimo. Pored toga, svi podaci moraju slediti istu strukturu. To može zahtevati značajnu pripremu unapred i to znači da ukoliko bi se desila promena strukture tada bi doslo do konflikta u celom sistemu. Sa druge strane NoSQL baza podataka, imaju dinamičke šeme za nestruktuirane podatke, a podaci se čuvaju na više načina. Mogu biti orijentisani na kolone, orijentisani na dokumente, zasnovani na grafovima ili organizovane kao skladište parova ključ-vrednost. Ovim se postiže jako velika fleksibilnost, možemo kreirati dokumente bez potrebe da prethodno definišemo njihovu strukturu, zatim svaki dokument se može razlikovati od baze do baze podataka i moguće je menjati strukturu pamćenja dokumenta.

Kod relacionih baze podataka da bi smo pretražili bazu podataka mi izvršavamo sledeću komandu:

```
SELECT kolona  
FROM ime_tabele  
WHERE uslov
```

Nakon izvršenja komande, upita, kao rezultat, baza podataka će vratiti informacije o traženim podacima iz baze podataka. Ovakva struktura upita je relativno ista kod skoro svih relacionih baze podataka. Kod Google Cloud NoSQL baza podataka postoje male izmene prilikom pretraživanja baze podataka.

Pre same pretrage podataka neophodno je pristupiti bazi podataka. Google Cloud Firestore bazu podataka se pristupa korišćenjem njenih biblioteka koje se nalaze u većini SDK za mobilne, web i desktop aplikacije. Da bismo koristili Firestore bazu podataka, pre svega je potrebno kreirati projekat u okviru Google Firebase platforme, a nakon toga kreirati i samu bazu. Prilikom kreiranja Firestore baze podataka, dobija se *.json* fajl koji se, u zavisnosti od tehnologije, sačuva u neki od foldera projekta. Ovim fajlom se naša aplikacija vezuje za naš projekat na Firebase platformi, a samim tim i Firestore bazu.

Kod Firestore baze podataka upiti su ekspresni, efikasni i fleksibilni. Moguće je kreirati upite za čitanje podataka na nivou dokumenta bez potrebe da se pročitaju i cele kolekcije ili

podkolekcije. Takođe moguće je dodati i sortiranje, filtriranje i limitiranje na upite. Kako ne bi morali da ponovo da čitamo celu bazu svaki put kada se desi neka promena, u aplikaciju se mogu dodati realtime *listeners*, koji se mogu vezati za neku konkretnu kolekciju ili za celu bazu i koji su deo Firestore biblioteke. Oni obaveštavaju aplikaciju svaki put kada se desi neka promena i aplikaciji šalje snapshot koji sadrži samo podatke koji su npr. dodati ili promenjeni.

U daljem tekstu pokazaćemo kako se izvršavaju upiti nad Google Cloud Firestore bazi podataka korišćenjem programskog jezika Python.

3.4.1. Primer kreiranja kolekcije i pretraživanje dokumenata unutar kolekcije

Kao što smo u prethodnom poglavlju (poglavlje 3.3) rekli da bi se povezali sa bazom podataka potrebno je preuzeti .json fajl koji se nalazi u sekciji Project settings/Service accounts/Firebase Admin SDK i klikom na dugme generate new private key dobijemo serviceAccountKey.json fajl koji postavimo u folder aplikacije. Ovim je omogućeno povezivanje na projekat na Firebase platformi.

Nakon što smo uspešno preuzeli fajl, da bi se povezali sa našom bazom podataka, u zavisnosti od programskog okruženja, neophodno je povezati se sa bazom podataka odnosno projektom. Pošto primeri će biti prikazani u programskom okruženju Python, da bi se pravilno povezali neophodno je da imamo instaliran paket *firebase_admin*. Nakon inicijalizacije okruženja neophodno je izvršiti sledeće komande:

```
import firebase_admin

from firebase_admin import credentials

from firebase_admin import firestore

cred = credentials.Certificate("serviceAccountKey.json")

firebase_admin.initialize_app(cred)
```

Nakon pokretanja ukoliko je došlo do greške prilikom povezivanja u konzoli pokazaće se greška. Funkcijom `credentials.Certificate` omogućavamo autentifikaciju na samu bazu podataka. Fajl `serviceAccountKey.json` izgleda kao na slici 8.

```
{
  "type": "service_account",
  "project_id": "database-seminarski",
  "private_key_id": "",
  "private_key": "",
  "client_email": "firebase-adminsdk-t1l7n@database-seminarski.iam.gserviceaccount.com",
  "client_id": "",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-t1l7n@database-seminarski.iam.gserviceaccount.com"
}
```

Slika 8: Izgled serviceAccountKey.json fajla

Zbog sigurnosti, uklonjena su polja, `private_key_id`, `private_key` i `client_id`.

Da bi smo kreirali kolekciju neophodno je definisati koje ćemo podatke pamtiti u dokumentu u okviru kolekcije. Sledeći kod ilustruje kreiranje jedne kolekcije u python programskom okruženju.

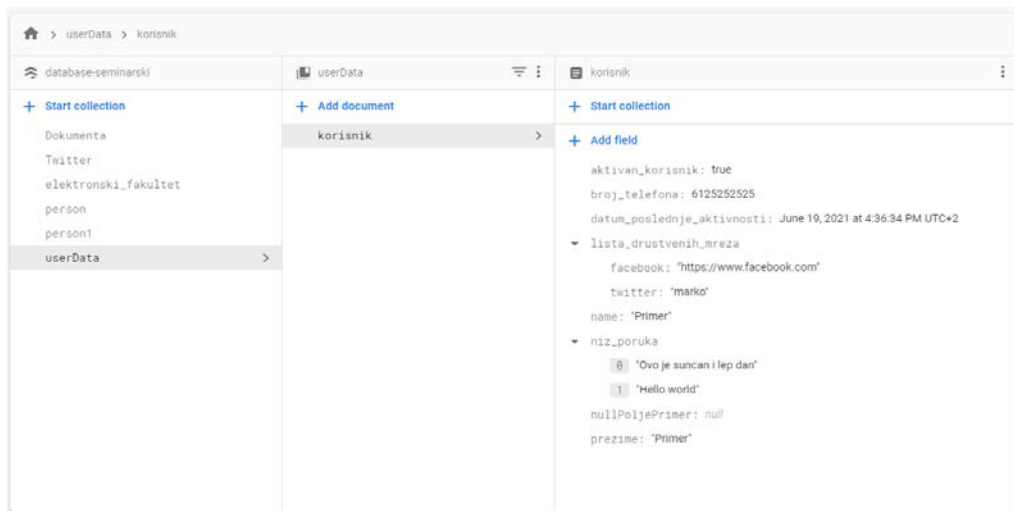
```
db = firestore.client()

data = {
    'name': 'Primer',
    'prezime': 'Primer',
    'lista_drustvenih_mreza': {
        'facebook': 'https://www.facebook.com',
        'twitter': 'marko'
    },
    'aktivan_korisnik': True,
    'datum_poslednje_aktivnosti': datetime.datetime.now(),
    'niz_poruka': ['Ovo je suncan i lep dan', 'Hello world'],
    'nullPoljePrimer': None,
    'broj_telefona': +6125252525
}

db.collection('userData').document('korisnik').set(data)
```

U ovom primeru kao nazvi dokumenta naveli smo ime korisnik, ukoliko pak izostavimo deo naredbe `document` u bazi podataka biće upisana vrednost koju sam Cloud Firestore baza podataka generiše odnosno ID dokumenta.

Nakon kreiranja ovakve kolekcije u Google Cloud Firestore konzoli prikazaće se kolekcija `userData` koja sadrži dokument `korisnik` koji sadrži definisane podatke. Rezultat izvršenja naredbi dat je na slici 9.



Slika 9: Izgled Firestore baze podataka nakon kreiranja kolekcije `userData`

Da bi smo ažurirali podatke u bazi podataka i da bi smo manipulisali dokumentima, neophodno je kreirati referencu na samu kolekciju ili na određeni dokument, kao na primeru u nastavku

```
userDataCollectionReference = db.collection('userData')
documents = userDataCollectionReference.stream()
for doc in documents:
    print(doc.id, " ", doc.to_dict())
print("UserData Collection reference: ", userDataCollectionReference)
userDataReference = db.collection('userData').document('korisnik')
print("Korisnik document reference: ", userDataReference)
doc = userDataReference.get()
if doc.exists:
    print('Document id :', doc.id, ' Document data: ', doc.to_dict())
else:
    print("No such document")
```

Nakon izvršenja prve naredbe dobijamo referencu na celu kolekciju `userData`. Da bismo videli koje dokumente pamti zadata kolekcija potrebno je izvršiti operaciju `stream()` i nakon toga obići svaki dokument iz povratne vrednosti funkcije i prikazujemo *dokument id* i sam sadržaj dokumenta. Da bi smo dobili informacije o tačno određenom dokumentu potrebno prilikom preuzimanje reference specificirati i tasan *id* dokumenta za koji želimo da preuzmemo informacije.

Firestore svojim korisnicima dozvoljava da pamte unutar dokumenata kolekcije, naš primer *Elektronskog fakulteta*. Da bismo dobili pregled svih podkolekcija unutar kolekcije neophodno je izvršiti sledeću komandu:

```
collections = db.collection('elektronski_fakultet').document('moduli').collections()

for collection in collections:
    for doc in collection.stream():
        print(doc.id, " : ", doc.to_dict())
```

Google Cloud Firestore podržava samo pretraživanje do 2 dubine kolekcija u stablu kolekcija, ukoliko kao u našem slučaju imamo potrebu za pretraživanjem do 4 dubine moramo navesti direktnu putanju do željenog dokumenta kao u sledećem primeru:

```
collections =
db.collection('elektronski_fakultet').document('moduli').collection('racunarstvo_i_informatika').document('profesori').collections()

for collection in collections:
    for doc in collection.stream():
        print(doc.id, " : ", doc.to_dict())
```

3.4.2. Primer kreiranje upita za pretraživanje podataka u Google Cloud Firestore bazi podataka

Upite koje kreiramo mogu se koristiti samo za pronalaženje podataka unutar dokumenata unutar jedne određene kolekcije ili podkolekcije. Primer pretraživanja kolekcije prikazane u prethodnom poglavlju je sledeći:

```
docs =
db.collection('userData').where('lista_drustvenih_mreza.twitter','==','marko').stream()

for doc in docs:
    print(doc.id, " : ", doc.to_dict())
```

Kao što se iz primera može primetiti postoji razlika u odnosu na SQL naredbe. Da bi smo izvršili bilo koji upit u Cloud Firestore bazi podataka neophodno je prvo referencirati se na kolekciju dokumenata koji želimo da pretražujemo. Zatim, kada smo referencirali željenu kolekciju u where funkciji definišemo upit i to

where('ime_polja','operacija/operator_uporedjivanja','vrednost_koju trazimo'). Firestore podržava sledeće operatore poredjenja:

1. < operator manje
2. <= operator manje ili jednako
3. == operator jednakosti
4. > operator veće
5. >= operator veće ili jednako
6. != operator nije jednako
7. array-contains – da li array polje sadrži element
8. array-contains-any – da li niz polje sadrži element s tom razlikom što se ponasa kao operator OR i ukoliko navedenu vrednost prilikom pretrage definišemo sa 3 elementa niza, vratiće dokumente ako se barem jedno polje od zadatog niza nalazi u dokumentu
9. in
10. not-in

Firestore poseduje ograničenja vezana za neke od gore nabrojanih operatora, za operator != ograničenja su da :

1. za zadati upit sa operatorom != se podudaraju dokumenti u kojima postoji dato polje (u slučaju da ne postoji polje neće uzeti u obzir ostala dokumenta),
2. kombinacija not-in i != operatora u složenom upitu nije dozvoljena i
3. kao poslednje ograničenje je to da u složenom upitu, operatori (<,<=,>,>=) i operatori !=,not-in moraju se filtrirati u istom polju.

Pored ovih ograničenja postoje i ograničenja za operatore in, not-in, array-contains-any i to su sledeća:

1. sva tri operatora podržavaju do 10 uporednih vrednosti (na primer moguće je uporediti vrednost polja dokumenta koji je sačuvan kao niz sa 10 elemenata zadatog niza),
2. možemo koristiti najviše jednu klauzulu koja sadrži array-contains operator,
3. nije moguće porediti array-contains i array-contains-any,
4. od ove tri klauzule moguće je koristiti najviše jednu ovu klauzulu po upitu i nije dozvoljeno kombinovanje ova tri parametara u istom upitu,
5. ne možemo operator not-in i operator != kombinovati u jednom upitu

6. kao poslednje ograničenje jeste to da operator jednakosti == i operator *in* ne možemo kombinovati u istom upitu.

Firestore takođe korisnicima nudi mogućnost ograničenja prilikom pretraživanja kao i sortiranja prilikom izvršavanja upita, kao što kod SQL baza posedujemo TOP, LIMIT, FETCH FIRST ili ROWNUM i ORDER BY klauzule tako i kod Cloud Firestore baze podataka možemo definisati koliko rezultata želimo da prikažemo i u kom redosledu. To je moguće postići koristeći funkcije orderBy() i limit() funkcijama. Primer je prikazan u nastavku:

```
doc_ref = db.collection('userData')
query = doc_ref.order_by('name',direction=firestore.Query.DESCENDING).limit(1)
for doc in query.get():
    print(doc.id," : ",doc.to_dict())
```

Ovim primerom vratićemo 1 dokument koji će biti sortiran po imenu i to u opadajućem redosledu. Vrlo je bitno da Firestore uređuje podatke leksikografski.

Kod Firestore baze podataka nije dozvoljeno pretraživanje baze po jednom polju a zatim urediti podatke na osnovu drugog polja, primer je dat u nastavku:

```
doc_ref = db.collection('userData')
query = doc_ref.where('lista_drustvenih_mreza.twitter','==','marko').order_by('name')

for doc in query.get():
    print(doc.id," : ",doc.to_dict())
```

Kao rezultat ovakvog upita javice sledeću grešku kao na slici 10:



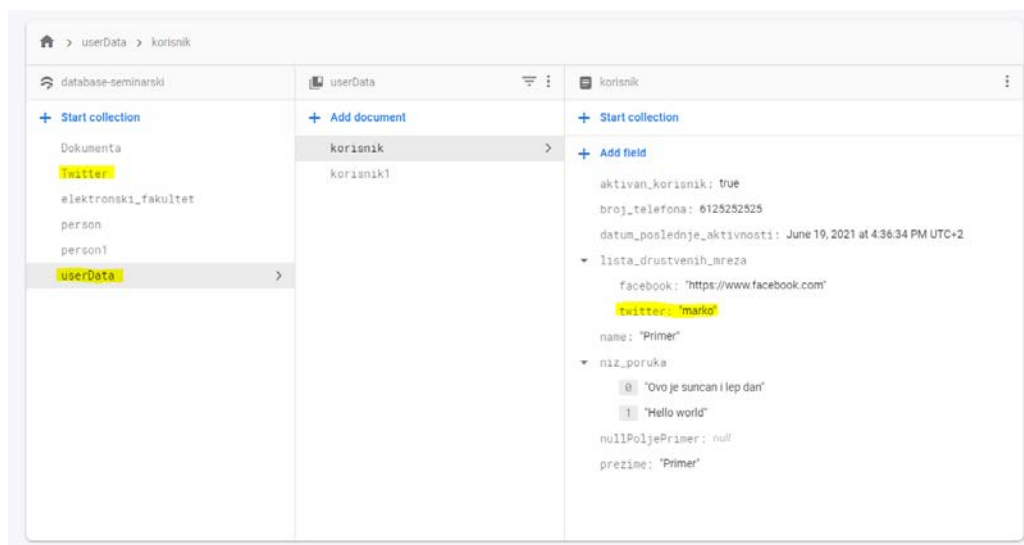
```
google.api_core.exceptions.FailedPrecondition: 400 The query requires an index. You can create it here: https://console.firebase.google.com/v1/r/project/database-seminarski/firestore
```

Slika 10: Rezultat izvršenja upita

Ovakav upit „nije dozvoljen“ kod Firestore baze podataka, jer svako polje u dokumentu predstavlja indeks, zato što kao što smo i na početku poglavlja govorili upiti kod Firestore baze podataka su ekspresni i brzi, samim tim svako polje u dokumentu predstavlja indeks a indeksi služe za brzu pretragu baze podataka, ovim dolazimo do konflikta u bazi. Pretražujemo po indeksu *lista_drustvenih_mreza.twitter* i *name* indeksi nad istim dokumentom i dolazimo do problema. Cloud Firestore svojim korisnicima kada prepozna ovakav tip upita kao grešku prikazaće poruku kao na slici 10. Korisnik da bi izvršio ovakav upit potrebno je da kreira kompozitni ključ sa ovim poljima zatim je moguće izvršiti upit u bazu. Pored te informacije

Google Cloud Firestore baza podataka korisniku nudi link putem koga može kreirati tip indeksa i time omogućiti pretraživanje na osnovu korisnički definisanih kolona. Klikom na dugme otvara se stranica gde je potrebno samo pretisnuti dugme ok i Firestore će kreirati indeks za korisnika i samim tim upit je moguće izvršiti. Detaljnije o indeksima i indeksnim strukturama govorićemo u sledećem poglavlju. Da korisnici na bi dolazili do konflikta polja u where i order_by klauzuli moraju da budu ista.

Kod SQL baza podataka mogli smo izvršiti naredbu preseka dve tabele, JOIN, kod Firestore baze podataka to nije moguće. Kao primer uzećemo kolekciju *userData* koja pamti korisnike i pamti listu društvenih mreža, i koristićemo kolekciju *twitter* koja sadrži informacije o korisnicima (Slika 11). Ukoliko zelimo da vidimo informacije o *Twitter* korisniku, specificiranom u upitu, mi možemo to izvršiti na sledeći način:



Slika 11: Kolekcije dokumenata nad kojim želimo da izvršimo JOIN SQL naredbu

```
doc_ref = db.collection('userData')
doc_ref_twitter = db.collection('Twitter')

query= doc_ref.where('lista_drustvenih_mreza.twitter','==','marko')
for doc in query.stream():
    pom = doc.to_dict()
    query =
doc_ref_twitter.where('username','==',pom.get('lista_drustvenih_mreza').get('twitter'))
    for rez in query.stream():
        print(rez.id," : data: ",rez.to_dict())
```

Kao rezultat ovakvog upita biće prikazan korisnik Twitter društvene mreže koji ima korisničko ime marko. Rezultat upita je dat na slici 12.

```
C:\Users\marko\AppData\Local\Programs\Python\python39\python.exe "F:/MASTER_STUDIJE/Sistemi za upravljanje bazama podataka/Seminarski3/primer2.py"
user : data: {'username': 'marko', 'name': 'Marko', 'tweets': {'date': '13.05.2021', 'tweet': 'Tekst @marko'}}

Process finished with exit code 0
```

Slika 12: Rezultat izvršenja upita

Cloud Firestore baza podataka takođe poseduje već ugrađenu funkciju kojom se može inkrementirati vrednost number polja u dokumentu, to se postiže pozivom funkcije `firestore.Increment(inkrementalni_segment)`.

3.4.3. Ažuriranje dokumenata kod Google Cloud Firestore baze podataka

Da bismo ažurirali dokument u Cloud Firestore bazi podataka, pre svega neophodno je kreirati referencu na željeni dokument koji želimo da ažuriramo i zatim u metodi update navedemo polja koja želimo da ažuriramo. Primer je dat u nastavku:

```
doc = db.collection('userData').document('korisnik1')
doc.update({'aktivan_korisnik':False})

print(doc.get().to_dict())
```

Pored ovakvog tipa ažuriranja, Firestore nudi korisnicima ugrađene funkcije za ažuriranje nizove podataka pozivima funkcija `ArrayUnion()` ili `ArrayRemove()`. Primer je dat u nastavku:

```
doc = db.collection('userData').document('korisnik1')
doc.update({'niz_poruka':firestore.ArrayUnion(['Proba1'])})
doc.update({'niz_poruka':firestore.ArrayRemove(['Ovo je suncan i lep dan'])})
print(doc.get().to_dict())
```

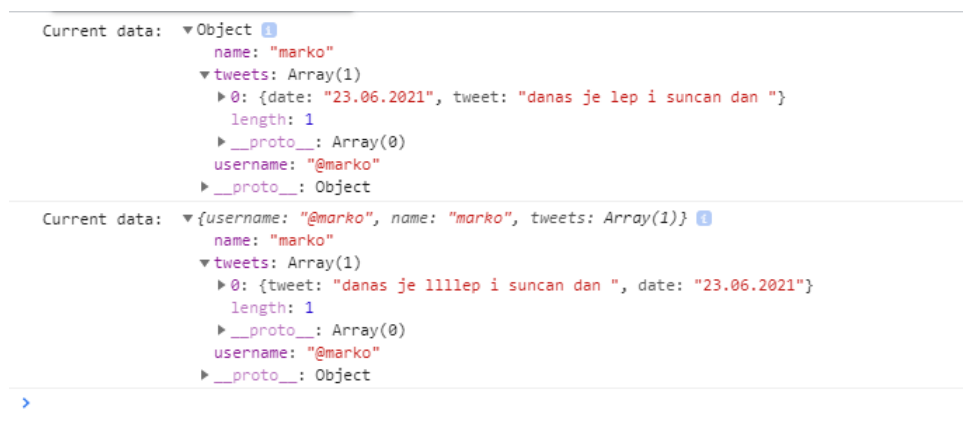
Metoda `ArrayUnion` koristi se kada dokument poseduje niz kao strukturu podataka i izvršenjem ove naredbe u zadati niz dodaje se jos jedan element u našem slučaju `Proba1`. Metoda `ArrayRemove` se koristi prilikom brisanja elementa niza u zadatom polju.

3.4.4. Osluškivanje promene na dokumentima kod Google Cloud Firestore baze podataka

Pored osnovnih operacija koje možemo izvršiti nad dokumentima u bazi podataka, postoji i opcija osluškivanja promena nad dokumentima u bazi podataka. Da bi smo osluškivali promene nad samom dokumentu potrebno je u web aplikaciji dozvoliti, uključiti, osluškivač. To se postiže izvršenjem sledeće komande

```
db.collection("Twitter").doc("marko")
  .onSnapshot((doc) => {
    console.log("Current data: ", doc.data());
  });
```

Nakon svake izmene u konzoli će se prikazati trenutno stanje podataka nakon izmene, primer je prikazan na slici 13.



Slika 13: Prikaz konzole nakon podešavanja osluškivača na dokument čiji je id marko

Pored osluškivanja promene samo jednog dokumenta u kolekciji, moguće je kreirati osluškivače na više dokumenata u okviru kolekcije. To se postiže izvršenjem sledeće komande

```
db.collection("Twitter").where("status", "==", "aktivan")
  .onSnapshot((querySnapshot) => {
    var tweets = [];
    querySnapshot.forEach((doc) => {
      tweets.push(doc.data().tweets);
    });
    console.log("Current tweets : ", tweets.join(", "));
  });
```

Ovom komandom iz naše kolekcije Twitter za svakog aktivnog korisnika pratićemo informacije o postovima unutar naše društvene mreže twitter.

3.5. Indeksi kod Google Cloud Firestore baze podataka

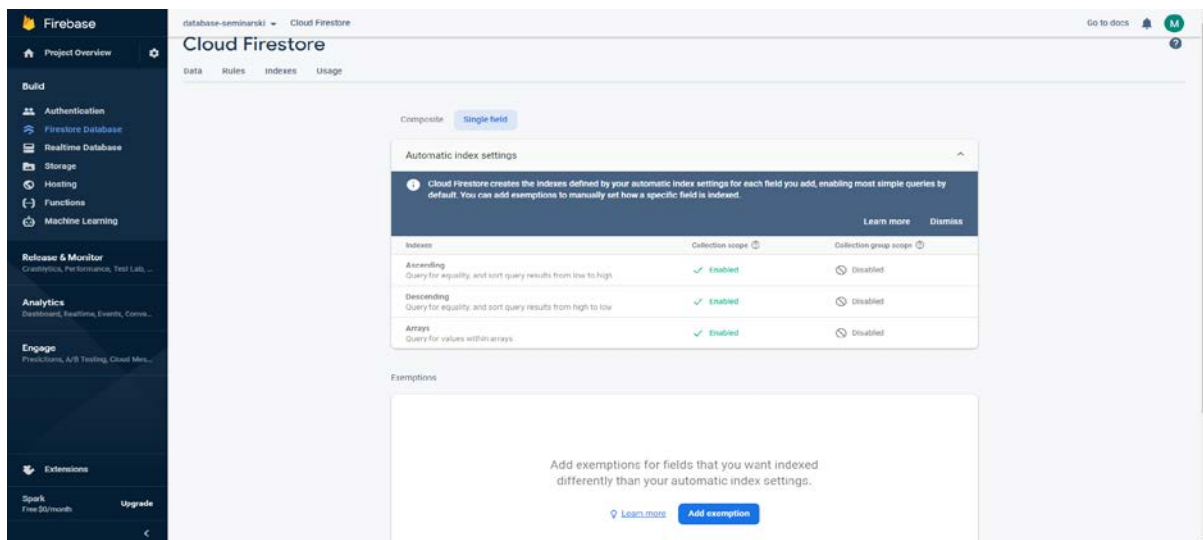
Kada smo govorili o indeksima kod relacionih baza podataka rekli smo da indeksi obezbeđuju brzu pretragu podataka, to im je glavna karakteristika i namena korišćenja. Kod Oracle baze podataka postoje dva tipa indeksa B-tree i Bitmap indeksi. Primer kreiranja indeksa kod Oracle baze podataka je sledeći:

```
CREATE INDEX "marko".PRODUCT_ID ON "marko".PRODUCT (ID ASC);
```

Google Cloud Firestore baza podataka kao što smo i već napomenuli garantuje visoke performanse upita koristeći indekse za sve upit. Prilikom kreiranja novog dokumenta u kolekciji automatski se kreira indeks za svako polje u tom dokumentu, a ako se u dokumentu nalazi i neki ugnježdjeni objekat takođe sva njegova polja se indeksiraju i tako do dubine 20 nakon čega prestaje sa indeksiranjem polja. Naravno, u zavisnosti od potrebe aplikacije mogu se kreirati i dodatni indeksi.

Kod Google Cloud Firestore baze podataka koriste se dva tipa indeksa: kompozitni indeksi i indeksi sa jednim poljem (Single-field indexes). Pored toga što se razlikuju u broju polja, indeksi jednog polja i kompozitni indeksi se razlikuju i po načinu upravljanja.

Single-field indeksi koji se kreiraju za svako polje u okviru dokumenta. Svaka vrsta kod single-field indeksa sadrži vrednost određenog polja dokumenta i lokaciju tog dokumenta u bazi. Sve vrste su sortirane. Firestore baza koristi ove indekse za obične upite čitanja dokumenata. Iako se ovi indeksi automatski kreiraju moguće je podesiti ovo kreiranje kao i dodati neke izuzetke. Podrazumevana podešavanja su da se za svako polje koje nije niz ili objekat, kreiraju dva single-field indeksa na nivou kolekcije, jedan u opadajućem redosledu, a drugi u rastućem, za svako polje ugnježđenog objekta koji nije niz ili objekat kreiraju se dva single-field indeksa na nivou kolekcije, jedan u opadajućem a drugi u rastućem i kao treće podrazumevano podešavanje, za svako polje koje je niz u dokumentu, kreira se jedan array-contains indeks na nivou kolekcije.



Slika 14: Podrazumevana podešavanja indeksa kod Google Cloud Firestore baze podataka

Na slici 14 su prikazana podrazumevana podešavanja za automatso kreiranje sigle-field indeksa.

Pored automatskog kreiranja single-field indeksa moguće je napraviti izuzetke za indeksiranje nekih polja i tako u potpunosti izbaciti indeksiranje tih polja ili samo podesiti da se kreira samo jedan umesto dva single-field indeksa. Ukoliko se napravi izuzetak za polje koje ustvari objekat, onda se taj izuzetak odnosi na sva njegova polja. Takođe moguće je napraviti izuzetak i za polja ugnježđenog objekta.

3.5.1. Kompozitni indeksi

Kada smo govorili o indeksima rekli smo da Google Cloud Firestore baza podataka automatski kreira indeks za svako polje i prilikom izvršavanje upita koji se odnosi na više polja dešava se greška. U poglavlju 3.4.2 rekli smo da se korisniku kao obaveštenje o tome dostavlja informacija o grešci koja ima sledeći tekst (400 The query requires an index <link_do_vase_baze>). Ovim korisniku se ostavlja mogućnost kreiranja kompozitnog indeksa za potrebe svoje aplikacije. Cloud Firestore sam kreira single-field indekse, razlog zbog čega ne kreira kompozitnih indeksa je ukoliko imamo dokument sa 20 polja, obično da bi se kreirali i single-field indeksi i kompozitni potrebno je $1.048576e+26$ dokumenata koje pamte indeksirane vrednosti, što je prilično mnogo, dok za dokument sa 6 polja broj indeksa potrebnih da bi pokrili ceo upit je 46656. Deluje da nije potrebno puno prostora za skladištenje ovolikog broja dokumenata o indeksima ali u stvari jeste jer postoji u datom trenutku milion korisnika koji koristi bazu podataka pa je potrebno održavati svih 46656 indeks tabela za svakog korisnika, sto je opterećenje za sistem. Zbog toga Firestore dozvolja korisniku da kreira kompozitne indekse za podatke koje najčešće pretražuje.

Kompozitni indeksi za razliku od single-field indeksa, mogu da sadrže više polja umesto jednog. Firestore baza koristi kompozitne indekse kako bi podržala upite koji nisu podržani od strane single-field indeksa. Kompozitni indeksi se ne kreiraju automatski od strane Firestore baze iz razloga što postoji veliki broj kombinacija koje zavise od boja polja koje

ima dokument. Jedan način je da mi ručno kreiramo indekse preko Firebase konzole i dela koji se odnosi na Firestore bazu (slika 15). A drugi način koji se i najviše preporučuje, jeste način koji smo opisali u uvodu, jednostavnim klikom na link koji je prikazan u konzoli prilikom izvršenja upita i korisniku će se prikazati prozor kao na slici 16 koji nam omogućava da automatski kreiramo potreban kompozitni indeks koji je Firestore već podesio na osnovu našeg upita. Sve kompozitne indekse moguće je videti u Firestore delu „Indexes“ u okviru Firebase konzole.

Create a composite index

Cloud Firestore uses composite indexes for compound queries not already supported by single field indexes (ex: combining equality and range operators).

★ **Recommended**
Instead of defining a composite index manually, run your query in your app code to get a link for generating the required index. [Learn more](#)

Collection ID
userData

Fields to index
At least two fields required*

1	prezime	Ascending
2	datum_poslednje_aktivnosti	Descending

[Add field](#)

[Cancel](#) [Create index](#)

Creation time depends on the amount of data being indexed

Slika 15: Primer ručnog kreiranja kompozitnog indeksa kod Firestore baze podataka

Create a composite index

Cloud Firestore uses composite indexes for compound queries not already supported by single field indexes (ex: combining equality and range operators).

Index	Collection(s)	Query scope	Fields
	userData	Collection	lista_drustvenih_mreza.twitter Ascending name Ascending

[Cancel](#) [Create index](#)

Creation time depends on the amount of data being indexed

Slika 16: Primer automatskog kreiranja kompozitnog indeksa na osnovu procene Firestore baze podataka

3.5.2. Veličina indeksa i limiti

Kao i kod svih baza podataka i kod Firestore baze indeksi zauzimaju neki prostor. Koliko će indeks zauzimati prostor zavisi od nekoliko faktora.

Kod single-field indeksa koji ima scope na nivou jedne kolekcije u veličinu indeksa ulaze: veličina imena indeksiranog dokumenta koji se posebno računa na osnovu imena svake kolekcije i dokumenta koje se nalaze u imenu, veličina imena roditelja indeksiranog dokumenta pa njegovog roditelja koji se isto posebno računa kao što je prethodno navedeno za ime indeksiranog dokumenta, veličina stringa za naziv polja i plus 1 bajt, veličina indeksirane vrednosti polja i plus dodatnih 32 bajta. Tako da na primeru dokumenta koji se nalazi u podkolekciji i ima ime `userD/marko/tasks/task_id` za polje `"završen": false`, index zauzimaće 109 bajta, od kojih se 44 bajta odnose na ime indeksiranog dokumenta, 27 bajta na ime roditelja roditelja, 4 plus 1 bajt za ime polja, 1 bajt za bool vrednost i 32 dodatna bajta. A za single-field indekse koji imaju scope na nivou grupe kolekcija računica je slična kao i u prethodnom slučaju, samo što se ne računa veličina imena roditelja roditelja indeksiranog dokumenta i umesto 32 dodatna bajta dodaje se 48 bajta. Tako da za isti primer indeks na nivou grupe kolekcija bi zauzimao 98 bajtova, od čega su 44 bajta na ime, 5 na ime polja, 1 na vrednost polja i 48 dodatnih bajtova.

Za kompozitne indekse dodaćemo na primer još jedno polje `"prioritet": 1`. Kod kompozitnih indeksa koji ima scope na nivou jedne kolekcije veličina indeksa se računa na sličan način kao i kod single-field indeksa, samo što se ne računa vrednost imena polja, ali se dodatno sabiraju vrednosti svih indeksiranih polja. Tako da bi kompozitni indeks na nivou jedne kolekcije u našem primeru zauzimao 112 bajta, od kojih je 44 bajta na ime indeksiranog dokumenta, 27 bajta na ime roditelja roditelja, 1 bajt za bool vrednost polja `„završen“`, 8 bajta za integer vrednost polja `„prioritet“` i 32 dodatna bajta. A kod kompozitnih indeksa koji ima scope na nivou grupe kolekcija veličina se razlikuje od prethodnog scopa samo što se u veličinu ne uračunava veličina imena roditelja roditelja indeksiranog dokumenta. Tako da bi u našem primeru kompozitni indeks na nivou grupe kolekcija zauzimao 85 bajta, od kojih je 44 bajta za ime, 1 bajt za bool vrednost polja `„završen“`, 8 bajtova za integer vrednost polja `„prioritet“` i 32 dodatna bajta.

Firestore baza ipak ima neke limite što se tiče indeksa. Maksimalan broj kompozitnih indeksa je 200, maksimalan broj izuzetaka za single-field indekse je 200, maksimalan broj indeksiranih polja za svaki dokument je 40,000, maksimalna veličina svakog indeksa je 7,5 KiB, maksimalna vrednost svih indeksa po dokumentu je 8 MiB i maksimum veličina vrednosti indeksiranog polja je 1500 bajta sve preko 1500 bajta se deli na delove.

Neke najbolje prakse koje se tiču indeksa jeste da se na primer za polja koja sadrže velike stringove napravi izuzetak i ne indeksira to polje ili ako dokument sadrži veće nizove ili objekte broj verovatno će broj indeksa da pređe dozvoljen limit, tako da je bolje da se i za njih napravi izuzetak i da se ne indeksiraju ta polja.

3.6. Offline režim rada Cloud Firestore baze podataka

Cloud Firestore baza podataka omogućava korisnicima da čitaju, upisuju i menjaju podatke iako njihov uređaj nije na mreži. Ovo se postiže time što Firestore biblioteke keširaju podatke koje aplikacija koristi iz Firestore baze, tako da ih aplikacija može koristiti i kada je uređaj offline. Kada se uređaj ponovo poveže na mrežu Firestore baza sinhronizuje sve lokalne promene koje su se desile.

Offline režim rada je trenutno nedostupan za programsko okruženje python, omogućen je samo za mobilne i web aplikacije pa tako u daljem izlaganju baziraćemo se na konfiguraciji i objašnjenju Offline režima rada na web aplikacijama. Inicijalno, za mobilne aplikacije Offline režim rada aplikacije je automatski uključena opcija, dok kod web aplikacija potrebno je izvršiti inicijalizaciju. Razlog zbog čega nije dostupan Offline režim rada za ostale aplikacije je taj zbog toga što postoji različiti nivoi „offline-a“. Pod offline režimom rada aplikacije se može smatrati aplikacija koja je zbog tehničkih stvari trenutno nema pristup internetu, ili pak internet je suviše slab da bi izvršio upit nad bazom, ili pak korisnik je podesio sistem na airplane režim rada. Zbog tih problema jedino mobilne aplikacije imaju inicijalno postavljen offline režim rada baze podataka.

Kod web aplikacija da bi koristili Offline režim rada neophodno je inicijalizovati *enablePersistence* funkcije. Firestore biblioteke automatski obezbeđuju online i offline pristup podacima kao i sinhronizaciju podataka kada korisnik ponovo bude online.

U ovom režimu rada, Firestore biblioteke kesiraju svaki dokument koji se pročita iz baze kako bi se kasnije mogli koristiti. Veličina keša je postavljena na 40 MB i periodično se brišu stari i ne korišćeni dokumenti. Moguće je postaviti proizvoljnu veličinu keša, kao i da se isključi funkcija periodičnog čišćenja starih dokumenata. Ukoliko ne postoje kesirani podaci korisnik će uvek dobiti prazan niz kada pokuša da učita podatke dok ukoliko pokuša da ucita dokument dobiće informacije o grešci.

Na sledećem primeru prikazano je kako se postavlja proizvoljna velicina kesa kao i poziv *enablePersistence* kojom se inicijalizuje offline režim rada baze podataka

```
firebase.firestore().settings({  
    cacheSizeBytes: firebase.firestore.CACHE_SIZE_UNLIMITED  
});  
  
firebase.firestore().enablePersistence();
```

Pošto Cloud firestore baza podataka koristi realtime listeners kako bi obezbedio realtime funkcionalnost, u slučaju offline režima ovi listeneri dobijace promene (snapshot) koje su se desile nad keširanim podacima. Moguće je proveriti da li su promene pristigle od keširanih podataka ili iz baze koristeći *fromCache* property *SnapshotMetadata* pristiglog *snapshot-a*.

Ako je property `fromCache` postavljen na `true` onda je reč o keširanim podacima, a ako je `false` onda su to poslednje promene koje su pristigle iz baze. Pošto se ni jedan event ne emituje kada se promene keširani podaci potrebno je prilikom `subscribe`-ovanja na event postaviti i property `includeMetadataChanges` na `true`, kako bi obezbedili da se eventi emituju i kada se promene keširani podaci.

Sledeći primer ilustruje `subscribe` funkcije na event-e kada dođe do promene dokumenta kolekcije `userData` čiji je property `aktivan_korisnik` jednak `true`, ovim postizemo da vidimo koliko je korisnika promenilo svoj status u neaktivan i da li se radi o keširanim podacima ili o podacima sa servera.

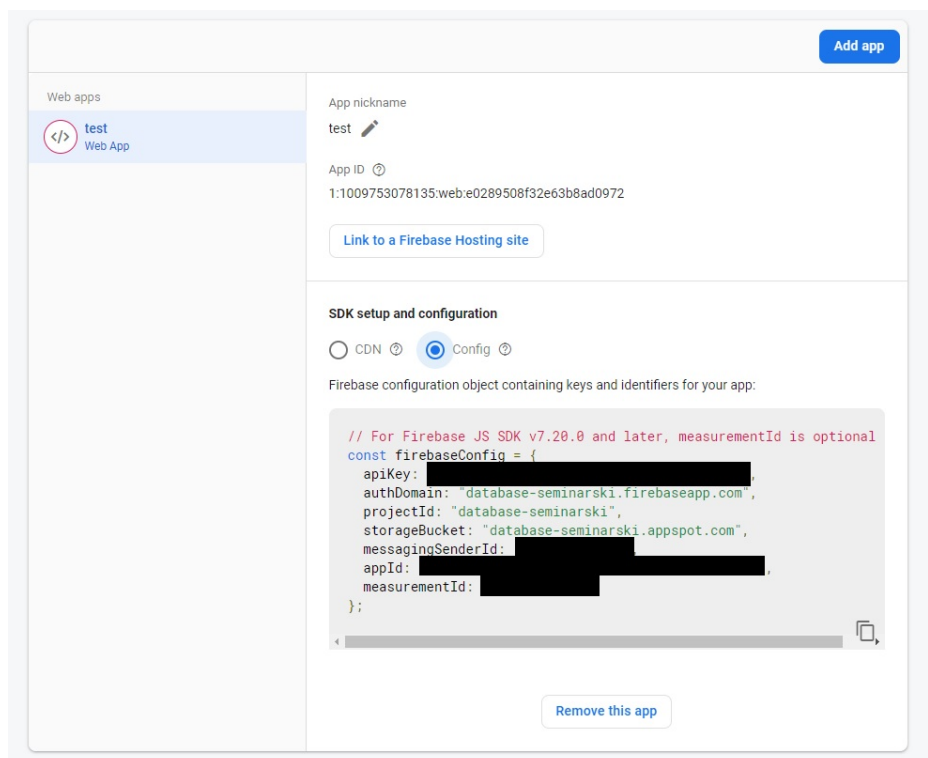
```
db.collection("userData").where("aktivan_korisnik", "==", true)
.onSnapshot({ includeMetadataChanges: true }, (snapshot) => {
  snapshot.docChanges().forEach((change) => {
    if (change.type === "added") {
      console.log("New aktivan_korisnik: ", change.doc.data());
    }

    var source = snapshot.metadata.fromCache ? "local cache" : "server";
    console.log("Data came from " + source);
  });
});
```

3.6.1. Praktični primer offline režima rada Cloud Firestore Baze podataka

U prethodnom poglavlju objasnili smo offline režim rada Cloud Firestore baze podataka, sada ćemo na praktičnom primeru pokazati kako zapravo podesiti offline režim rada korektno.

Da bismo pravilno definisali offline režim rada, prvo je neophodno prijaviti se na samu bazu podataka, odnosno projekat i klikomna dugme settings u general sekciji dodati aplikaciju i zatim u sekciji SDK setup and configuration izabrati Config konfiguraciju i kopirati je u projekat. Izgled konfiguracionog fajla dat je na slici 17.



Slika 17: Izgled jednog konfiguracionog fajla za aplikaciju test

Pored ove konfiguracije potrebno je takođe u aplikaciji dodati deo koji se odnosi na samu konfiguraciju firebase baze podataka, to se postiže kopiranjem sledeće linije koda.

```
<script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-app.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-auth.js"></script>
<script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-firestore.js"></script>
```

Zatim je potrebno inicijalizovati aplikaciju, i to radimo tako što ćemo, pre svega, iskopirati konfiguraciju našeg projekta pa zatim izvršiti sledeću liniju koda.

```
firebase.initializeApp(firebaseConfig);
```

Ukoliko je neuspela konfiguracija, korisniku će se pokazati greška.

Nakon uspešne konfiguracije naše aplikacije neophodno je inicijalizovati offline režim rada aplikacije, to se postiže korišćenjem funkcije `enablePersistence`, primer inicijalizacije offline rešima rada je sledeći:

```
firebase.firestore().enablePersistence().then(function(){
    alert("Uspeli smo");
})
.catch(function(err){
    if(err.code=='unimplemented'){
        console.log(err.code);
    }
    if(err.code == 'failed-precondition'){
        console.log(err.code);
    }
});
```

Nakon uspešnog inicijalizovanog možemo definisati veličinu keš memorije, to se postiže sledećom komandom:

```
firebase.firestore().settings({
    cacheSizeBytes: firebase.firestore.CACHE_SIZE_UNLIMITED
});
```

Nakon podešavanja veličine keša, potrebno je inicijalizovati samo okruženje. To se postiže izvršenjem sledeće komande:

```
var db = firebase.firestore();
```

Nakon svih potrebnih podešavanja i svih potrebnih parametara možemo koristiti offline režim rada. Da bi smo demonstrirali primer kreiraćemo offline režima rada kreirali smo jednostvanu formu koja izgleda kao na slici 18. Ova forma pamti informacije o korisnicima *twitter* društvene mreže, simulacija *twitter* društvene mreže. Kao podatke koje ćemo pamtit i to su ime korisnika, username korisnika mreže kao i tweet odnosno post koji korisnik hoće da postavi na društvenu mrežu. Za demonstraciju ovog primera neophodno je pre svega da imamo kolekciju koja će pamtit i informacije o korisničkim aktivnostima.

Uneti username korisnika

Uneti ime korisnika

Twittet

Save

Slika 18. Primer jednostavne forme

U prethodnim poglavljima govorili smo o kolekciji *Twitter* koja pamti informacije o aktivnostima korisnika na „društvenoj mreži“. Klikom na dugme Save inicira se upis/ažuriranje samog dokumenta.

Pre nego što demonstriramo offline režim rada potrebno je videti samu definiciju dugmete *save*. Definicija samog dugmeta *save* data je u nastavku.


```

saveButton.addEventListener("click",function(){
  db.collection("Twitter").doc(userNameInput.value).get().then((doc)=>{
    if(doc.exists){
      db.collection("Twitter").doc(userNameInput.value).update({
        "tweets":firebase.firestore.FieldValue.arrayUnion({
          "date": new Date().toString(),
          "tweet":tweet.value
        })
      })
    }
    .then()==>{
      console.log("Document successfully updated!");
    });
    var source = doc.metadata.fromCache ? "local cache" : "server";
    console.log("Data came from " + source);
  } else {
    db.collection("Twitter").doc(userNameInput.value).set({
      "name":nameInput.value,
      "tweets":[{
        "date":new Date().toString(),
        "tweet":tweet.value
      }],
      "username":"@"+userNameInput.value
    }).then()==>{
      console.log("Document successfully created");
    });
    var source = doc.metadata.fromCache ? "local cache" : "server";
    console.log("Data write on " + source);
  }
}).catch((error)==>{
  console.log("Error ",error);
});
});

```

Nakon klika na dugme save inicira se preuzimanje dokumenata iz baze podataka, ukoliko postoji dokument sa zadatim id-ijem (u ovom primeru kao id dokumenta koristi se parametar username koji predstavlja korisničko ime na našoj društvenoj mreži) izvršiće se update naredba i time ćemo zapamtiti vrednost posta odnosno Tweet polje u samoj bazi. Ukoliko ne postoji dokument sa zadatim id-ijem kreiraće se nov dokument sa svim unetim podacima. Prilikom kreiranja i ažuriranja samih podataka u bazi postoji opcija metadata kojom se ispituje da li je dokument upisan u lokalni keš ili direktno na sam server na kom se pamti baza podataka.

Nakon unosa svih vrednosti u formi, prikazanoj na slici 19, korisniku će se u konzoli prikazati rezultat kao na slici 20.

Data write on server	test1.js:58
Document successfully created	test1.js:55
>	

Slika 19. Rezultat upisa u bazu podataka

Ovakva vrednost je i očekivana, podaci su upisani u dokument na serveru. Da bismo videli moć offline režima rada neophodno je isključiti internet. Nakon što prešli u offline režim rada sada ćemo probati da promenimo vrednost polja tweet i sada će rezultat u konzoli biti kao na slici 20.

Data came from local cache	test1.js:45
▶ GET https://www.google.com/images/clear dot.gif?zx=ytlcvka7wmfw net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=ytlcvka7wmfw:1	
▶ [2021-06-25T19:19:05.618Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=7os0rxytq1j&t=1 net::ERR_NAME_NOT_RESOLVED	
▶ GET https://www.google.com/images/clear dot.gif?zx=rq3m2q9pmk88 net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=rq3m2q9pmk88:1	
▶ [2021-06-25T19:19:05.628Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=vobtmdei3i8k&t=1 net::ERR_NAME_NOT_RESOLVED	
▶ GET https://www.google.com/images/clear dot.gif?zx=6n7lm2fv5wal net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=6n7lm2fv5wal:1	
▶ [2021-06-25T19:19:06.603Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=ol6b0ucgsnep&t=1 net::ERR_NAME_NOT_RESOLVED	
▶ GET https://www.google.com/images/clear dot.gif?zx=hsapz12wufli net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=hsapz12wufli:1	
▶ [2021-06-25T19:19:07.859Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=dr2fe83hluyj&t=1 net::ERR_NAME_NOT_RESOLVED	
▶ GET https://www.google.com/images/clear dot.gif?zx=e0htq414ilf net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=e0htq414ilf:1	
▶ [2021-06-25T19:19:11.067Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=5jt27twlvswx&t=1 net::ERR_NAME_NOT_RESOLVED	

Slika 20. Rezultat izvršenja operacije prilikom prelaska na offline režim rada

Pošto eksplicitno nismo naglasili kada prelazimo u offline režim javljaće se ovakve greške. Razlog je taj zato što naša aplikacija pokušava da upise na server informacije o dokumentu svakog puta prilikom klika na aplikaciju. Takođe u ovom primeru se ogleda nivo offline-a. Kod web aplikacija mogu biti različiti nivoi offline režima rada, kao što smo i govorili, pa zbog toga naša aplikacija će na svakih 1 minut pokušati da upiše vrednost na sam server. Ukoliko pak postanemo online, ukoliko se konektujemo na internet, situacija u konzoli će biti kao na slici 21.

▶ [2021-06-25T19:22:00.769Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=vlxtoa4rljpr&t=1 net::ERR_NAME_NOT_RESOLVED	
▶ GET https://www.google.com/images/clear dot.gif?zx=mbklok26t6q4 net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=mbklok26t6q4:1	
▶ [2021-06-25T19:22:40.235Z] @firebase/firestore: Firestore (8.6.8): Connection WebChannel transport errored: logger.ts:115	
▶ Ir {type: "c", target: Er, g: Er, defaultPrevented: false, status: 1}	
▶ POST https://firestore.googleapis.com/google.firestore.v1.Firestore/Write/channel_1%3A1009753078135%3Aweb%3Ae028950...%0D% xhrio.js:671	
0A&zx=57oownkuhux2&t=1 net::ERR_NAME_NOT_RESOLVED	
▶ GET https://www.google.com/images/clear dot.gif?zx=nia55fpjgest net::ERR_NAME_NOT_RESOLVED www.google.com/image...f?zx=nia55fpjgest:1	
Document successfully updated!	test1.js:42
>	

Slika 21: Rezultat izvršenja operacija prilikom konekcije na online režim rada

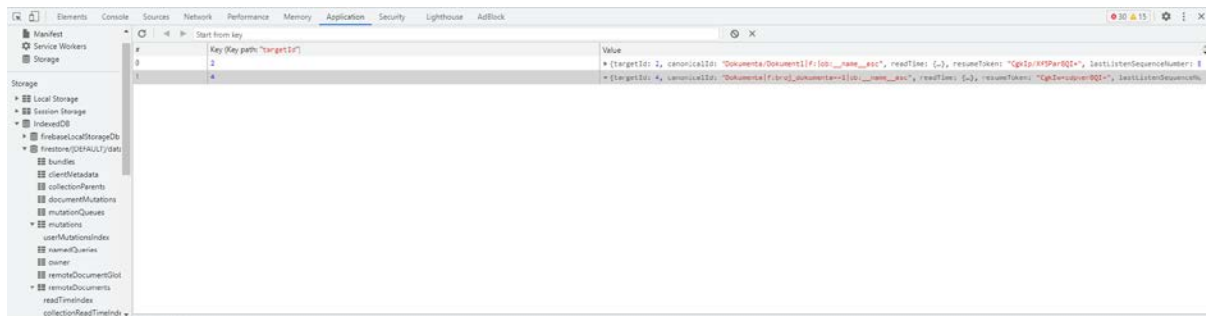
Kada se uspostavi veza između naše aplikacije i servera na kom je baza podataka smeštena, tada će se automatski izvršiti ažuriranje/upis u samu bazu podataka. Ukoliko je postojalo više

upisa, redosled u kome će se upisati podaci u bazu podataka je u rastućem redosledu, odnosno prvo će se upisati prvo kreirani dokument pa zatim drugi i tako redom.

Da bi smo izbegli greške u konzoli, stalan upit da se ažuriraju podaci na samom serveru, kod Google Cloud Firestore baze podataka moguće je uključiti i isključiti offline režim rada. To se postiže korišćenjem funkcije `firebase.firestore().disableNetwork()` koja automatski prelazi na offline režim rada. Pravilnije je omogućiti ovakav offline režim rada jer time se automatski podrazumeva firestore-ov offline režim rada, i samim tim neće biti ovoliko grešaka u konzoli korisnika.

Problem sa offline režimom rada je taj što ukoliko postoje dva korisnika koja koriste istu bazu podataka, korisnik A i korisnik B. Ukoliko korisnik A izvrši izmenu u dokumentu prilikom offline režima rada i u koliko korisnik B koji je online korisnik izvrši upis u dokument doći će do preklapanja. U bazi podataka prvo će se upisati dokument online korisnika pa zatim će se prepisati rezultat operacije offline korisnika.

Da bismo videli keširane podatke, potrebno je u konzoli u sekciji Application otvoriti u Storage kartici otvoriti IndexedDB, izgled je prikazan kao na slici 22.

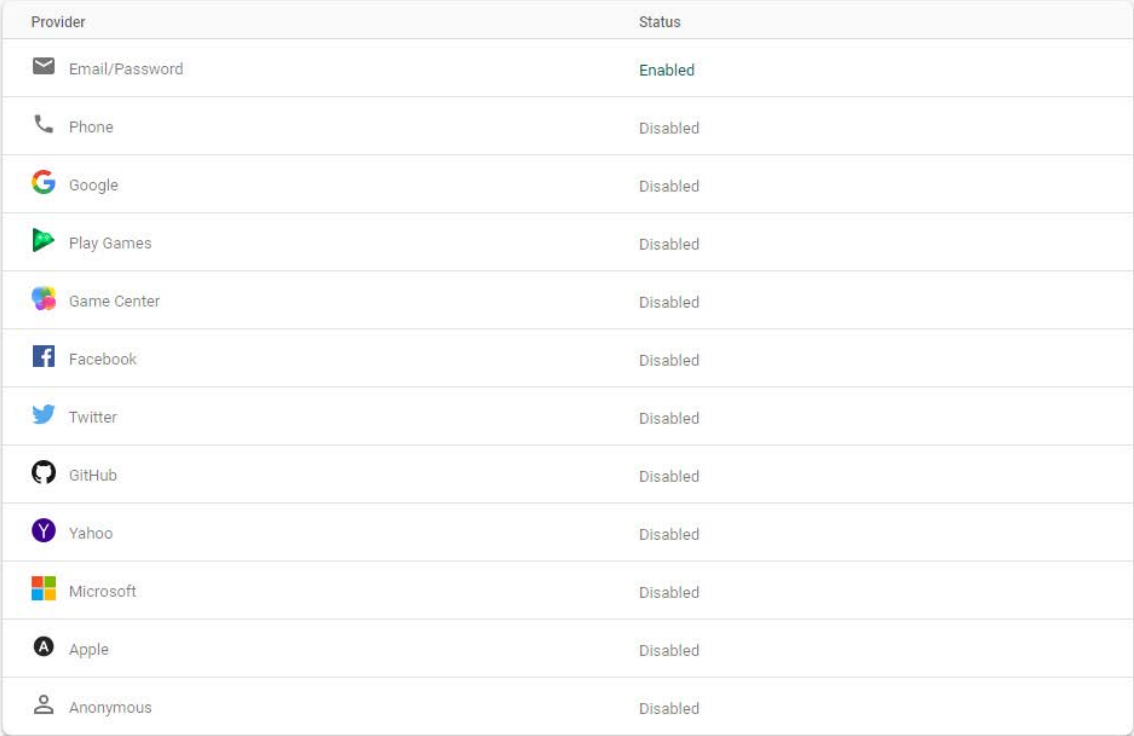


Slika 22: Prikaz keširanih informacije u konzoli

3.7. Kreiranje i autentifikacija korisnika na Google Cloud Firestore bazu podataka

Google Cloud Firestore baza podataka nudi robusno upravljanje pristupom i autentifikaciju kroz dve različite metode u zavisnosti od biblioteka koje se koriste. Za mobilne (Android, iOS) i web klijentske biblioteke koristi se Firebase Authentication servis i Cloud Firestore Security Rules za rukovanje autentifikacijom, autorizacijom i validacijom podataka bez servera, a za serverne klijentske biblioteke (Java, Python, Node.js, Go, C#, ...) koriste Cloud Identity and Access Management (IAM) kako bi se upravljalo pristupom bazi.

U ovom seminarskom pokazaćemo jedan od načina autentifikacije korisnika korišćenjem email-a i lozinke što je jedan od najjednostavnijih načina autentifikacije korisnika. Google Cloud Firestore pored ovakvog načina autentifikacije korisnika nudi i opcije koje su prikazane na slici 23.

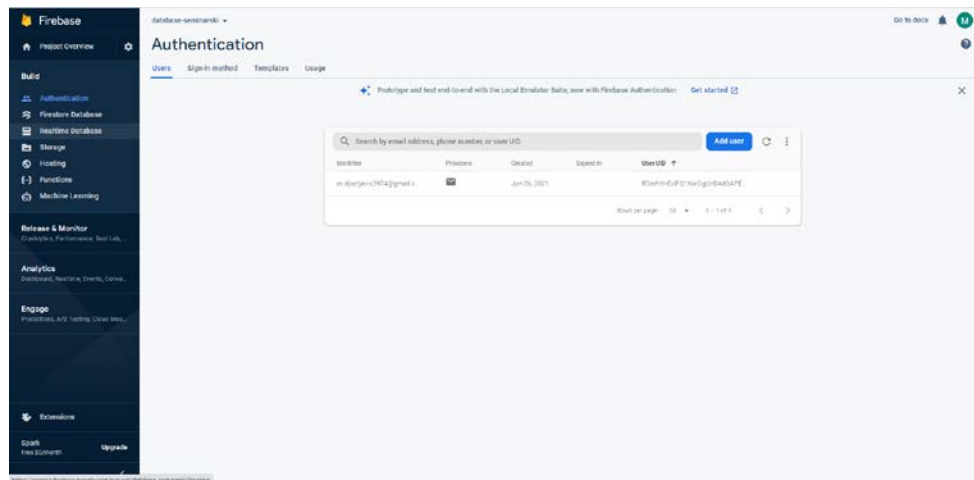


Provider	Status
Email/Password	Enabled
Phone	Disabled
Google	Disabled
Play Games	Disabled
Game Center	Disabled
Facebook	Disabled
Twitter	Disabled
GitHub	Disabled
Yahoo	Disabled
Microsoft	Disabled
Apple	Disabled
Anonymous	Disabled

Slika 23 : Prikaz načina autentifikacije korisnika na bazu podataka

3.7.1. Autentifikacija korisnika korišćenjem email i lozinke

Google Cloud Firestore baza podataka nudi jednostavne i već ugrađene mehanizme za autentifikaciju korisnika na bazu podataka. U sekciji Authentication u podsekciji Users moguće je kreirati korisnike naše baze podataka. Na slici je prikazana sekcija u kojoj možemo kreirati korisnike sistema.



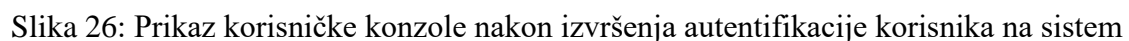
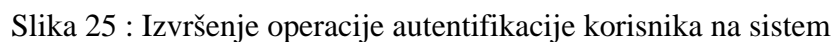
Slika 24: Sekcija Authentication kod Google Cloud Firestore baze podataka

Sa slike 24 možemo videti da u našem sistemu, pored samog administratora baze podataka imamo i još jednog korisnika sistema čiji su email i lozinka sačuvani u samoj bazi podataka.

Da bismo autentifikovali korisnika na sistem potrebno je izvršiti sledeću funkciju u našem projektu

```
firebase.auth().signInWithEmailAndPassword("m.djordjevic3974@gmail.com", "*****")
.then((userCredential) => {
  user = userCredential.user;
  console.log(user);
  alert("Successfully login "+user.email);
})
.catch((error) => {
  var errorCode = error.code;
  console.log(error);
  var errorMessage = error.message;
});
```

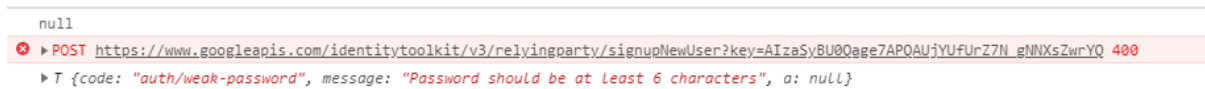
Rezultat autentifikacije korisnika na sistem dat je na slikama 25 i 26



3.7.2. Kreiranje korisničkih naloga korišćenjem email i lozinke

```
firebase.auth().createUserWithEmailAndPassword("mejl@email.com", "testkorisnik1")
    .then((userCredential) => {
        var korisnik = userCredential.user;
        console.log(korisnik);
        alert("Successfully created user "+korisnik.email);
    })
    .catch((error) => {
        var errorCode = error.code;
        var errorMessage = error.message;
        console.log(error);
    });
```

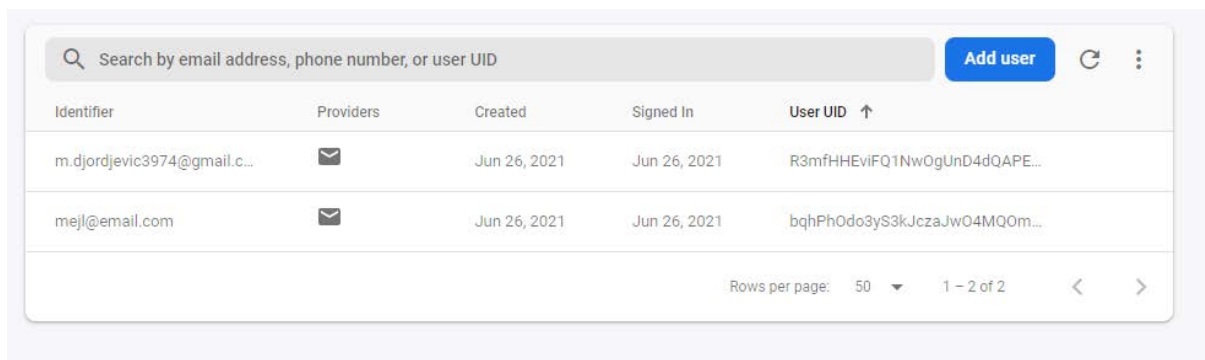
Ukoliko je došlo do greške prilikom registracije na sistem u konzoli prikazaće se rezultat izvršene operacije, kao na slici 27.



Slika 27 : Prikaz korisničke konzole nakon izvršene registracije korisnika na sistem

Sa slike možemo videti da prilikom kreiranja korisničkog naloga kao grešku sistem je prikazao nevalidnost lozinke. Lozinka mora sadržati minimum 6 karaktera.

Da bismo videli pravilno izvršenu operaciju registracije na sistem, potrebno je u sekciji Authentication u podsekciji Users proveriti da li je korisnik uspešno kreiran. Izgled naše registracije dat je na slici 28.



Slika 28: Prikaz korisnika sistema nakon uspešne autentifikacije korisnika.

3.8. Cena korišćenja

Za razliku od svog prethodnika Cloud Firestore baza ima drugačiji način naplaćivanja korišćenja. Dok većina baza naplaćuje samo utrošen prostor koji baza zauzima, u cenu korišćenja Firestore baze uračunava se i broj izvršenih operacija čitanja, upisa i brisanja, kao i količinu iskorišćenog bandwidth-a. Svaka funkcija get, set, update i delete predstavljaju jednu operaciju i tako se i naplaćuju. Kod čitanja naplaćuje se svaki pročitani dokument, a pošto se koriste listener-i svaki put kada se desi promena pročitaju se novi dokumenti tako da se i to naplaćuje. Naravno prilikom čitanja moguće je ograničiti broj dokumenata koji se čita, čime se može uštedeti novac.

Naravno Google nudi free plan koji za svaki projekat nudi 1 GB prostora za podatke, 50000 čitanja, 20000 upisa, 20000 brisanja po danu, kao i 10 GB bandwidth-a. Ukoliko se ovaj free plan pređe onda se naplaćuje prethodno navedeno po određenim cenama koje najviše zavise od regiona u kome se server na kome se projekat nalazi. Na slici 29. i 30. date su cene u slučaju izbora servera u Zurich-u i multi regiona u Evropi.

Zürich	
Pricing beyond the free quota	
Document reads	\$0.042 per 100,000 documents
Document writes	\$0.126 per 100,000 documents
Document deletes	\$0.014 per 100,000 documents
Stored data	\$0.210/GiB/month

Slika 29: Cena korišćenja Cloud Firestore baze podataka ukoliko se izabere server u Zurich-u

Europe (multi-region)	
Pricing beyond the free quota	
Document reads	\$0.06 per 100,000 documents
Document writes	\$0.18 per 100,000 documents
Document deletes	\$0.02 per 100,000 documents
Stored data	\$0.18/GiB/month

Slika 30: Cena korišćenja Cloud Firestore baze podataka ukoliko se izabere multi-regionalni način pamćenja podataka na serveru u Evropi

3.9. Najbolje prakse korišćenja

Saveti prilikom korišćenja Google Cloud Firestore baze podataka

- Odabir dobre lokacije koja je najbliža korisnicima, kako bi se obezbedilo manje kašnjenje i ukoliko želimo da maksimizujemo dostupnost naše aplikacije treba izabrati multi region.
- Treba voditi računa prilikom ručnog postavljanja ID-jeva dokumentima, jer treba izbegavati korišćenje specijalnih karaktera kao i korišćenje ID-jeva koji se ručno inkrementiraju (npr. User 1, User 2, User 3, ...).
- Izbegavati korišćenje preveliko korišćenje indeksa. Ovo se misli na custom kreirane indekse, ali ako ih je potrebno koristiti onda treba voditi računa i ne koristiti indekse kod propertija koji sadrže velike stringove ili ako su propertiji nizovi ili mape.
- Poželjno je ograničiti broj listenera po korisniku (optimalno 100 listenera po korisniku).
- Ne bi trebalo menjati jedan dokument više od jednom u sekundi, jer u slučaju da se dokument menja mnogo puta u sekundi može doći do većeg kašnjenja, tajmauta ili drugih grešaka.
- Kako bi se zaštitili od neželjenog pristupa podacima u bazi poželjno je koristiti sigurnosna pravila. Na primer: koristiti sigurnosna pravila kako bi se sprečio scenario da neželjeni korisnik više puta download-uje celu bazu čime direktno utiče na cenu.

4. Zaključak

Većina NoSQL baza podataka kao što je Firestore baza podataka imaju prednost sa aspekta performansi kao što su jednostavna implementacije i integraciju u bilo koji projekat, jednostavno sigurnosno kopiranje podataka i slično. Cloud Firestore baza podataka su relativno nove tehnologije i ne poseduju kvalitetnu dokumentaciju za razliku od SQL relacionih baza podataka koje se koriste već dugi niz godina. Za razliku od relacionih baza podataka Firestore baza podataka nema definisan način struktuiranja podataka, što korisnicima daje potpunu slobodu prilikom kreiranja sopstvenih aplikacije. Nedefinisana pravila struktuiranja podataka mogu dovesti do problema kod izvršenja upita nad samom bazom, filtriranjem podataka i analize postojećih podataka. Prema tome, Firestore baza podataka je prvenstveno napravljena za kreiranje jednostavnih aplikacijskih rešenja, zbog same slobode struktuiranja podataka može često doći i do konflikta prilikom pamćenja podataka. Pošto je Firestore relativno novija baza podataka postoje tendencije da će nadmašiti svoju konkurenciju i postati pravi izbor baze podataka.

Pored navedenih problema, korišćenjem cloud baza podataka ne samo da se obezbeđuje jednostavan pristup i korišćenje baze, već se otklanjaju i svi mogući problemi koji mogu da nastanu prilikom održavanja same baze. Mogućnost da se ne razmišlja o tome da li će naš sistem da podnese povećanje baze usled povećanja broja korisnika, kao i to da se ne mora voditi računa da će podaci biti izgubljeni u slučaju neke greške ili katastrofe, daju veliki plus u odnosu na relacione baze podataka. Sve ove prednosti koje nude ove baze dolaze za odgovarajuću cenu, iako većina kompanija nudi određeni period besplatno korišćenje, kasnije može doći do toga da cena za neke korisnike bude i previsoka. Ali uzimajući u obzir šta je sve uračunato u cenu može se reći da će cloud baze podataka polako u budućnosti postati prvi izbor prilikom odabira baze.