Elektronski fakultet u Nišu

Katedra za računarstvo

SEMINARSKI RAD

SISTEMI ZA UPRAVLJANJE BAZAMA PODATAKA

Sigurnost kod MongoDB baze podataka

Student:  
Vukadin Drašković, br. indeksa 1613

Sadržaj

[1. Uvod 3](#_Toc167026276)

[2. Sigurnost baza podataka 4](#_Toc167026277)

[2.1. Mehanizmi kontrole pristupa 4](#_Toc167026278)

[3. Sigurnost kod MongoDB baze podataka 8](#_Toc167026279)

[3.1. SCRAM 9](#_Toc167026280)

[3.2. Korisnici u MongoDB-u 12](#_Toc167026281)

[3.2.1. Localhost Exception 13](#_Toc167026282)

[3.2.2. Kreiranje korisnika 15](#_Toc167026283)

[3.3. Uloge u MongoDB-u 17](#_Toc167026284)

[3.3.1. Ugrađene uloge 17](#_Toc167026285)

[3.3.2. Korisnički definisane uloge 18](#_Toc167026286)

[3.4. Upravljanje korisnicima i ulogama u MongoDB-u 20](#_Toc167026287)

[4. Zaključak 26](#_Toc167026288)

[5. Literatura 27](#_Toc167026289)

# Uvod

Uspešno poslovanje jedne kompanije u mnogome zavisi od toga da li njeni klijenti mogu imati poverenja u nju. Privatni podaci koje klijenti dostavljaju kompanijama moraju ostati tajni, kako se ne bi narušila privatnost korisnika. Curenje ovakvih informacija samo jednog korisnika dovoljno je da kompanija izgubi svoju poziciju na tržištu.

Kako se podaci najčešće skladište u bazama podataka, dolazimo do zaključka da sigurnost baze podataka, pored ostalih stvari, može igrati ključnu ulogu u izgradnji poverenja između klijenata i kompanija. Ovaj seminarski rad će prvo obraditi osnovne pojmove sigurnosti i bezbednosti softverskih sistema, a zatim će opisati osnovne mehanizme kontrole pristupa bazi podataka. Nakon toga će biti obrađena sigurnost MongoDB baze podataka, gde će se na kraju primerima demonstrirati upravljanje samim MongoDB-om.

# Sigurnost softvera i baza podataka

Sigurnost baza podataka predstavlja skup osnovnih koncepata i praksi koje omogućavaju bezbednost i sigurnost baze. Trijumvirat **CIA** predstavlja vodeći princip kojim se pribegava prilikom projektovanja bezbednosti računarskih sistema [1]. Kako sistemi za upravljanje bazama podataka (engl. *Database Management System, DBMS*) predstavljaju jedan tip računarskog sistema, tako je moguće ovaj model primeniti na njih. Prema ovom modelu, softverski sistem mora obezbediti sledeće karteristike kako bi se smatrao sigurnim:

1. **Pouzdanost** (engl. *Confidentiality*) - nosi se sa zabranom pristupa informacijama neautorizovanim korisnicima, odnosno definiše ko ima dozvolu čitanja određenih podataka.
2. **Integritet** (engl. *Integrity*) - pod integirtetom se podrazumeva sprečavanje neovlašćene modifikacije podataka.
3. **Dostupnost** (engl. *Availability*) - podaci koji se koriste moraju biti uvek dostupni korisnicima. DoS (engl. *Denial-of-Service*) napad predstavlja sajber napad zlonamernog korisnika koji ima za cilj da obori mašinu ili mrežu na kojoj se nalazi server kojem korisnici pristupaju, kako bi onemogućio pristup podacima. Taj server, koji predstavlja metu napada, može biti upravo server baze podataka.

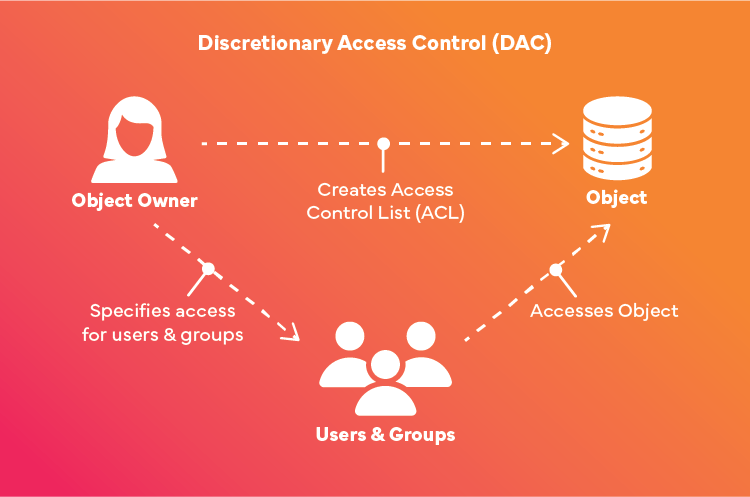
Dva koncepta koja takođe predstavljaju osnovu sigurnosti baza podataka su **autentifikacija** (engl. *Authentication*) i **autorizacija** (engl. *Authorization*). Pod autentifikacijom se podrazumeva da će korisnik biti identifikovan prilikom pristupa bazi podataka, odnosno da će se znati ko joj pristupa, dok se pod autorizacijom podrazumeva da se prilikom izvršenja određenih akcija u bazi podataka ispituje da li korisnik (koji je prethodno autentifikovan) ima pravo na njihovo izvršenje. Autentifikacija i autorizacija se jednom rečju zovu **kontrola pristupa** (engl. *Control* *Access*).

Dobra praksa je da svaki sistem ima jednog ili više administratora koji će imati maksimalne privilegije u sistemu, odnosno koji će imati dozvolu da izvrše bilo koju naredbu. Administrator sistema mora biti pažljiv prilikom upravljanja, jer može izvršiti neke komande koje mogu biti fatalne za sistem (npr. naredba za brisanje čitavih baza ili tabela u DBMS-u).

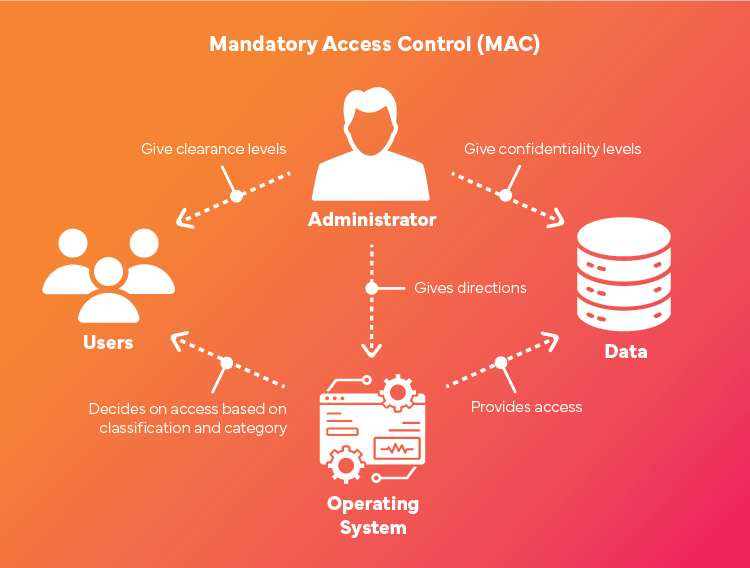
## Mehanizmi kontrole pristupa

Najčešće korišćeni mehanizmi kontrole pristupa u sistemima za upravljanje bazama podataka su diskreciona kontrola pristupa (engl. *Discretionary access control*, DAC), obavezna kontrola pristupa (engl. *Mandatory access control*, MAC) i kontrola pristupa bazirana na ulogama (engl. *Role-Based access control*).

**Diskreciona kontrola pristupa** omogućava vlasniku resursa u bazi podataka da upravlja kontrolom pristupa tom resursu, što znači da vlasnik resursa odlučuje ko i na koji način može pristupati tim resursima [2]. Ovim se upravljanje kontrole pristupa prenosi sa administratora na korisnika. Na slici 2.1 je prikazano funkcionisanje ovog mehanizma. Vlasnik resursa kreira listu kontrole pristupa (engl. *Access Control List*), poznatiju kao ACL listu, kojom definiše ko će imati pristup resursima čiji je on vlasnik i na koji način. Kada korisnici žele da pristupe tim resursima, prvo će se obaviti pristup ACL listi kako bi se utvrilo da li oni uopšte imaju pravo pristupa tom resursu, i ako imaju, koje su im privilegije dodeljenje nad tim resursom. Privilegije se u gruboj podeli mogu podeliti na privlegije čitanja podataka, modifikacije (upisa, ažuriranja i brisanja) podataka i administracije, odnosno upravljanja samim resursom.

  
Slika 2.1. Prikaz funkcionisanja DAC mehanizma

**Obavezna kontrola pristupa** je mehanizam koji funkcioniše pod striktnim sistemski definisanim pravilima [2]. Kod ovog modela, sistem je taj koji kontroliše kojim resursima korisnici mogu pristupiti i na koji način. Svaki resurs u DBMS-u ima svoj nivo sigurnosti, dok se za korisnika definiše nivo dozvole. Koji nivoi sigurnosti postoje u DBMS-u definiše sam DBMS, a najčešće se sreću *Top Secret* (TS), *Secret* (S), *Confidential* (C) i *Unclassified* (U), pri čemu je TS nivo najveće sigurоsti, zatim ide S, C i na kraju U kao nivo najmanje sigurnosti. Na slici 2.2 je prikazan način funkcionisanja ovog mehanizma. Administrator dodeljuje nivoe sigurnosti resursima u sistemu, i daje određene dozvole korisnicima. Kada korisnici pristupaju resursima, proverava se njhov nivo dozvole sa nivoom sigurnosti korisnika. Ukoliko se zahteva čitanje, vrši se provera da li je nivo sigurnosti korisnika veći ili jednak nivou sigurnosti resursa. Prilikom upisa, proverava se da li je nivo sigurnosti resursa veći ili jednak nivou sigurnosti korisnika. Ovo pravilo prilikom upisa može na prvi pogled delovati nelogično, ali je ideja da korisnici sa većim privilegijama u sistemu ne mogu da modifikuju resurse sa nižim privilegijama, jer bi time korisnici sa nižim privilegijama mogli da vide podatke korisnika sa višim privilegijama.

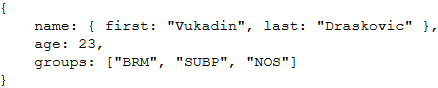
  
Slika 2.2. Način funkcionisanja MAC mehanizma na primeru operativnog sistema

**Kontrola pristupa bazirana na ulogama** [3] je još jedan mehanizam kontrole pristupa. U sistemu se definišu uloge koje mogu imati određene privilegije nad resursima u bazi podataka, a zatim administrator te uloge dodeljuje korisnicima kako bi im ograničio kontrolu pristupa. Uloge se najčešće kreiraju tako da predstavljaju prave uloge pojedinaca u nekoj kompaniji - na primer, ukoliko u kompaniji postoje sektori za menadžment, finansije, razvoj, testiranje i logistiku, dobra je praksa da se kreiraju posebne uloge u sistemu za svaki od ovih sektora. Korisniku se može dodeliti više uloga. Administrator sistema bi se ovim mehanizmom kreirao kao i svaki drugi korisnik, s tim što bi mu bila dodeljena uloga sa maksimalnim privilegijama.

Kako je tema ovog seminarskog rada sigurnost MongoDB baze podataka, treba reći da MongoDB koristi kontrolu pristupa baziranu na ulogama.

# Sigurnost kod MongoDB baze podataka

MongoDB [4] je tipičan predstavnik dokument NoSQL (*Not only SQL*) baza podataka, gde se svaki rekord baze podataka (analogno torci, odnosno slogu u relacionim bazama podataka) skladišti u posebnom dokumentu. Podaci koje skladišti MongoDB su slični JSON (*JavaScript Object Notatio*n) objektima, gde su podaci organizovani u obliku *ključ: vrednost* (Slika 3.1). Vrednost određenog ključa može biti niz, neki drugi dokument, ili čak niz dokumenata. Podaci su organizovani u kolekcije koje su analogne tabelama u relacionim bazama podataka, dok je baza podataka zadužena za upravljanje većim brojem kolekcija.

  
Slika 3.1. Struktura dokumenta u MongoDB-u

MongoDB nudi podršku za autentifikaciju, kao i za autorizaciju. Mehanizmi za autentifikaciju [5] koje podržava MongoDB su:

1. SCRAM (*Salted Challenge Response Authentication Mechanism*) – podrazumevani mehanizam autentifikacije u najnovijoj verziji MongoDB-a 7.0
2. Autentifikacija na osnovu x.509 sertifikata
3. Kerberos
4. LDAP Proxy
5. OpenID Connect

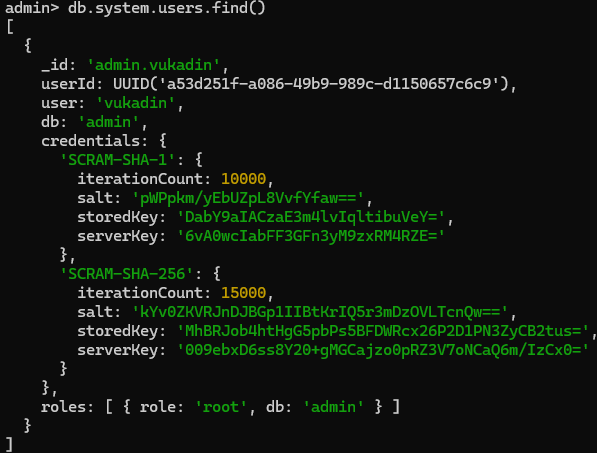
Kontrola pristupa se u MongoDB-u može omogućiti na dva načina [6] – navođenjem *--auth* opcije prilikom pokretanja instance MongoDB-a, ili promenom konfiguracionog fajla *mongod.conf.orig* u folderu */etc* dodavanjem dela prikazanog na slici 3.2. Inicijalno je kontrola pristupa onemogućena.

  
Slika 3.2. Izmena konfiguracionog fajla MongoDB-a radi aktivacije kontrole pristupa

Autentifikacija korisnika se u MongoDB-u odvija na osnovu korisničkog imena korisnika i njegove lozinke, dok se autorizacija uspostavlja na osnovu uloga (engl. *role*), odnosno rola, koje definišu koji tip privilegija (privilegije za čitanje, modifikaciju, brisanje podataka, kreiranje novih kolekcija, itd.) korisnik može da ima nad određenim resursom, gde resurs može predstavljati čitavu bazu podataka, određeni skup kolekcija unutar baze ili pojedinačnu kolekciju.

## 3.1. SCRAM

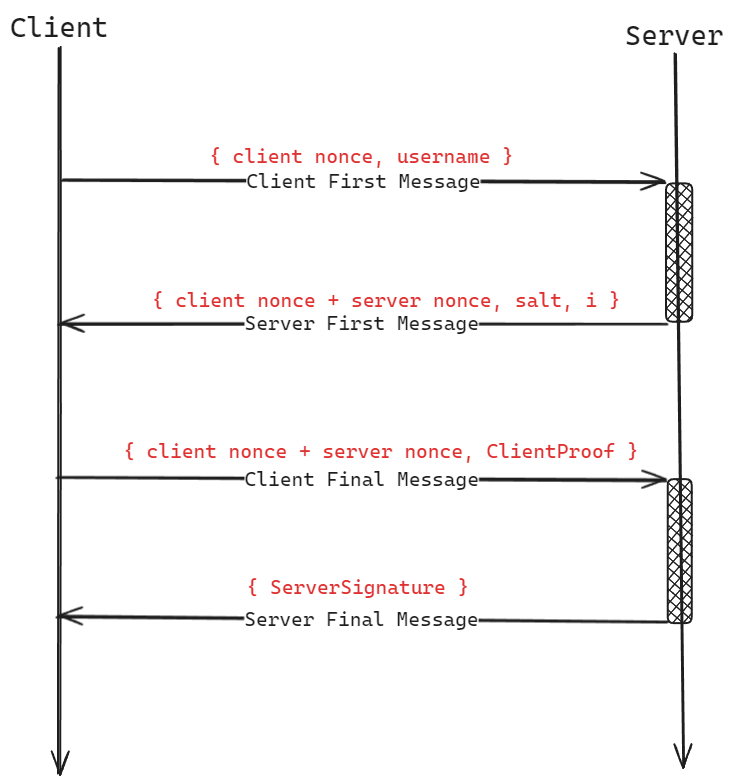
Kao što je već rečeno, podrazumevani mehanizam autentifikacije u MongoDB-u je SCRAM. **SCRAM** je bidirekcionalni mehanizam autentifikacije (potvrda identiteta obe strane u komunikaciji) baziran na IETF RFC 5802 standardu.

  
Slika 3.3. *Credentials* atribut predstavlja podatke neophodne za funckionisanje SCRAM-a

Kada korisnik želi da se autentifikuje, od njega se zahteva unos korisničkog imena (engl. *username*), lozinke (engl. *password*) i baze autentifikacije (inicijalna baza podataka kojoj pripada korisnik). Iako se od korisnika zahteva unos lozinke, u bazi podataka se nigde ne skladišti lozinka korisnika (čak ni u enkriptovanom obliku), čime se sprečavaju bilo kakvi pokušaji krađe korisnikove lozinke. Ipak, za korisnika se čuvaju informacije neophodne za kreiranje izazova (engl. *challenge*) kojeg korisnik treba da reši kako bi se uspešno autentifikovao. Te informacije su broj iteracija (engl. *iterationCount*) neophodan za generisanje potvrde klijenta, i *salt* koji označava kojim se skupom karaktera treba „začiniti” (engl. *digest*) lozinka korisnika kako bi se generisala potvrda. Takođe, server čuva i *storedKey* i *serverKey* koji učestvuju u bidirekcionalnoj autentifikaciji. U MongoDB-u SCRAM može da koristi neku od SHA-1 ili SHA-256 heš funkcija. Na slici 3.3 je prikazano da se za korisnika u bazi podataka nigde ne čuva *password*, već samo prethodno navedeni podaci.

Način funkcionisanja SCRAM mehanizma autentifikacije je prikazan na slici 3.4.

Na početku, klijent inicira komunikaciju slanjem svoje prve poruke *Client First Message* koja se sastoji od *nonce* (skraćeno od *number once*) dela klijenta i username-a za koji klijent želi da se autentifikuje. Nonce predstavlja podatak koji služi da jedinstveno identifikuje komunikaciju klijenta i servera, a karakteristika mu je da se jednom iskorišćeni nonce ne sme ponovo koristiti u nekoj narednoj autentifikaciji korisnika.

  
Slika 3.4. Razmena poruka u SCRAM

Kada Server prihvati Client First Message od klijenta, na osnovu username-a u bazi podataka traži *salt* i broj iteracija *i* neophodan za uspešno autentifikovanje korisnika. Nakon toga, server u *Server First Message* klijentu šalje salt, broj iteracija i nonce, koji se dobija konkatenacijom klijentskog nonce-a i serverskog nonce-a koji on sam kreira. Upravo ovi podaci predstavljaju potrebne informacije kako bi klijent mogao da se autentifikuje po SCRAM mehanizmu.

Po prijemu Server First Message poruke, klijent počinje da gradi *ClientProof* podatak koji treba poslati u *Client Final Message* poruci na sledeći način:

SaltedPassword = Hi(password, salt, i)

ClientKey = HMAC(SaltedPassword, “Client Key”)

StoredKey = H(ClientKey)

AuthMessage se gradi na osnovu nonce-a

ClientSignature = HMAC(StoredKey, AuthMessage)

ClientProof = ClientKey XOR ClientSignature

Broj iteracija *i* ima ulogu u funkciji *Hi(str, salt, i)* koja je definisana sledećim skupom koraka:

U1 = HMAC(str, salt + INT(1))

U2 = HMAC(str, U1)

...

Ui-1 = HMAC(str, Ui-2)

Ui = HMAC(str, Ui-1)

Hi = U1 XOR U2 XOR ... XOR Ui.

HMAC predstavlja MAC (*Message Authentication Code*, srp. *autentifikacioni kod poruke*) koji se generiše na osnovu heš funkcije i tajnog ključa, gde tajni ključ predstavlja parametar *str*, dok drugi parametar predstavlja podatak za autentifikaciu. Odatle se može zaključiti da se prilikom računanja *SaltedPassword*-a *password* koristi kao tajni ključ. *H* predstavlja upravo heš funckiju koju koristi SCRAM, što u slučaju MongoDB-a, kao što je već rečeno, može biti SHA-1 ili SHA-256.

Treba obratiti pažnju da će *AuthMessage* biti različit prilikom svakog pokušaja registrovanja korisnika, zbog njegovog generisanja na osnovu nonce-a, čime se sprečava zlonameran pokušaj krađe identiteta (zlonamerni korisnik može da iskoristi nonce koji je klijent koristio prilikom registrovanja na sistem, ali će server znati da je u pitanju napad jer je taj nonce već jednom iskorišćen).

Prijemom Client Final Message poruke, server pokušava da autentifikuje klijenta računanjem *ClientSignature*-a na osnovu *StoredKey*-a (koji se za korisnika čuva u bazi) i na osnovu *AuthMessage* kojeg server generiše na osnovu nonce-a. Kada server kreira *ClientSignature*, obavlja XOR operaciju sa primljenim *ClientProof*-om iz Client Final Message poruke i dobija *ClientKey*, čijim heširanjem dobija *StoredKey* koji treba da se poklopi sa *StoredKey* vrednošću koju čuva u bazi za korisnika. Ukoliko se vrednosti poklope, korisnik je uspešno autentifikovan i server šalje *Server Final Message* poruku. U suprotnom, došlo je do neuspešne autentifikacije.

Server Final Message se sastoji od *ServerSignature*-a koji se dobija na sledeći način:

ServerSignature = HMAC(ServerKey, AuthMessage)

SCRAM je mehanizam koji obezbeđuje bidirekcionu autentifikaciju, što znači da i klijent može da autentifikuje server. Kada klijent primi Server Final Message, on računa *ServerSignature* na sledeći način:

ServerKey = HMAC(SaltedPassword, “Server Key”)

ServerSignature = HMAC(ServerKey, AuthMessage)

gde je *SaltedPassword* već jednom izračunat prilikom generisanja *ClientProof* podatka za potrebe Client Final Message poruke. Ukoliko se kreirani *ServerSignature* poklop sa pristiglim, onda znači da je ispravan server učestvovao u registraciji. U suprotnom, server je neuspešno autentifikovan od strane klijenta, čime klijent odbija nastavak komunikacije sa serverom. U ovoj razmeni poruka, klijent nije nijednom poslao svoju lozinku u obliku koji je čitljiv za server, tako da je klijent zaštićen od zlonamernog napada u kome je korišćen lažni server.

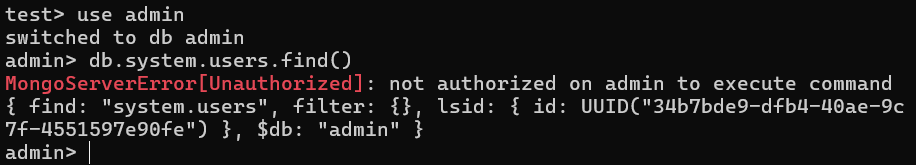
## 3.2. Korisnici u MongoDB-u

Ukoliko nismo omogućili kontrolu pristupa u MongoDB-u, korisnik nije dužan da se identifikuje pre izvršenja naredbi u bazi podataka. U suprotnom, ako je kontrola pristupa aktivirana, svaki korisnik je dužan da se autentifikuje na osnovu svog korisničkog imena, lozinke i baze. Korisnici se u MongoDB-u kreiraju *createUser* naredbom, a autentifikuju se ili prilikom povezivanja na bazu, ili nakon povezivanja izvršenjem *auth* naredbe. Za korisnika je potrebno da se specificira njegova uloga u bazi, gde uloga označava skup privilegija za koje su omogućene određene akcije nad resursima MongoDB-a.

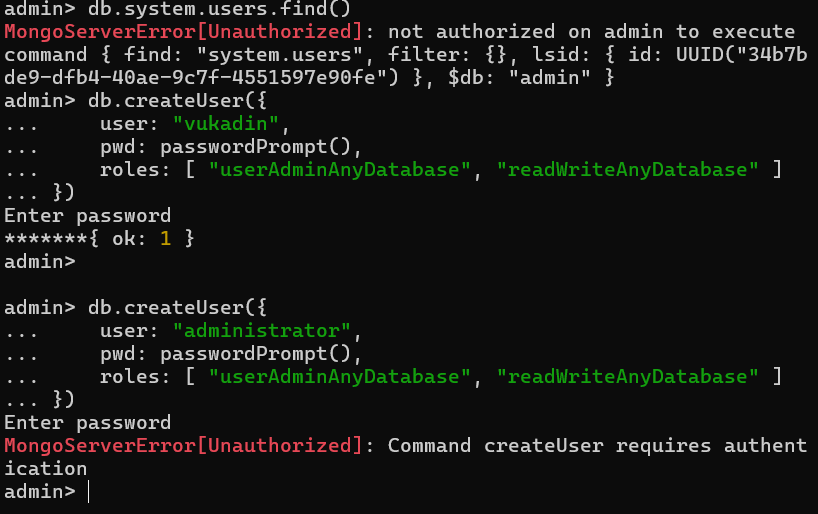
Dobra je praksa da svaki korisnik baze podataka ima svoj nalog, gde se u korisnike pored ljudi svrstavaju i aplikacije koje koriste MongoDB. Kao što je već rečeno u uvodnom poglavlju, očekivano je da postoji administrator baze podataka koji će imati najveće privilegije u bazi podataka koji će biti zadužen za upravljanjem celog sistema, odnosno samim podacima u bazi, korisnicima, rolama, itd...

### 3.2.1. Localhost Exception

Aktiviranjem kontrole pristupa se zahteva da korisnici koji se povežu na bazu moraju biti autentifikovani. Kada se prvi put pokrene baza podataka, u sistemu ne postoji nijedan korisnik koji bi mogao da se autentifikuje kako bi mogao da izdaje naredbe. Tu nastaje problem jer nijedan korisnik nije u stanju da upravlja bazom podataka ili dodaje podatke u istu zbog nemogućnosti autentifikovanja, što je prikazano na slici 3.5. Ali i za to postoji rešenje.

  
Slika 3.5. Nemogućnost izvršenja naredbi u bazi podataka sa omogućenom kontrolom pristupa pri nepostojanju korisnika

Prilikom prvog povezivanja na MongoDB, kada je kontrola pristupa omogućena, dešava se *localhost exception* [8] koji omogućava da se kreira inicijalni korisnik u *admin* bazi podataka. Najčešća praksa je da se na ovaj način kreira nalog administratora.

  
Slika 3.6. Localhost exception

Na slici 3.6. je prikazan *localhost exception*. Uočavamo da nije dozovljeno izvršavanje drugih naredbi u *admin* bazi podataka, osim naredbe za kreiranje korisnika. Nakon kreiranja inicijalnog korisnika nije moguće kreirati novog, zato što MongoDB zna da sada u sistemu već postoji korisnik koji se može autentifikovati, i ne dozvoljava da se bez autentifikacije izvršava bilo koja naredba u sistemu, jer je kontrola pristupa omogućena. Na slici 3.7 je prikazano da je nakon autentifikovanja kao novokreirani korisnik moguće izvršavati naredbe u sistemu. Konkretno, naredba *db.system.users.find()* u admin bazi podataka ima za cilj da prikaže sve korisnike koje postoje u MongoDB-u. U trenutnoj listi korisnika baze podataka postoji samo admin korisnik koji je kreiran u prethodnom primeru.

  
Slika 3.7. Logovanje u sistem kao novokreirani korisnik i mogućnost izvršenja naredbi

Naredba *auth* koja je prikazana na slici 3.7 omogućava logovanje na MongoDB [9]. Može se pozvati na više različitih načina:

1. Navođenjem samo korisničkog imena korisnika nakon čega se od korisnika zahteva unos lozinke.

admin> db.auth(“vukadin”)

1. Navođenjem korisničkog imena korisnika uz *passwordPrompt()* argument koji od korisnika eksplicitno zahteva da unese lozinku. Ekvivalenat je prvom načinu izvršenja *auth* naredbe.

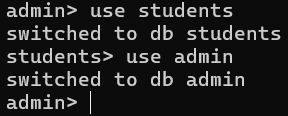
admin> db.auth(“vukadin”, passwordPrompt())

1. Navođenjem korisničkog imena i lozinke. Ovaj metod se izbegava jer je korisnikova lozinka vidljiva u terminalu prilikom autentifikovanja što je prikazano na slici 3.8, čime se narušava privatnost korisnika, tako da se je preporučeno da se koristi neki od prethodna dva načina.

admin> db.auth(“vukadin”, “vukadin”)

### Kreiranje korisnika

Za potrebe kreiranja korisnika u MongoDB-u postoji *createUser* naredba. Korisnik će se kreirati u onoj bazi podataka u kojoj se terminal nalazi pri izvršenju *createUser* komande. Naredba koja nam omogućava da pređemo iz jedne u drugu bazu podataka je *use <database>*, koja je prikazana na slici 3.8. Pri promeni baze podataka, menja se *prompt* (početni deo linije terminala) koji signalizira u kojoj bazi podataka se trenutno nalazimo.

  
Slika 3.8. *use* naredba

Naredba createUser ima sledeći format:

db.createUser(user, writeConcern)

gde prvi argument *user* predstavlja sve podatke o korisniku za koga se kreira nalog, dok drugi argument *writeConcern*, koji je za razliku od prvog opiconi, predstavlja nivo potvrde za *write* operacije korisnika u MongoDB-u.

Argument *user* predstavlja MonogDB dokument koji može imati sledeći oblik:

{

user: “<name>”,

pwd: passwordPrompt(), // ili “<plaintext password>”

customData: { <any information> },

roles: [

{ role: “<role>”, db: “<database>” } | “<role>”,

…

],

authenticationRestrictions: [

{

clientSource: [“<IP>” | “<CIDR range>”, …],

serverAddress: [“<IP>” | “<CIDR range>”, …]

},

…

],

mechanism: [“<SCRAM-SHA-1|SCRAM-SHA-256>”, …],

passwordDigestor: “<server|client>”

}

Polja koja predstsavljaju *user* argument su:

1. **user** - podatak koji je tipa *string* i predstavlja korisničko ime.
2. **pwd** - *string* podatak koji predstavlja lozinku korisnika. Postoje dva načina za postsavljanje lozinke - prvi način je da korisnik postavi šifru u čitljivom obliku, što nije preporučljivo jer će se lozinka videti na ekranu čime se narušava bezbednost. Drugi način je da se za *pwd* polje ostavi poziv *passwordPrompt()* metode koja zahteva od korisnika da unese šifru nakandano. Kada se naknadno unosi šifra u terminal, karakteri nisu vidljivi na ekranu, čime se ne ugrožava bezbednost.
3. **customData** - ovo polje je tipa *document* i opciono je. Očekuje se da se u customData navedu informacije koje će bolje opisati korisnika za koga se kreira nalog.
4. **roles** - predstavlja niz rola koje korisnik može da ima. Uloge mogu biti ugrađene ili definisane od strane korisnika. Ovo polje nije opciono, i moramo navesti prazan niz, odnosno *[]* ukoliko korisnik nema role. Role se mogu navoditi tako što će se navesti dokument koji sadrži dva polja, prvo polje koje definiše rolu, a drugo polje koje predstavlja za koju bazu podataka važi rola. Drugi način je da se samo navede naziv role, gde se podrazumeva da korisnik dobija određenu ulogu u bazi podataka u kojoj je kreiran.
5. **authenticationRestrictions** - predstavlja niz ograničenja gde *clientSource* definiše sa kojih IP adresa, ili iz kog opsega IP adresa se korisnik može prijaviti na bazu podataka, dok *serverSource* predstavlja opseg adresa, ili adresu sa koje server može da prihvati korisnika (ima primenu u distribuiranom korišćenju MongoDB-a na klasteru).
6. **mechanism** - opciono polje koje definiše koji se mehanizam koristi za autentifikaciju korisnika. Podrazumevano se koriste “*SCRAM-SHA-1“* i “*SCRAM-SHA-256“*, tako da korisnik može da se loguje na sistem koristeći oba mehanizma. Ostali mehanizami koje se mogu navesti su “*MONGODB-X509*“, “*GSSAPI*“, “*PLAIN*“ i “*MONGODB-OIDC*“.
7. **passwordDigestor** - opciono string polje koje može uzimati isključivo dve vrednosti - prva vrednost je “*server*“, koja je i podrazumevana, dok je druga vrednost “*client*“. Označava ko će „začiniti“ lozinku prilikom kreiranja korisnika.

## Uloge u MongoDB-u

Uloge se, kako u ostalim bazama podataka, tako i u MongoDB-u, definišu kao skup privilegija, gde svaka privilegija označava koji skup akcija se može izvršiti nad definisanim resursima. MongoDB ima **ugrađene role** (engl. *Built-In Roles*) koje se mogu dodeliti korisnicima, a postoji mogućnost kreiranja **korisnički definisanih uloga** (engl. *User-Defined Roles*) ukoliko za tim ima potrebe.

### 3.3.1. Ugrađene uloge

Ugrađene uloge [11] se u MongoDB-u mogu podeliti na korisničke i administratorske uloge koje važe za sve baze podataka, i ostale uloge koje se mogu dodeliti korisniku isključivo nad *admin* bazom podataka.

Korisničke uloge (engl. *Database User Roles*) koje se mogu dodeliti nad svim bazama podataka su:

1. **read** - omogućava čitanje podataka iz kolekcija u bazi podataka, prikaz indeksa i kolekcija koje postoje u bazi, kao i nekih statističkih parametara.
2. **readWrite** - na skup privilegija koje podrazumeva *read* uloga dodaje i privilegije za modifikaciju podataka, kao i privilegije za kreiranje i modifikaciju indeksa i kolekcija.

Korisničke uloge ne omogućavaju administraciju baze podataka, već samo čitanje i upravljanje podacima u njoj. Uloge za potrebe administracije baze podataka (engl. *Database Administration Roles*) su:

1. **dbAdmin** - omogućava modifikaciju šeme i indeksa baze podataka, kao i prikupljanje određene statistike. Ne uključuje privilegije za upravljanje korisnicima i ulogama. Nad nesistemskim bazama ne omogućava *find* (srp. *pronaći*) privilegiju za čitanje podataka. Jedina sistemska baza nad kojom se može dodeliti ova uloga je *system.profile*, koja služi da čuva istoriju komandi koje su se izvršile nad instancom MongoDB-a.
2. **dbOwner** - predstavlja skup privilegija koje omogućavaju *dbAdmin*, *readWrite* i *userAdmin* uloge.
3. **userAdmin** - isključivo omogućava privilegije za kreiranje korisnika i uloga u bazi podataka, kao i modifikaciju istih.

MongoDB sadrži ugrađene uloge za administraciju klastera (engl. *Cluster Administration Roles*), s obzirom na to da nudi podršku za rad u distribuiranom okruženju na klasteru uređaja. Ove uloge se isključivo mogu primeniti nad *admin* bazom podataka, a to su **clusterManager**, **clusterMonitor**, **hostManager** i **clusterAdmin**, koji uključuje sve privilegije prethodnih uloga za administraciju klastera.

Za potrebe kreriranja rezervnih kopija (engl. *backup*) i oporavka baze podataka (engl. *restore*), postoje uloge **backup** i **restore**. Ove uloge se moraju primeniti nad *admin* bazom podataka, a predstavljaju skup privilegija koje su neophodne za izvršenje akcija vezanih za kreiranje rezervnih kopija i oporavka svih baza podataka.

Uloge koje su dostupne u *admin* bazi podataka, a čije se privilegije primenjuju na sve baze podataka (engl. *All-Database Roles*) u sistemu, izuzev na *local* i *config* bazu su:

1. **readAnyDatabase** - omogućava privilegije *read* uloge nad svim bazama podataka u sistemu.
2. **readWriteAnyDatabase** - na skup privilegija *readAnyDatabase* uloge dodaje privilegije *readWrite* uloge koja se može primeniti nad svim bazama podataka.
3. **userAdminAnyDatabase** - predstavlja skup privilegija *userAdmin* uloge koje se mogu primeniti nad svim bazama podataka.
4. **dbAdminAnyDatabase** - skup privilegija *dbAdmin* uloge koje se mogu primeniti nad svim bazama podataka u sistemu.

Ukoliko korisnik dobije **root** ulogu nad *admin* bazom podataka, to znači da ima sve privilegije nad svim bazama u sistemu. S toga ovaj korisnik predstavlja glavnog administratora MongoDB-a (engl. *superuser*).

### 3.3.2. Korisnički definisane uloge

Iako MongoDB nudi veliki broj ugrađenih rola, postoji šansa da nijedna od njih ne zadovoljava naše potrebe, odnosno ne obuhvata skup privilegija koji želimo da dodelimo korisniku nad nekim resursom. S toga postoji mogućnost kreiranja korisnički definisanih uloga [12]. Za te potrebe postoji *createRole* naredba [13]. Rola se kreira u bazi podataka u kojoj se terminal nalazio u trenutku izvršenja *createRole* komande. Kako bi jedinstveno identifikovao ulogu, MongoDB koristi naziv uloge i bazu podataka u kojoj uloga važi. Ovo omogućava kreiranje uloga sa istim nazivom u različitim bazama podataka.

Naredba *createRole* ima sledeći oblik:

db.createRole(role, writeConcern)

gde je prvi argument *role* namenjen za navođenje ključnih stvari u ulozi koja se kreira, dok je drugi argument isti kao i kod *createUser* naredbe. Argument role predstavlja dokument koji može imati sledeći oblik:

{

role: “<name>”,

privileges: [

{ resource: { <resource> }, actions: [ “<action>”, …] },

…

],

roles: [

{ role: “<role>”, db: “<database>” } | “<role>”,

…

],

authenticationRestrictions: [

{

clientSource: [“<IP>” | “<CIDR Range>”, …],

serverSource: [“<IP>” | “<CIDR Range>”, …]

},

…

]

}

Polja koja predstavljaju *role* argument su:

1. **role** - *string* podatak koji predstavlja naziv uloge koja se kreira.
2. **privileges** - niz koji predstavlja skup privilegija koje će dobiti novokreirana uloga. Ovo nije opcioni argument, tako da je neophodno ostaviti prazan niz u slučaju kada uloga nema privilegije. Elementi ovog niza su dokumenti koji mogu imati dva polja - prvo polje *resource*, koji definiše nad kojim resursom se dodeljuje privilegija, i drugo polje *actions*, koje predstavlja niz podataka tipa string, gde svaki od njih označava koja je akcija omogućena nad resursom. Resurs može biti cela baza podataka, a može biti i kolekcija unutar baze. Neke od najčešće korišćenih akcija su *find*, *insert*, *remove*, *update*, *createRole*, *grantRole*, *dropRole*, *createUser*, *dropUser*, *createCollection*, *dropCollection*...
3. **roles** - predstavlja niz uloga od kojih će novokreirana uloga naslediti privilegije. Uloge čije privilegije novokreirana rola nasleđuje mogu biti i korisnički definisane, i ugrađene. Argument nije opcioni i analogno *privileges* polju zahteva prazan niz ukoliko uloga ne nasleđuje privilegije drugih rola.
4. **authenticationRestrictions** - jedini opcioni argument. Predstavlja niz ograničenja koja definišu opseg IP adresa sa kojih se može pristupati resursima, analogno istom polju u *createUser* naredbi.

Za potrebe prikaza svih uloga koje postoje u bazi podataka u kojoj se treuntno nalazi terminal postoji *getRoles* komanda [14]. Ova komanda za argument prihvata dokument, koji je opcioni, sa sledećim poljima:

1. **rolesInfo** - ukoliko se navodi dokument koji je argument *getRoles* komande, onda se ovo polje mora navesti. Celobrojnog je tipa i ukoliko se za vrednost stavi 1 komanda će vratiti korisnički definisane uloge.
2. **showAutentificationRestrictions** - opcioni *boolean* argument kojim se označava da li će komanda prikazati *authenticationRestrictions* za ulogu. Podrazumevano je postavljen na *false*.
3. **showBuiltInRoles** - opciona *boolean* vrednost koja određuje da li će se prikazati ugrađene role.
4. **showPrivileges** - opcioni argument koji može uzimati *true* ili *false* vrednost. Vrednost ovog argumenta određuje da li će se za ulogu prikazati privilegije ili ne. Podrazumevana vrednost je *false*.

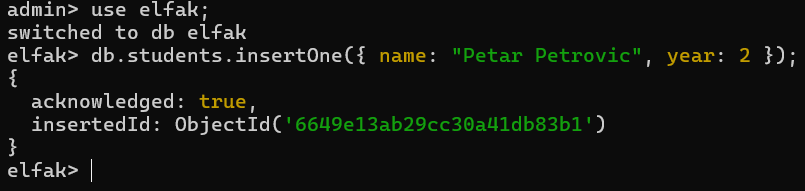
Ukoliko se komanda getRoles izvrši bez argumenata, onda će vratiti sve korisnički definisane uloge u bazi.

## Upravljanje korisnicima i ulogama u MongoDB-u

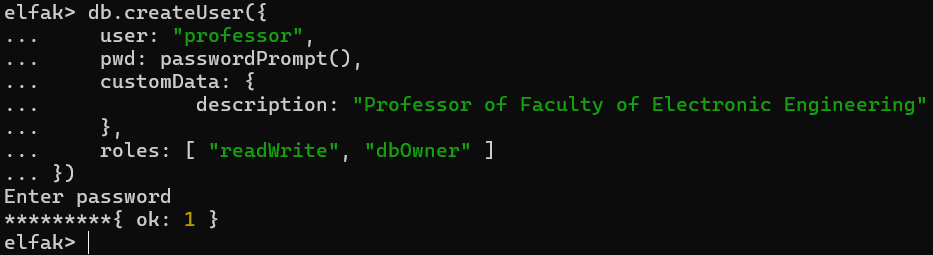
U ovom poglavlju će biti demonstrirano upravljanje korisnicima i ulogama u MongoDB-u na primerima. U poglavlju *3.2.1. Localhost Exception* je prikazano kreiranje *admin* korisnika.

Za početak će admin MondoDB-a kreirati bazu podataka pod nazivom *elfak* koje će služiti za demonstraciju. U kolekciji *students* novokreirane baze podataka su uneti podaci o jednom studentu (slika 3.9), kako bi baza podataka bila prikazana u listi baza prilikom izvršavanja komande *show databases* (u MongoDB-u se baza kreira *use* komandom, ali ona neće postojati sve dok se u nju ne smeste neki podaci).

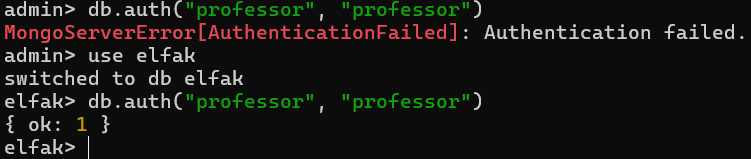
U ovoj bazi podataka će se kreirati dva korisnika - prvi će biti sa korisničkim imenom *professor*, dok će drugi imati korisničko ime *student*.

  
Slika 3.9. Kreiranje *elfak* baze podataka

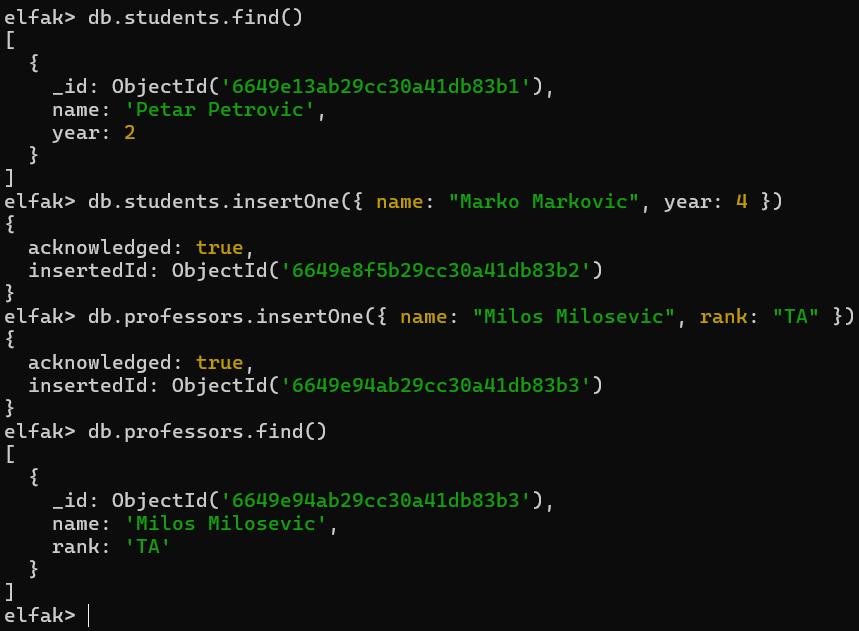
Korisnik *professor* će imati *readWrite* ulogu nad *elfak* bazom podataka (prikazano na slici 3.10). Ovim se omogućava da profesor može da dodaje nove studente u kolekciju *students*, ali isto tako i da kreira nove kolekcije u *elfak* bazi. Druga uloga koju će imati profesor je administratorska uloga *dbOwner*, kako bi mogao da u potpunosti upravlja *elfak* bazom bez intervencije administratora. Prilikom kreiranja ovog korisnika, terminal je bio pozicioniran u *elfak* bazi, tako da nema potrebe da se prilikom dodele uloga naglašava nad kojom bazom podataka se primenjuju.

  
Slika 3.10. Kreiranje *professor* korisnika

Na slici 3.11. je prikazano da logovanje na sistem kredencijalima korisnika *professor* neće uspeti. Razlog tome je što je terminal bio pozicioniran u *admin* bazi prilikom logovanja, nad kojom *professor* korisnik nema nikakve privilegije. Zato će se preći u *elfak* bazu da bi se *professor* uspešno logovao.

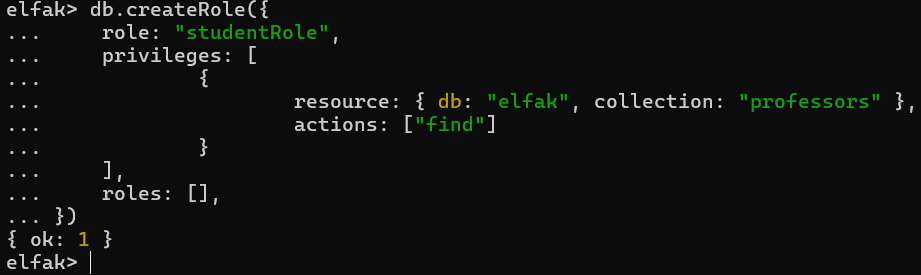
  
Slika 3.11. Prijava korisnika *professor* na sistem

Na slici 3.12 je prikazana demonstracija uloga koje ima *professor*. Prvo će pročitati podatke iz kolekcije *students* koja se nalazi u *elfak* bazi, nakon čega će uneti podatke u tu istu kolekciju. Kreiraće i novu kolekciju *professors* u koju će dodati podatke o jednom profesoru fakulteta. Ovim su demonstrirane privilegije koje je *professor* dobio od posedovanja *readWrite* uloge nad *elfak* bazom podataka.

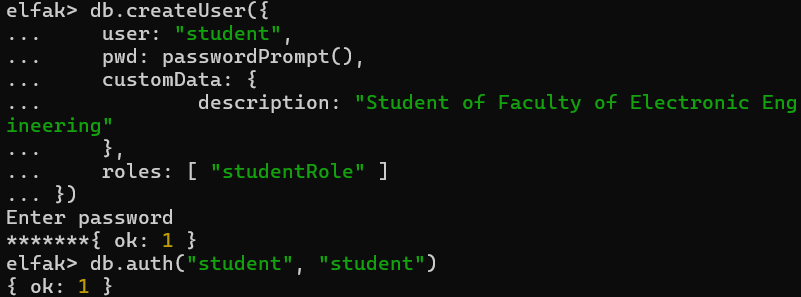
  
Slika 3.12. Demonstracija privilegija profesora od *readWrite* role

Posedovanjem *dbOwner* uloge nad *elfak* bazom podataka, profesor ima mogućnost da kreira nove korisnike i upravlja postojećim korisnicima u ovoj bazi. Iz tog razloga će profesor kreirati korisnika *student* koji će imati privilegije za čitanje podataka iz kolekcije *professors*, kako bi mogao da vidi koji su sve profesori angažovani na fakultetu. Nad kolekcijom *students* neće imati nikakve privilegije, jer bi se time narušila privatnost studenata koji studiraju na fakultetu.

Za ove potrebe je neophodno napraviti korisnički definisanu ulogu kojom će se isključivo definisati privilegije potrebne za čitanje podataka i to isključivo nad *professors* kolekcijom. Korisnik *professor* posedovanjem *dbOwner* uloge ima mogućnost da kreira korisnički definisane uloge nad *elfak* bazom, što je prikazano na slici 3.13.

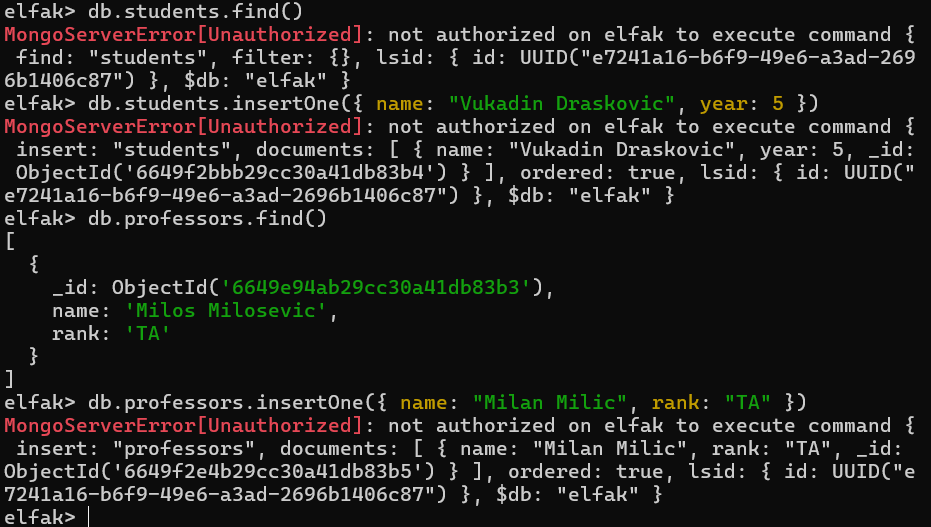
  
Slika 3.13. Kreiranje korisnički definisane uloge *studentRole*

Sada, kada je kreirana uloga koja sadrži privilegije koje treba da ima *student*, profesor će kreirati nalog studenta u *elfak* bazi podataka. Kreiranje *student* naloga i logovanje studenta na sistem je prikazano na slici 3.14.

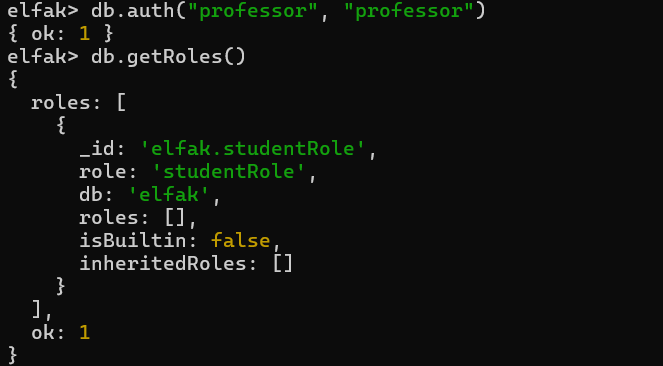
  
Slika 3.14. Kreiranje *student* korisnika u *elfak* bazi podataka i njegovo logovanje

Na slici 3.15 je prikazano da korisnik *student* nema mogućnost čitanja podataka iz *students* kolekcije, kao ni upisa podataka u istu. Što se tiče *professors* kolekcije, student ima pravo čitanja podataka iz nje, ali ne i pravo upisa u istu kolekciju. Ovim je demonstrirano korišćenje korisnički definisanih rola i njihovo dodeljivanje određenim korisnicima u MongoDB-u.

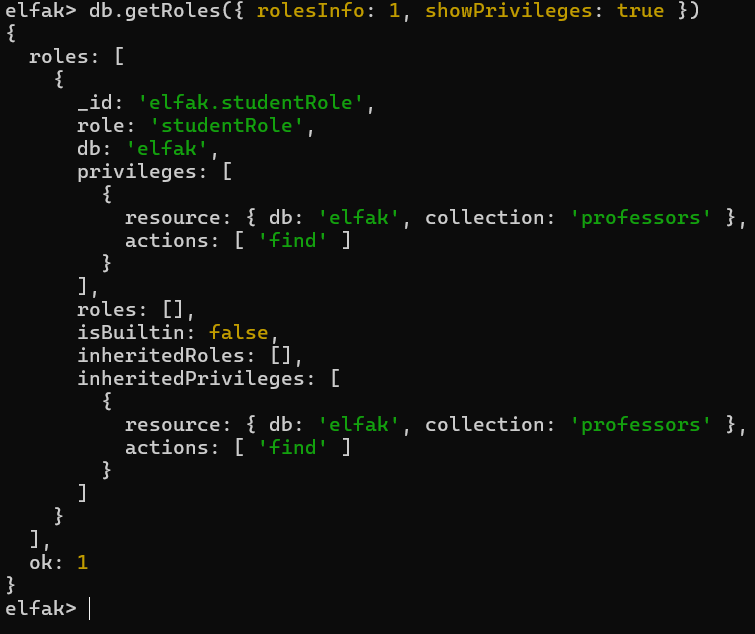
Prikaz uloga koje se nalaze u *elfak* bazi podataka i koje se mogu dodeliti korisnicima u istoj

  
Slika 3.15. Demonstracija akcija koje može izvršiti korisnik *student* u *elfak* bazi podataka

je moguće pomoću *getRoles* komande. Da bi se ova komanda izvršila, potrebno je da se na sistem bude autentifikovan nalogom koji ima mogućnost administracije *elfak* baze podataka, što je nalog *professor*. Izvršenje *getRoles* komande bez arguemata će prikazati samo korisnički definisane uloge u bazi podataka, bez privilegija koje prikazane uloge imaju (Slika 3.16).

  
Slika 3.16. Prikaz korisnički definisanih rola u *elfak* bazi podataka

Ukoliko želimo da se prikažu i privilegije koje imaju prikazane uloge, neophodno je proslediti komandi argument *showPrivileges*, što je prikazano na slici 3.17.

  
Slika 3.17. Prikaz privilegija korisnički definisanih uloga u *elfak* bazi podataka

# Zaključak

U radu su prvo predstavljeni osnovni pojmovi sigurnosti baza podataka i mehanizmi kojima se ona sprovodi, da bi se kasnije ta sigurnost demonstrirala na primeru MongoDB baze podataka. Za razliku od relacionih baza podataka, koje se najvećim delom zasnivaju na DAC mehanizmu kontrole pristupa, MongoDB koristi kontrolu pristupa baziranu na ulogama, tako da se može reći da se administracija MongoDB-a i relacionih baza podataka razlikuje u samom konceptu. Pored ove razlike, postoji i razlika u sintaksi naredbi za administraciju, što može predstavljati dodatni problem administratorima relacionih baza podataka koji su dobili zadatak da budu administratori MongoDB-a i upravljaju njime.

# Literatura

[1] Mark Stamp, *Information Security - Principles and Practice*, WILEY

[2] Discretionary Access Control vs. Mandatory Access Control: What's the Difference?, dostupno na: <https://www.zuar.com/blog/discretionary-access-control-vs-mandatory-access-control/> (pristupljeno 1. maja 2024.)

[3] What Is Role-Based Access Control & Why Your Portal Needs It, dostupno na: <https://www.zuar.com/blog/what-is-role-based-access-control-for-portals/> (pristupljeno 1. maja 2024.)

[4] MongoDB, dostupno na: <https://www.mongodb.com/> (pristupljeno 7. maja 2024.)

[5] Authentication, dostupno na: <https://www.mongodb.com/docs/manual/core/authentication/> (pristupljeno 7. maja 2024.)

[6] Use SCRAM to Authenticate Clients, dostupno na: <https://www.mongodb.com/docs/manual/tutorial/configure-scram-client-authentication/> (pristupljeno 10. maja 2024.)

[7] Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms, dostupno na: <https://datatracker.ietf.org/doc/html/rfc5802> (pristupljeno 10. maja 2024.)

[8] Localhost Exception, dostupno na: [https://www.mongodb.com/docs/manual/core/localhost-exception/#:~:text=The%20localhost%20exception%20allows%20you,privileges%20to%20create%20other%20users](https://www.mongodb.com/docs/manual/core/localhost-exception/%23:~:text=The%20localhost%20exception%20allows%20you,privileges%20to%20create%20other%20users) (pristupljeno 11. maja 2024.)

[9] db.auth(), dostupno na: <https://www.mongodb.com/docs/manual/reference/method/db.auth/> (pristupljeno 11. maja 2024.)

[10] db.createUser(), dostupno na: <https://www.mongodb.com/docs/manual/reference/method/db.createUser/> (pristupljeno 12. maja 2024.)

[11] Built-In Roles, dostupno na: <https://www.mongodb.com/docs/manual/reference/built-in-roles/> (pristupljeno 14. maja 2024.)

[12] User-Defined Roles, dostupno na: <https://www.mongodb.com/docs/manual/core/security-user-defined-roles/> (pristupljeno 15. maja 2024.)

[13] db.createRole(), dostupno na: [https://www.mongodb.com/docs/manual/reference/method/db.createRole/#mongodb-method-db.createRole](https://www.mongodb.com/docs/manual/reference/method/db.createRole/%23mongodb-method-db.createRole) (pristupljeno 16. maja 2024.)

[14] db.getRoles(), dostupno na: <https://www.mongodb.com/docs/manual/reference/method/db.getRoles/> (pristupljeno 17. maja 2024.)