SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mirta Štefanac

WEB SUSTAV ZA UPRAVLJANJE DOKUMENTIMA TEMELJEM POLUSTRUKTURIRANIH BAZA PODATAKA

DIPLOMSKI RAD

Varaždin, 2016.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mirta Štefanac

Matični broj: 42584/13-R

Studij: Baze podataka i baze znanja

WEB SUSTAV ZA UPRAVLJANJE DOKUMENTIMA TEMELJEM POLUSTRUKTURIRANIH BAZA PODATAKA

DIPLOMSKI RAD

Mentor:

Doc.dr.sc. Markus Schatten

Varaždin, kolovoz 2016

Sažetak

U ovom radu biti će realizirano cjelovito web rješenje za poduzeće. Web rješenje obuhvaća oglednu stranicu tvrtke te aplikaciju za upravljanje dokumentima putem koje će se zaposlenicima omogućiti dodavanje, mijenjanje, brisanje i dijeljenje dokumenata. Aplikaciji će se moći pristupiti samo uz odgovarajuće korisničko ime i lozinku. Za realizaciju aplikacije biti će korištene polustrukturirane baze podataka temeljene na JSON formatu (MongoDB). U teorijskom dijelu rada biti će detaljno razrađene sve teme i tehnologije vezane uz izradu ove aplikacije.

Ključne riječi: polustrukturirane baze podataka, upravljanje dokumentima, mongodb, json

Sadržaj

1. Uvod 1
2. Polustrukturirani podaci 3
   - 2.1 Polustrukturirani model podataka 4
     - 2.1.1 Object Exchange Model (OEM) 5
   - 2.2 Upiti nad polustrukturiranim podacima 6
     - 2.2.1 Lorel 7
3. JSON 8
   - 3.1 JSON tipovi podataka 8
     - 3.1.1 Broj 8
     - 3.1.2 Tekst 9
     - 3.1.3 Logički tip podataka 9
     - 3.1.4 Polje 10
     - 3.1.5 Objekt 10
     - 3.1.6 Null vrijednosti 11
   - 3.2 JSON Shema 11
   - 3.3 JSONiq 12
   - 3.4 Usporedba JSON i XML formata 13
4. Sustav za upravljanje dokumentima 14
5. NoSQL baze podataka 16
   - 5.1 Ključ-vrijednost baze podataka 16
   - 5.2 Dokument baze podataka 18
     - 5.2.1 MongoDB 20
   - 5.3 Stupčane baze podataka 23
   - 5.4 Graf baze podataka 24
   - 5.5 Usporedba NoSQL i relacijskih baza podataka 25
6. Node.js 28
   - 6.1 Express.js 30
   - 6.2 Jade 31
7. Praktični rad 32
   - 7.1 Zahtjevi aplikacije 32
   - 7.2 Implementacija aplikacije 34
     - 7.2.1 Arhitektura aplikacije 34
     - 7.2.2 Podešavanje razvojnog okruženja 35

7.2.3 Baza podataka....36
7.2.4 Kreiranje prijave, registracije i provjere korisnika....38
7.2.5 Kreiranje strukture....46
7.2.6 Dodavanje dokumenta....50
7.2.7 Pretraživanje dokumenata....53
7.3 Moguće nadogradnje aplikacije....54
8. Zaključak....56
9. Literatura....57

1. Uvod

Veliki dio današnje poslovne interakcije, kao i poslovanja općenito, temelji se na dokumentima. Dokumente šaljemo i primamo od naših klijenata, papirnato ili elektronski, generiramo ih iz vlastitih sustava i aplikacija ili ih pak izrađujemo ručno. Problem nastaje ako ti podaci nisu strukturirani i pohranjeni na pravilan način jer onda njihova iskoristivost, mogućnost analize i korištenje u svrhu unapređenja poslovanja postaje vrlo mala. Ta količina dokumentacije, kao i informacije koje ti dokumenti sadrže, povlače niz izazova koji nas navode na razmišljanje o upravljanju dokumentacijom. Dokumentacija predstavlja temelj funkcioniranja svakog poslovnog sustava, stoga svaka organizacija mora definirati procese i dokumente kojima projektira organizaciju, analizira procese i provodi neophodne aktivnosti. Upravljanje dokumentima osigurava protok informacija te pravodobnu i uspješnu komunikaciju.

Rastom organizacije neizbježno dolazi do rasta količine podataka, dokumenata i papira. Vrlo brzo zaposlenici se nađu okruženi papirima i registratorima u kojima se vrlo teško snaći, gubi se previše vremena na traženje dokumenata, često dolazi do zabune oko toga koja je posljednja verzija nekog dokumenta i sl. Nedugo nakon pojave računala, počeli su se razvijati i prvi sustavi za upravljanje dokumentima (eng. *document management systems*). Sustav za upravljanje dokumentima odnosi se na korištenje računalnog sustava i softvera za pohranu, upravljanje i praćenje elektroničkih dokumenata. Takvi sustavi su odgovor na sve probleme koje dolaze s dokumentacijom u organizaciji. Oni služe kao alat za sigurnu i urednu pohranu svih digitalnih dokumenata u organizaciji. Ali oni su i više od toga: omogućuju jednostavno pretraživanje, zajednički rad, određivanje prava pristupa, sigurnost, praćenje verzije dokumenata, kontrolu, i slično.

Takvi sustavi tradicionalno su baziran na relacijskim bazama podataka, no zadnjih godina sve popularnije postaju NoSQL (eng. *Not Only SQL*) baze podataka, koje zapravo i više odgovaraju ovakvim sustavima zato što se temelje na polustrukturiranim i nestrukturiranim podacima, a takvih je najviše u organizacijama. Također NoSQL baze omogućuju pohranjivanje cijelih dokumenata. Još jedna, možda i najvažnija razlika između SQL i NoSQL baza podataka je ta što se NoSQL baze ne moraju imati definiranu shemu, dok SQL (eng. *Structured Query Language*) baze moraju imati točno definirani model podataka.

U ovom radu prikazat ću jedan takav sustav za upravljanje dokumentima. Bit će temeljen na polustrukturiranim podacima uz korištenje jedne od najpopularnije NoSQL baze

podataka zvane MongoDB te će biti baziran na webu. Također, biti će korištena Node.js platforma za izradu serverske strane aplikacije. Klijentska strana aplikacije bit će izrađena koristeći klasične web tehnologije – HTML, CSS i JavaScript.

Više o polustrukturiranim podacima, JSON formatu, NoSQL bazama podataka i sustavu za upravljanje dokumentima slijedi u nadolazećim poglavljima.

2. Polustrukturirani podaci

Tradicionalne relacijske baze podataka imaju dugotrajnu poziciju u većini organizacija i to iz jako dobrih razloga. Kao prvo, relacijski model je star preko 40 godina (Edgar Frank Codd, 1970., *"A Relational Model of Data for Large Shared Data Banks"*), relacijske baze podupiru aplikacije koje se koriste za trenutne poslovne potrebe, postoji veliki broj poduzeća koja izgrađuju i održavaju takve sustave, podaci su uredno strukturirani u tablice koje se mogu povezivati itd. Međutim, dizajn klasičnih relacijskih baza podataka posljednjih desetak godina nije se bitno mijenjao i danas sve teže može pratiti zahtjeve velikih količina podataka (eng. *big data*)[1] s kojima se moraju nositi sadašnje i buduće aplikacije. Također, porastom korisnika Interneta sve više i više podataka se razmjenjuje putem mreže, a ti podaci nisu baš strukturirani te su heterogeni i često nepotpuni. Tu se počela javljati potreba za polustrukturiranim modelom podataka. Osvrnimo se prije svega na razliku između strukturiranih, polustrukturiranih i nestrukturiranih podataka.

Dakle vrste podataka prema strukturiranosti možemo podijeliti na strukturirane, polustrukturirane i nestrukturirani.

Kada pričamo o **strukturiranim** podacima najbitnije je istaknuti da su oni su bazirani na shemi. To znači da je struktura podataka stroga i poznata unaprijed. Tek kad je shema definirana onda se mogu unositi podaci koji odgovaraju toj shemi. Najjednostavnije objašnjeno, strukturirani podaci su svi oni koji se mogu smjestiti u SQL bazu podataka u tablicu s recima i stupcima. Takvi podaci su najzastupljeniji u razvoju aplikacija i pružaju najjednostavniji način obrade podataka. Ali strukturirani podaci predstavljaju vrlo mali postotak svih informatičkih podataka, od 5% do 10%. (Seth Grimes, *Unstructured data and the 80 percent rule*, 2008)

Kod **polustrukturiranih** podataka (eng. *semi-structured data*) slučaj je nešto različit. Oni nemaju definiranu rigoroznu shemu, recimo da je kod podataka ove vrste shema opcionalna. U nekim oblicima polustrukturiranih podataka uopće nema sheme, a kod nekih postoji, ali stavlja labava ograničenja nad podatke. Te podatke karakterizira nepravilna i implicitna struktura, fleksibilnost...

[1] *Big data* je relativno novi pojam koji se odnosi na skupove podataka koji su toliko veliki i kompleksni da tradicionalne aplikacije za obradu podataka nisu adekvatne.

```
{file: {name: "file1.txt"}, {dateCreated: "10-12-2015"}, {dateModified: "02-05-2016"}}
{file: {name: "image1.jpg"}, {dateCreated: "01-07-2016"}}
{file: {name: "document1.docx"}, {dateCreated: "02-09.2014"}, {size: "1581"}}
```

Slika 1 Primjer polustrukturiranih podataka nepravilne strukture (JSON format)

Primjeri polustrukturiranih podataka su XML(eng. *eXtensible Markup Language*), JSON (eng. *Javascript Object Notation*), NoSQL(eng. *Not Only SQL*) baze podataka. Slično kao i kod strukturiranih, na polustrukturirane otpada samo 5 do 10% informatičkih podataka. Iz toga se već može zaključiti da najveći postotak (oko 80%) pripada nestrukturiranim podacima. (Seth Grimes, *Unstructured data and the 80 percent rule*, 2008)

Nestrukturirani podaci su svugdje. Zapravo većina osoba i organizacija svoje živote temelje oko nestrukturiranih podataka. To uključuje tekstualni i multimedijski sadržaj. Na primjer: e-mail poruke, tekstualni dokumenti, fotografije, video datoteke, prezentacije, audio datoteke, web stranice itd. Iako ovo datoteke mogu imati unutarnju strukturu i dalje se smatraju nestrukturiranima zato što podaci koje sadržavaju ne stanu uredno u bazu podataka.

Polustrukturirani podaci su postali vrlo važna tema proučavanja iz više razloga. Kao prvo, voljeli bi tretirati izvore podataka kao što je Web kao bazu podataka, međutim Web ne može biti ograničen shemom. Drugo, bilo bi poželjno imati iznimno fleksibilni format za razmjenu podataka između dviju baza podataka koje se razlikuju po tipu. Treće, čak i kad se radi o strukturiranim podacima, bilo bi korisno vidjeti ih kao polustrukturirane u svrhu pregledavanja (eng. *browsing*).

2.1 Polustrukturirani model podataka

Definirajmo prvo pojam modela podataka:

Model podataka je apstraktni model čija je svrha opisati kako se podaci mogu učinkovito koristiti i predstavljati.Izraz model podataka se može gledati na dva načina. Prvi je kada govorimo o teoriji modela podataka. To je formalni opis kako se podaci mogu koristiti i strukturirati. Drugo je kada govorimo o instanci modela podataka, tj. kako je primijenjena pojedina teorija modela podataka u svrhu stvaranja prikladne instance za neku aplikaciju. Osnovne elemente svakog modela podatka čine struktura, operacije i ograničenja.[2]

Prednosti polustrukturiranog modela podataka uključuju mogućnost prikaza podataka koje nije lako ograničiti shemom (često se zove "*schema-less*") ili samoopisujući model

[2] Agile Data, *Data Modeling* (Izvor: http://www.agiledata.org/essays/dataModeling101.html)

podataka (eng. *self describing*), prednost je i njegova fleksibilnost u pogledu prijenosa podataka, sposobnost prikaza strukturiranih podataka kao i sposobnost mijenjanja strukture tijekom određenog vremena. Prema [13].

Polustrukturirani model obično ima sljedeće karakteristike:

- Podaci su modelirani u obliku stabla ili grafova.
- Podaci mogu postojati bez sheme (iako ona može postojati)

2.1.1 Object Exchange Model (OEM)

Najpopularniji model polustrukturiranih podataka je OEM model (*Object Exchange Model*). OEM je bio eksplicitno definiran za TSIMMIS (*The Stanford-IBM Manager of Multiple Information Sources*)[3], sustav za integraciju heterogenih izvora podataka.

OEM je model na kojeg se može gledati kao na označeni usmjereni graf (eng. *direct graph*), gdje su podaci prikazani kao kolekcije objekata. Svaki objekt je definiran kao četvorka koja mora sadržavati oznaku (eng. *label*), identifikator (oid, eng. *object identifier*), tip i vrijednost. Objekti, prema tipu, mogu biti jednostavni ili složeni. Vrijednost jednostavnog objekta može biti broj, znakovni niz, slika, zvuk, itd. Vrijednost složenog objekta je skup od nula ili više jednostavnih objekata. Dakle, podaci su prikazani grafom u kojem su čvorovi objekti, bridovi sadrže imena atributa, a vrijednosti se nalaze u listovima grafa. Graf također mora sadržavati korijenski objekt (eng. *root object*) koji sadržava sve ostale objekte. Prema[1], str. 18.

Formalno, polustrukturirani graf s podacima je četvorka:

$$G = (V, E, r, v)$$

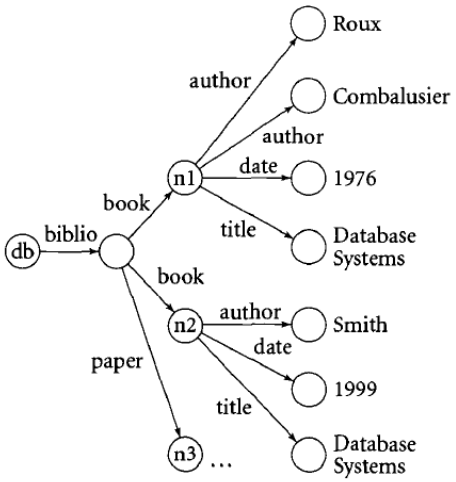
gdje je V (eng. *vertex*, vrh) konačan set čvorova koji je rastavljen na složene (V_c) i jednostavne (V_c) čvorove $V = V_c \cup V_a$, gdje je $E \subseteq V \times \Sigma \times V$ skup čije elemente nazivamo označenim bridocima, gdje je $r \in V$ korijen (eng. *root*) i gdje $v : V_a \rightarrow D$ funkcija koja pridružuje vrijednosti jednostavnim objektima (gdje je D je skup jednostavnih vrijednosti). Prema [15].

[3] TSIMMIS - http://infolab.stanford.edu/tsimmis/

2.2 Upiti nad polustrukturiranim podacima

U ovom poglavlju biti će opisan jednostavni, osnovni jezik za upite nad polustrukturiranim podacima.

Jedna od glavnih značajki upitnih jezika za polustrukturirane podatke je sposobnost dohvaćanja proizvoljnih dubina u podatkovnom grafu. To omogućuje korištenje *path expressiona*. *Path expression* je zapravo jedostavan upit nad podatkovnim grafom čiji je rezultat skup čvorova. Na primjer, rezultat upita `biblio.book` na Slika 2. je skup čvorova `{n1, n2}`. Rezultat izraza `biblio.book.author` je skup čvorova koji je zapravo skup znakova `{'Roux', 'Combalusier', 'Smith'}`. Prema [1], str 55.



Slika 2 Baza prikazana u obliku grafa [1]

Kako ne bi morali specificirati cijelu putanju, možemo je specificirati prema nekim svojstvima. Svojstvo može biti svojstvo putanje ili svojstvu individualnog brida. Za definiranje tih svojstava koriste se regularni izrazi (eng. *regular expression, regex*). Regularni izraz je niz znakova koji opisuje druge nizove znakova (engl. *string*), u skladu s određenim sintaksnim pravilima. Prvenstvena svrha regularnog izraza je opisivanje uzorka za pretraživanje nizova znakova. Na primjer rezultat upita `biblio.(book|paper).author` biti će čvor `{n1}` ili `{n3}`. Prema [1], str 55.

Iako su ovakvi izrazi osnovna značajka upitnih jezika nad polustrukturiranim podacima, sposobnosti su im ograničene. Na primjer, mogu vratiti samo podskupove čvorova, ne mogu konstruirati nove čvorove, ne mogu raditi spajanja, ne mogu testirati vrijednosti... *Path expressions* čine centralnu komponentu upitnih jezika. Prema [1], str 56.

2.2.1 Lorel

Lore (*Lightweight Object Repository*) je sustav za upravljanje bazama s polustrukturiranim podacima, koji se temelji na OEM podatkovnom modelu, a Lorel (*Lore Language*) je njen upitni jezik. Lorel ima poznatu *select-form-where* sintaksu i baziran je na OQL (*Object Query Language*) jeziku sa određenim modifikacijama i ekstenzijama koje su korisne kada se rade upiti nad polustrukturiranim podacima. Podaci smješteni u Loreu nisu ograničeni shemom, te mogu biti nepravilni ili nepotpuni. U principu, Lore pokušava iskoristiri strukturu tamo gdje postoji, ali ako je nema, napravljena je tako da se može nositi i s nepravilnim podacima. Prema [7].

Prikažimo osnovnu sintaksu jezika Lorel kroz sljedeći primjer:

```
% Query  q1
select   author: X
from     biblio.book.author X
```

U ovom primjeru varijabla X se veže za svaki čvor specificiran u izrazu `biblio.book.author`. Efekt upita je formirati novi čvor i spojiti ga s bridovima koji imaju oznaku `author`. Rezultat ovog upita je novi podatkovni graf koji čini skup svih autora sa Slike 2. Prema [1], str 59.

```
{author: "Roux", author: "Combalusier", author: "Smith"}
```

3. JSON

JSON (*JavaScript Object Notation*) je tekstualni format za prijenos strukture podataka koji je čitljiv i razumljiv ljudima i strojevima. Stvaranje JSON-a pripisuje se Douglasu Crockfordu. Iako ga nije prvi primijetio, dao mu je ime i formalnu gramatiku te je prihvaćen unutar RFC 4627 i ECMA-404 standarda. ECMA standard opisuje dozvoljenu sintaksu, a RFC pruža i semantiku i odgovore na pitanja sigurnosti. Službena vrsta medijske datoteke (eng. *Internet Media Type*) za JSON je *application/json*, a ekstenzija za JSON datoteku označava se s *.json*.

3.1 JSON tipovi podataka

Osnovni tipovi podataka JSON formata su sljedeći [18]:

- Broj (*Number*)
- Tekst (*String*)
- Logički tip podataka (*Boolean*)
- Polje(*Array*)
- Objekt (*Object*)
- Null vrijednost (*Null*)

3.1.1 Broj

JSON podržava i cijele i decimalne brojeve, s time da su decimalni odvojeni točkom. Zapravo je vrlo sličan kao broj u C ili Java programskom jeziku, osim što se ne koriste oktalni i heksadecimalni formati. Sljedeća tablica prikazuje tipove brojeva koji se koriste u JSON-u:

Tip	Opis
Cijeli broj (*Integer*)	Znamenke 0 – 9, pozitivne ili negativne
Razlomak (*Fraction*)	Razloci u obliku .3, .9
Eksponent (*Exponent*)	Eksponenti u obliku e, e+, e-, E, E+, E-

Tablica 1Tipovi brojevi u JSON formatu [18]

Sintaksa i primjer:

```
var jsonObject = {string : value, ... }
var object = {years: 17}
```

3.1.2 Tekst

Niz od nula ili više Unicode znakova. Stringovi su razdvojeni s dvostrukim navodnicima i podržavaju "*backslash*". Sljedeća tablica prikazuje tipove stringova koji se koriste u JSON-u:

Tip	Opis
"	double quotation
****	reverse solidus
/	solidus
b	backspace
f	form feed
n	new line
r	carriage return
t	horizontal tab
u	four hexadecimal digits

Tablica 2 Tipovi stringova u JSON formatu [5]

Sintaksa i primjer:

```
var jsonObject = { string : "string value", ... }
var object = { name: „Marko“ }
```

3.1.3 Logički tip podataka

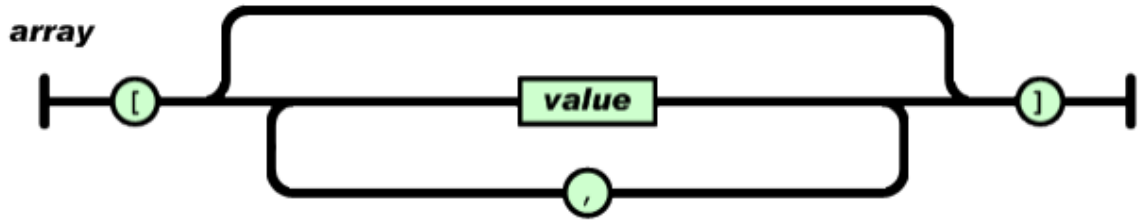
Odnosi se na istinite ili lažne vrijednost (true / false).

Sintaksa i primjer:

```
var jsonObject = { string : true/false, ... }
var object = { flag: true }
```

3.1.4 Polje

Sortirani niz vrijednosti koje mogu biti bilo kojeg tipa podataka. Polje može sadržavati i druga polja. Zatvoreni su unutar uglatih zagrada ' […] ', vrijednosti se odvajaju zarezom, a indeksiranje polja može početi s nulom ili jedinicom. Prema [18].



Slika 3 JSON polje [5]

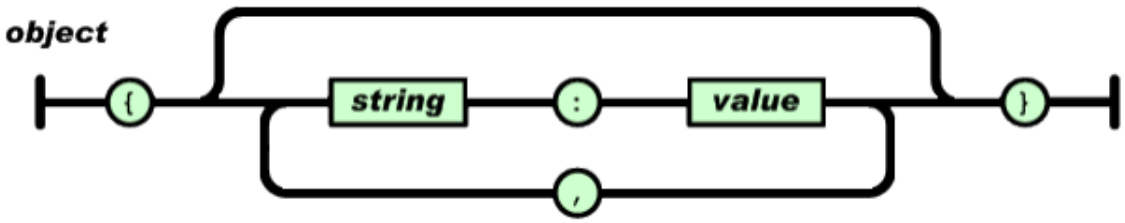
Sintaksa i primjer:

```
[ value, .......]

{
   "studenti": [
      { "ime":"Ana" , "godina_studija":"prva" },
      { " ime ":"Ivan" , " studija ":"peta" },
      { " ime ":"Josip" , " studija ":"treća" }
   ]
}
```

3.1.5 Objekt

Nesortirani niz oblika ključ/vrijednost (eng. *key/value*) parova. Nalaze se unutar vitičastih zagrada ' { … } ', nakon svakog ključa slijedi dvotočka ' : ', a ključ /vrijednost parovi su odvojeni zarezom ' , '. Ključevi također moraju biti nizovi znakova (*string*) i trebaju biti različiti. Prema [5].



Slika 4 JSON objekt [5]

Sintaksa i primjer:

```
{ string : value, ...}
```

```
{
   "id": "021801234",
   "name": "Marko",
   "years": 17
}
```

3.1.6 Null vrijednosti

Null vrijednost predstavlja praznu vrijednost.

Sintaksa i primjer:

```
null
var k = null;
```

3.2 JSON Shema

JSON shemom može se definirati struktura JSON podataka. Koristi se kako bi definirali dokumentaciju čitljivu ljudima i strojevima te služi za validaciju podataka. JSON shema bazirana je na konceptima XML sheme (XSD), ali se temelji na JSON-u. Prema [18].

Primjer JSON sheme:

```
{
   "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "Proizvod",
   "description": "Katalog proizvoda",
   "type": "object",

   "properties": {
      "id": {
         "type": "integer"
      },
      "name": {
         "description": "Ime proizvoda",
         "type": "string"
      },

      "price": {
         "type": "number",
         "minimum": 0,
         "exclusiveMinimum": true
      }
   },
   "required": ["id", "name", "price"]
}
```

Ovom shemom definirali smo da JSON dokument koji će se validirati prema ovoj shemi mora u sebi sadržavati objekt Proizvod koji obvezno mora imati jedinstveni identifikator, ime te cijenu. U primjeru su sadržane su neke bitne ključne riječi. Prvo nailazimo na ključnu riječ **$schema** što nas upućuje na činjenicu da je ova shema pisana prema *draft v4* specifikaciji. Ključne riječi **title** i **description** su prilično očite, njima dajemo naziv i opis sheme. Ključna riječ **type** definira prvo ograničenje nad ovom shemom, a to je da ona mora biti JSON objekt. Nad svojstvom `"id"` također je stavljeno ograničenje takvo da mora biti cijeli broj. Naziv proizvoda mora biti niz znakova, a cijena mora biti tipa broj, i ima ograničenje **minimum** što znači da minimalna cijena može biti 0. Ako **exclusiveMinimum** ima postavljenu vrijednost *true*, to znači da da minimum mora biti strogo veći od zadanog broja. U ovom primjeru to bi značilo da cijena mora biti strogo veća od nule. Ključnom riječi **required** označavamo koja svojstva su obvezna. Sljedećim JSON dokumentom možemo testirati, tj. validirati shemu: Prema [14]

```
[
   {
      "id": 6313,
      "name": "Acer prijenosno računalo",
      "price": 6500,
   }
]
```

3.3 JSONiq

JSONiq je upitni jezik specifično dizajniran za postavljanje upita nad JSON podatkovnom modelu. JSONiq je ekspresivni i visoko optimiziran jezik te je napisan po uzoru na XQuery. Bitno je napomenuti da je orijentiran na skupove (eng. *set-oriented*) - većina jezika je dizajnirana tako da može obrađivati jedan po jedan objekt dok je JSONiq dizajniran tako da može obrađivati setove (zapravo, sekvence) podatkovnih objekata. Također je deklarativan i funkcionalan jezik što znači da izrazi imaju glavno značenje prilikom obrade. Prema [6]

3.4 Usporedba JSON i XML formata

S obzirom da se JSON format pokazao kao idealno rješenje za prijenos podataka koji su organizirani u polja te jednostavnih varijabli, učestalo se susreće kao konkurencija, tj. alternativa XML-u. JSON je format koji polako zamjenjuje XML jer ima nekolicinu prednosti u odnosu na XML. JSON ne koristi oznake (eng. *tags*) pa stoga ima kraći kôd koji je lakši i brži za pisanje te razumljiviji za čitanje. Druga bitna razlika je što se u poruci s JSON zapisom može prenositi serijalizirani objekt. U jednom sustavu se objekt izradi, te se potom serijalizira. U drugom sustavu, koji komunicira s prvim sustavom, taj objekt u poruci može se deserijalizirati i nastaviti s daljnjim korištenjem. Upravo ta druga razlika je JSON učinila popularnim gdje su se prebrodila ograničenja XML-a. Treća prednost JSON-a u odnosu na XML je ta što se JSON prevodi (eng *parsing*) kroz standardnu JavaScript funkciju dok se XML prevodi kroz XML prevoditelj.

Primjer JSON zapisa:

```
{
   "marka": HP,
   "ime": "Pavillion",
   "cijena": 7000
}
```

Primjer XML zapisa:

```
<laptop>
   <marka> HP </marka>
   <ime> Pavillion </ime>
   <cijena> 7000 </cijena>
</laptop>
```

4. Sustav za upravljanje dokumentima

Strukturiranim informacijama - podacima - već se dugi niz godina uspješno upravlja upotrebom sustava za upravljanje bazama podataka. Međutim, kao što je već navedeno u poglavlju 1. Polustrukturirani podaci, oko 80% svih informacija u tvrtkama spada u tzv. nestrukturirane informacije. Takve informacije su fotografije, tekstualni dokumenti, audio zapisi, video zapisi, elektronička pošta, različita izvješća, Web sadržaj i sl. Takvi podaci nisu pogodni za pohranu u tradicionalnim relacijskim bazama podataka. Ovakve informacije često su pohranjene na različitim medijima i lokacijama. Dokumenti kreirani u aplikacijama za obradu teksta pohranjuju se na lokalne datotečne sustave ili mrežne/dijeljene datotečne sustave, elektronička pošta koristi se samo u sklopu specijaliziranih aplikacija, strukturirane informacije nalaze se unutar baza podataka - različitim grupama informacija upravlja se odvojeno. Ovakav pristup upravljanju informacijama povlači čitav niz poteškoća. Iz tog razloga razvijen je sustav za upravljanje dokumentima. Prema [20].

Sustav za upravljanje dokumentima (eng. *Document management system,* DMS), omogućava pojedincima i/ili grupama međusobno razmjenjivanje dokumenta. Na centralnom mjestu smješteni su svi važni dokumenti za poslovanje te dokumenti koji nastaju u poslovnom procesu. Takav sustav omogućava opisivanje, dodavanje i mijenjanje dokumenata, dodavanje i upravljanje mapama te korisnicima i korisničkim grupama koji tim dokumentima mogu upravljati. Osnovno svojstvo DMS-a podrazumijeva pretvorbu dokumenata iz papirnatog oblika u digitalni čime dobivamo digitalne datoteke na računalu.

Iako će borba između papirnatih i elektroničkih dokumenata trajati još neko vrijeme, a velike količine papira će se i dalje tiskati zato što je to dio uobičajenog poslovanja, uvođenje sustava za upravljanje dokumentima nudi niz prednosti. Kao prvo pojednostavljuje se skladištenje dokumenata te se smanjuje vrijeme traženja. Pružaju bolju organizaciju. Sljedeća bitna stavka je sigurnost. Sustavi za upravljanje dokumentima omogućuje čuvanje povjerljivih poslovnih informacija. Također, mogu se integrirati s postojećim poslovnim sustavom. Svako poslovanje ima niz procesa koji se mogu optimizirati, poput slanja ulaznih računa koji se od ulaza u poduzeće šalju na odobravanje. Ako se taj proces automatizira, skenirani računi mogu elektronski i automatski biti poslani osobama koji trebaju odobriti te račune. Tako se radi integracija dokumenata i poslovnih procesa. Uz sustav za upravljanje dokumenata, korisnici mogu dijeliti dokumente i surađivati na dokumentima s drugima. Vlasnik dokumenta može kontrolirati s kim surađuje na dokumentu, a isti se onda može slati i

drugima, izvan organizacije. Uz funkcionalnosti tijeka revizije, lako se može i vidjeti tko je kada gledao i uređivao određene dokumente, što je kritično u pronalasku grešaka i neučinkovitosti. Prema [21].

Vrste sustava za upravljanje dokumentima možemo podijeliti u nekoliko kategorija:

- Klijent - server sustavi – ovakav tip sustava ponekad može biti brži i snažniji od ostalih tipova sustava za upravljanje dokumentima. Također, jednostavnije ga je integrirati s postojećim sustavima u organizaciji.
- Web bazirani sustavi – predost ovih sustava je činjenica da klijentu nije potrebno dostavljati sofver niti postoji ikakva instalacija paketa. Sve što klijentu treba za pristup sustavu je internetska veza i web preglednik
- Upravljanje dokumentima u oblaku (eng. *cloud*) - ili *"hosted"* sustav za upravljanje dokumentima. Omogućuje klijentima korištenje softvera koji se izvršava na serveru i dozvoljava im da pristup preko poveznice (eng. *link*) u pregledniku. Slični su web baziranim sustavima, osim što su u "cloud" slučaju korisnički podaci pohranjeni na poslužiteljske servere, a ne lokalno kod korisnika na računalu. Nije potrebna instalacija softvera kod ovog pristupa, a najčešće se plača mjesečna pretplata za korištenje usluge.

5. NoSQL baze podataka

NoSQL (eng. *Not Only SQL*) baze podataka obuhvaćaju široku paletu različitih tehnologija koje su razvijane kako bi odgovorile na zahtjeve modernih aplikacija. Razlog porasta njihove popularnosti leži u nedostacima otkrivenim tijekom rada s relacijskim bazama podataka. Relacijske baze podataka su, dakako, još uvijek dominantne na tržištu, no sve se više korisnika okreće upotrebi upravo nekih od NoSQL baza podataka.

NoSQL je pristup upravljanju podacima i dizajnu baze podataka koji se može koristiti na velikim skupovima distribuiranih podataka. NoSQL nastoji riješiti pitanje skalabilnosti i pitanja performansi kada pričamo o velikim količinama podataka (eng. *Big Data*). Kao glavne prednosti NoSQL baza podataka ističu se dostupnost (eng. *open-source*[4]), distribuiranost[5], visoke performanse, tolerancija na greške, i kao najvažnija, skalabilnost[6]. NoSQL može biti posebno koristan kada je potrebno pristupiti i analizirati velike količine nestrukturiranih podataka koji su skladišteni na udaljenim lokacijama na više virtualnih servera u oblaku (eng. *cloud*). [10]

NoSQL baze podataka možemo podijeliti na četiri tipa, s obzirom na to koji model pohranjivanja podataka koriste:

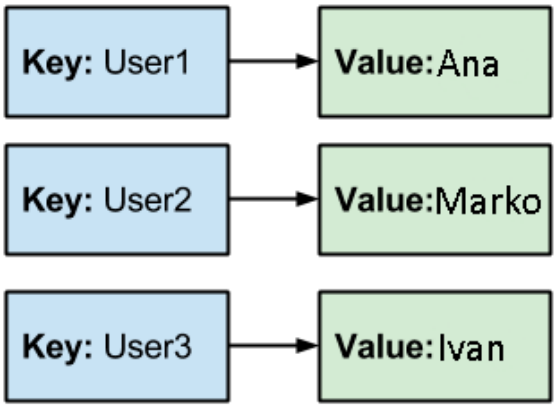
1. Ključ-vrijednost baze podataka (*Key-value stores*)
2. Dokument baze podataka (*Document databases*)
3. Stupčane baze podataka (*Wide-column stores*)
4. Graf baze podataka (*Graph stores*)

5.1 Ključ-vrijednost baze podataka

Ključ-vrijednost baze podataka su najjednostavnije NoSQL baze podataka te pružaju visoku razinu fleksibilnosti i skalabilnosti. Baze podataka bazirane na ovom podatkovnom modelu ne sadržavaju shemu niti ključ-vrijednost (sastoje od indeksiranog ključa i vrijednosti).

[4] Open-source – softver čiji je izvorni kod dostupan unutar *open source* licence svim korisnicima koji mogu mijenjati, prepravljati i poboljšavati njegov sadržaj. To znači da uz *open source* programe dolazi i čitav izvorni kod u nekom programskom jeziku, pa se može i mijenjati sam program.
[5] Distribuiranost - distribuirane baze podataka se mogu nalaziti na različitim serverima
[6] Skalabilnost - mogućnost sustava da se nosi s rastućom količinom podataka

Ključ-vrijednost baze podataka su pogodne za aplikacije koje imaju česta, ali mala čitanja i pisanja u bazu, te za one koje imaju jednostavne podatkovne modele. Vrijednosti pohranjene u ključ-vrijednost baze podataka mogu biti jednostavne skalarne vrijednosti kao što su cijeli brojevi ili logičke vrijednosti, ali mogu biti i strukturirani tipovi podataka kao što su liste ili JSON strukture. Ključ-vrijednost baze generalno imaju jednostavne upite koji omogućavaju pregledavanje vrijednosti prema njenom ključu. Neke takve baze podržavaju značajku pretraživanja koje pružaju nešto više fleksibilnosti. Sve u svemu, ključ-vrijednost baze podataka nemaju tako dobro razvijenu mogućnost pretraživanja kao što to imaju dokument, stupčane i graf baze podataka. Prema [22]



Slika 5 Prikaz ključ-vrijednost baze podataka

Ovaj tip baza koristi se u aplikacijama za [22]:

- predmemoriranje (eng. *caching*) podataka iz relacijskih baza podataka radi poboljšanja performansi
- praćenje prolaznih atributa u web aplikacijama kao što je košarica (eng. *shopping cart*)
- pohranjivanje konfiguracije i korisničkih podataka za mobilne aplikacije
- pohranjivanje velikih objekata kao što su slike i audio zapisi

Primjeri ovakvih baza podataka: Cassandra. DyanmoDB, Azure Table Storage (ATS), Riak, BerkeleyDB,...

5.2 Dokument baze podataka

Dokument baze podataka su dizajnirane za pohranu, dohvat i upravljanje polustrukturiranim podacima te su vjerojatno najpopularniji tip NoSQL baza zbog njene fleksibilnosti, performansi i jednostavnosti korištenja. Ovakav tip baze ne zahtjeva shemu podataka (ali može postojati ukoliko je potrebno) te se u nju mogu spremiti podaci bez da baza ima ikakvo saznanje o njihovoj strukturi, tipu ili značenju. Ako aplikacija zahtjeva mogućnost pohrane različitih atributa zajedno s velikim količinama podataka, onda su ovakve baze podataka dobra opcija. Na primjer, kako bi prikazao proizvode u relacijskoj bazi podataka, programer može koristiti tablicu s zajedničkim atributima i dodatne tablice za svaki podtip proizvoda kako bi pohranio atribute samo u podtipu proizvoda. Dokument baze podatka se jednostavno mogu nositi s ovakvom situacijom. U ovom tipu baze pohranjuju se sve informacije za dani objekt u jednu instancu baze podataka, te svaki pohranjeni objekt može biti različit od ostalih. Ovo čini preslikavanje (eng. *mapping*) objekata u bazu jednostavnim zadatkom. Ova značajka čini dokument baze podataka privlačnim za programiranje web aplikacije koje su podložne čestim promjenama i gdje je brzina implementacije vrlo važna stavka.

Kao što relacijske baze podataka spremaju podatke u redove i stupce, dokument baze podataka spremaju podatke u dokumente (eng. *documents*). Dokument u dokument bazi predstavlja samo-opisujuću stablastu strukturu poput JSON, XML ili BSON (*Binary* JSON) strukture. Ova fleksibilnost može biti osobito korisna kod modeliranja nestrukturiranih i polimorfnih podataka. Dokumenti pružaju intuitivni i prirodan način modeliranja podataka koji je vrlo usklađen s objektno orijentiranim programiranjem - svaki dokument čini jedan objekt. Kao što je već rečeno, u dokument bazi podataka, pojam sheme je dinamičan: svaki dokument može sadržavati različita polja. Dokumenti sadrže jedno ili više polja, gdje svako polje sadrži upisanu vrijednost kao što je niz znakova (eng. *string*), datum (eng. *date*), binarna vrijednost (eng. *binary*) ili polje (eng. *array*). Dodavanjem u bazu, svakom dokumentu se automatski dodjeljuje indeks. To znači da se svaki dokument može pretraživati. Umjesto razmještanja zapisa preko različitih stupaca i tablica povezanih stranim ključem, svaki zapis i njegovi pridružujući podaci su pohranjeni zajedno u jedan dokument. To pojednostavljuje pristup podacima i u mnogim slučajevima eliminira potrebu za skupim JOIN operacijama i kompleksnim transakcijama koje bi u relacijskom svijetu trajale mjesecima. Prema [10].

Primjer dokumenta u JSON notaciji:

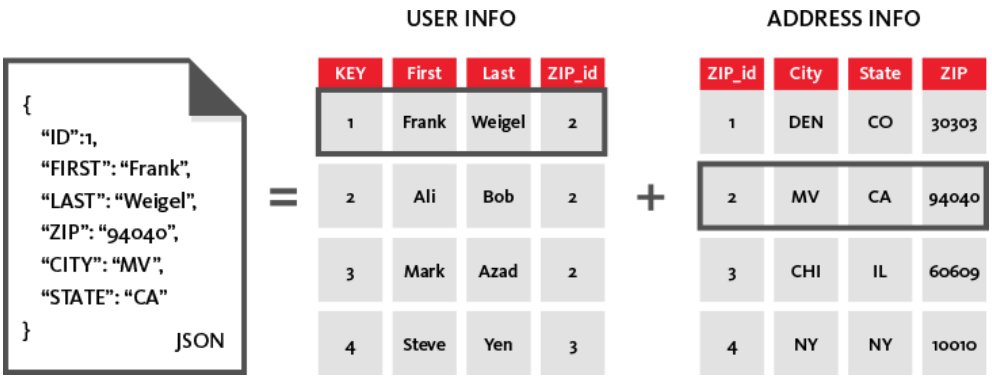
```
{
        "ime" : "Ana",
        "prezime" : "prva",
        "godiste" : 1991
}
```

Pohranjivanje podataka na ovaj način ima sljedeće prednosti [23]:

- dokumenti su neovisne jedinice što rezultira boljim performansama i čini distribuiranje podataka preko različitih poslužitelja (eng. *server*) puno jednostavnijim
- nestrukturirani podaci mogu jednostavno biti pohranjeni
- skupe migracije podataka se mogu izbjeći iz razloga što baza podataka ne mora imati definiranu shemu unaprijed
- dokument baze podataka generalno imaju vrlo moćne mehanizme pretraživanja i indeksiranja što čini izvođenje upita brzim i jednostavnim - snaga upitnog jezika dokument baza podataka je jedna od glavnih stavki koja ističe ovaj tip baze

Dokument baze podataka pogodne su za sljedeće [22]:

- *back-end* podršku web stranicama koje imaju česta čitanja i pisanja u bazu
- upravljanje vrstama podataka sa promjenjivim atributima kao što su npr. proizvodi
- aplikacije koje koriste JSON strukturu podataka
- aplikacije koje imaju koristi od denormalizacije[7] ugradnjom strukture unutar strukture



Slika 6 Prikaz dokument baza podataka u usporedbi s relacijskim
(Rassul Fazelat, 2015, *A Comprehensive Analysis - NoSQL vs RDBMS*)

Najpoznatije dokument baze podataka su MongoDB i CouchDB.

[7] Denormalizacija - spremanje istih podataka na više mjesta čime se povećava zalihost (redundancija)

5.2.1 MongoDB

MongoDB je snažna, fleksibilna i skalabilna baza podataka opće namjene. Osim kreiranja, čitanja, ažuriranja i brisanja podataka (CRUD) pruža sljedeće jedinstvene značajke: Prema [4] str 14.

- Indeksiranje - podržava sekundarne indekse što omogućuje razne brze upite i pruža jedinstvene, složene i geoprostorne mogućnosti indeksiranja
- Agregacija - MongoDB dopušta izradu kompleksnih agregacija iz jednostavnih dijelova i dozvoljava bazi da ih optimizira (eng. *aggregation pipeline*)
- Pohrana podataka - podržava protokol koji je jednostavan za korištenje a služi za pohranjivanje velikih datoteka i njihovih metapodataka

MongoDB sustav pruža relativno bogat skup funkcionalnosti s obzirom na njegovu jednostavnost. Sadržava alate koji nisu u potpunosti podržani ni u jednom relacijskom sustavu za upravljanje bazama podataka. Međutim, zbog očuvanja skalabilnosti MongoDB ne uključuje određene funkcionalnosti koje su karakteristične za relacijske sustave, kao što su spajanje (eng. *join*) i transkacije.

5.2.1.1. Osnovni koncepti MongoDB baze podataka:

Dokument(eng. *document*) je osnovna jedinica podataka u MongoDB-u. Dokument je ugrubo ekvivalent redu (eng. *row*) u relacijskom sustavu za upravljanje bazom podataka. Dokument se sastoji od jednog ili više polja (eng. *fields*), koje možemo usporediti s stupcima (eng. *columns*) u relacijskom svijetu.

Slično, kolekcija (eng. *collection*) se može shvatiti kao tablica s dinamičnom shemom. Dakle, kolekciju čini grupa dokumenata. Kolekciju se identificira prema njenom nazivu. Naziv može biti bilo koji UTF-8[8] niz znakova s nekoliko ograničenja – ne smije biti prazan (`" "`), ne smiju sadržavati znak \0 (`null`), ne smije počinjati s riječi *system.* jer je ona rezervirana za interne kolekcije, ne smije sadržavati znak `$`.

Osim što se dokumenti grupiraju u kolekcije, kolekcije se grupiraju u baze podataka. Jedna instanca MongoDB-a može posluživati više nezavisnih baza podataka, od koje svaka može imati svoje kolekcije. Svaki dokument ima specijalni ključ. `"_id"`, koji je jedinstven unutar kolekcije.

[8] UTF-8 (Unicode Transformation Format 8) je način zapisa kodnih točaka u standardu Unicode pomoću nizova 8-bitnih bajtova.

MongoDB dolazi s jednostavnom i snažnom JavaScript konzolom (eng. *shell*), koja je korisna za administriranje MongoDB instanci i za upravljanje podacima. Javascript Shell je potpuno opremljen JavaScript prevoditelj (eng. *interpreter*), sposoban za izvođenje proizvoljnih JavaScript programa.

5.2.1.2. CRUD operacije

U konzoli možemo koristiti četiri osnovne operacije *create*, *read*, *update* i *delete* (CRUD) kako bi manipulirali i pregledavali podatke. Prema [4].

Create

Funkcijom `insert` dodaje se dokument u kolekciju.

```
db.korisnici.insert({
  "ime" : "Ana",
  "prezime" : "Anic"
}
```

Ovom funkcijom u bazu `korisnici` dodan je dokument.

Read

Funkcijom `find` i `findOne` mogu se vršiti upiti nad kolekcijom. Ako želimo pronaći samo jedan dokument unutar kolekcije, koristi se funkcija `findOne`:

```
db.korisnici.findOne()
{
  "_id" : ObjectId("5037ee4a1084eb3ffeef7228"),
  "ime" : "Ana",
  "prezime" : "Anic"
}
```

Polje `"_id"` je dodano automatski, kao što je već objašnjeno ranije u ovom poglavlju.

Update

Ako je potrebno naknadno mijenjati vrijednosti unutar dokumenta koristi se funkcija `update`. Ona sadrži minimalno dva parametra: prvi je kriterij po kojem se traži dokument koji treba biti ažuriran, a drugi je vrijednost koja će se unijeti.

```
db.korisnici.update({ime : "Ana"}, "godiste" : 1991)
```

Delete

Funkcijom `remove` trajno se brišu dokumenti iz baze. Ako se pozove bez parametra, briše sve dokumente iz kolekcije. Ako se želi specificirati koji dokument je potrebno obrisati, dodaje se taj kriterij u funkciju.

```
db.korisnici.remove({ime : "Ana"})
```

MongoDB koristi verziju JSON formata koji se naziva BSON[9] (*Binary* JSON). Osnovna razlika između ova dva formata je u tome da se podaci smještaju u binarnom obliku u asocijativne nizove, umjesto u tekstualnom. Pored toga, BSON format uvodi nove tipove podataka koji nisu dio JSON standarda. Pored JSON standarda u kojem su podržani brojevi, stringovi, logičke varijable (eng. *boolean*), nizovi, objekti i *null* vrijednost, BSON format dodatno podržava: 32-bitne i 64-bitne cijele brojeve, te 64-bitne brojeve sa pokretnim zarezom (eng.*floating point*). Podržava *Object ID* kao jedinstveni 12-bajtni identifikator objekata. *Date*, odnosno datumi se smještaju kao milisekunde. Podržava regularne izraze (eng. *regular expression*), Kôd (eng. *code*) – u dokumentima se može čuvati JavaScript kôd, Binarne podatke (eng.*binary data*), nedefiniranu vrijednost (eng. *undefined*) – JavaScript pravi razliku između *null* i nedefinirane vrijednosti, nizovi (eng. *array*) koji podrazumijevaju skupove ili liste vrijednosti koje se predstavljaju u obliku nizova, ugnježdene dokumente (eng. *embedded document*) što podrazumijeva da dokumenti mogu imati ugnježdene pod-dokumente. Prema [4], str 16.

Tip podataka	Primjer korištenja
number	`{"x" : 3.14}`
date	`{"x" : new Date()}`
regular expression	`{"x" : /foobar/i}`
array	`{"x" : ["a", "b", "c"]}`
embedded document	`{"x" : {"foo" : "bar"}}`
object id	`{"x" : ObjectId()}`
binary data	/
code	`{"x" : function() { /* ... */ }}`

Tablica 3 Tipovi podataka korišteni u MongoDB bazi podataka [4]

[9]Binary JSON - http://bsonspec.org/

5.3 Stupčane baze podataka

Stupčane baze podataka su dizajnirane kako bi se mogli nositi s velikim količinama podataka (eng. *Big Data*)[10] , imaju dobre performanse čitanja i pisanja te visoku dostupnost. Ovaki sustavi za upravljanjem bazama podataka izvode se na klasterima/grozdovima (eng. *clusters*) više poslužitelja. Ako podaci koje imamo stanu unutar jednog poslužitelja, onda stupčane baze podataka nisu rješenje već je jednostavnije koristiti ključ-vrijednost ili dokument baze podataka. Prema [22].

Stupčane baze podataka koriste rijetku, distribuiranu, višedimenzionalnu sortiranu mapu za spremanje podataka. Svaki zapis može varirati u broju stupaca koji su spremljeni. Stupci mogu biti grupirani u obitelji stupaca radi boljeg pristupa ili mogu biti raspoređeni po raznim obiteljima. Podaci se dohvaćaju prema primarnom ključu za svaku obitelj stupaca. [10]

Stupčane baze korisne su za aplikacije: [22].

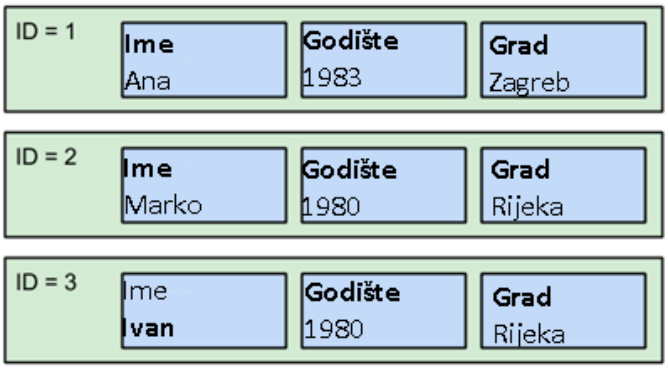
- koje zahtjevaju da se uvijek može pisati u bazu
- koje su geografski raspoređene na više podatkovnih centara
- koje mogu tolerirati kratkoročne nedosljednosti u replikama
- s dinamičnim poljima
- s potencijalno velikim količinama podataka (stotine terabajta podataka)

Nekoliko područja može iskoristiti ovu vrstu sposobnosti procesiranja takve količine podataka:

- Analitičari sigurnosti koji koriste mrežni promet
- "*Big Science*", kao što su bioinformatičari koji koriste genetske podatke
- Burza, analizirajući podatke o trgovini
- Društvene mreže

Najpoznatije stupčane baze podataka su Google-ov Bigtable i Facebook-ova Cassandra. Neke poznate aplikacije koje su bazirane na ovoj vrsti baza podataka su: Google Earth, Google Maps, Ebay, The New York Times, Comcast, Hulu,.. Ključ-vrijednost, dokument i stupčane baze podataka mogu se koristiti u različitim aplikacijama. Graf baze podataka su, međutim, prikladne samo za određene vrste problema. [22].

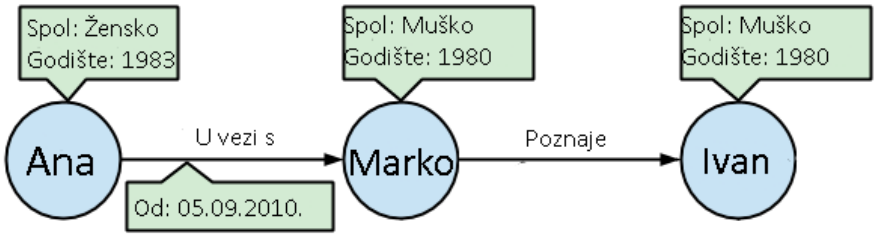
[10] *Big Data* - neformalno, skupovi podataka koji su preveliki da bi se efikasno mogli spremiti i analizirali relacijskim bazama podataka



Slika 7 Prikaz stupčanih baza podataka

5.4 Graf baze podataka

Za razliku od ostalih NoSQL baza podataka koje modeliraju podatke prema ključu i vrijednosti, graf baze podataka bazirane su na teoriji grafa te sadrže čvorove, veze i svojstva kako bi predstavili podatke. Čvorovi se mogu nazvati još i subjektima ili objektiva te predstavljaju entitete. Veze ili predikati predstavljaju bridovi koji mogu biti usmjereni. Podaci se modeliraju kao mreža ovisnosti između određenih elemenata. Iako model grafa može biti kontraproduktivan i zahtjeva određeno vrijeme da se nauči, može biti koristan za određenu klasu upita. Najbitnija stavka ovog modela je ta što olakšava oblikovanje i navigaciju odnosa između entiteta u aplikaciji. Prema [10].



Slika 8 Prikaz graf baza podataka

Upitni jezici kojima se pretražuju graf baze podataka zovu se Cypher (koristi se za Neo4j Graph bazu podataka) i Gremlin.

Primjeri korištenja graf baza:

- Upravljanje mrežom i IT infrastrukturom
- Upravljanje identitetom i pristupom
- Upravljanje poslovnim procesima
- Preporuka proizvoda i usluga
- Društvene mreže

– Mreža javnog prijevoza

Iz ovih primjera, očito je da kada postoji potreba za modeliranjem eksplicitnih odnosa/veza između entiteta, onda su graf baze podataka dobro rješenje.

5.5 Usporedba NoSQL i relacijskih baza podataka

SQL baze podataka su bili primarni mehanizmi pohrane podataka više od četiri desetljeća. Njihovo korištenje postiglo je vrhunac 1990-ih porastom web aplikacija i *open-source* opcija kao što su MySQL, PostgreSQL i SQLite. NoSQL baze podataka su tek nedavno dobile na popularnosti pojavom popularnih sustava kao što su MongoDB, CouchDB, Redis i Cassandra.

Kao prvo, treba razjasniti da NoSQL nije ništa bolji ili lošiji od SQL-a. Nekim aplikacijama bolje odgovara jedna vrsta, a nekima druga. To su dvije različite kategorije i možemo na njih gledati kao na alternative.

	SQL baze podataka	NoSQL baze podataka
Vrsta	Jedna vrsta s manjim varijacijama	Više različitih vrsta (*key-value, document, column, graph* baze)
Povijest razvoja	Razvijene 1970-ih kako bi se nosile s prvim valom aplikacija za pohranu podataka	Razvijene kasnih 2000-ih kako bi se nosile s ograničenjima SQL baza
Primjer	MySQL, Postgre, Microsoft SQL Server, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Primjer korištenja	Aplikacije gdje je integritet podataka esencijalan, centralizirane aplikacije (npr. ERP)	Decentralizirane aplikacije (npr. Web)
Model pohrane podataka	Podaci su spremljeni u tablice te su podijeljeni u redove i stupce. Povezani podaci nalaze se u drugim tablicama koje se spajaju kada se izvršavaju kompleksniji	Varira ovisno o tipu baze podataka. Ključ-vrijednost baze spremaju podatke slično kao i SQL baze, samo što imaju samo dva stupca (ključ i vrijednost).Dokument baze

	upiti.	spremaju podatke u JSON formatu,...
Tip podataka	Strukturiran tip podataka	Strukturiran, polustrukturiran, nestrukturiran tip podataka
Shema	Struktura i tip podataka mora se odrediti unaprijed. Kada želimo dodati novi tip podatka ili vezu, cijela baza se mora mijenjati.	Generalno dinamične, s ponekim pravilima validacije podataka. Aplikacije mogu dodavati nova polja u hodu, podaci koji nisu istog tipa mogu biti pohranjeni zajedno. Struktura se može jednostavno mijenjati po potrebi.
Skalabilnost	Vertikalna, što zahtjeva da poslužitelj mora biti spreman na pohranu velikih količina podataka.	Horizontalna, što znači da ako želi dodati kapacitet, administrator baze podataka može jednostavno dodati više poslužitelja. Baza podataka će se automatski proširiti preko poslužitelja po potrebi.
Razvojni model	Mješavina *open-source* (MySql, Postgre) i komercijalnih(Oracle, DB2) baza podataka	*Open-source*
Podrška transakcijama	Da. Kompleksne transakcije. Ažuriranja se mogu konfigurirati na način da se izvrše kompletno ili uopće ne.	U određenim okolnostima i razinama. Jednostavne transakcije.
Upravljanje podacima	Pomoću SQL naredbi (SELECT, UPDATE, DELETE)	Pomoću objektno orijentiranih API-ja
Konzistentnost (dosljednost) podataka	Stroga konzistentnost podataka.	Ovisi o bazi podataka koja se koristi. Npr. MongoDB može pružiti snažnu konzistentnost podataka dok Cassandra pruža eventualnu konzistentnost.

ACID (Atomicity, Consistency, Isolation, Durability)	Relacijske baze podataka funkcioniraju na ovom modelu.	Varira do tehnologija, ali veliki broj noSQL rješenja žrtvuje ACID radi boljih performansi i skalabilnosti.
Primjer 1 (naredba INSERT)	`INSERT INTO book ( `ISBN`,`title`, `author` ) VALUES ( '9780992461256', 'Full Stack JavaScript', 'Colin Ihrig & Adam Bretz' );`	`db.book.insert({ ISBN: "9780992461256", title: "Full Stack JavaScript", author: "Colin Ihrig & Adam Bretz" });`
Primjer 2 (naredba UPDATE)	`UPDATE book SET price = 19.99 WHERE ISBN = '9780992461256`	`db.book.update( { ISBN: '9780992461256' }, { $set: { price: 19.99 } } );`

Tablica 4 Usporedba SQL i NoSQL baza podataka (Izvor: https://www.mongodb.com/nosql-explained)

Kada ih uspoređujemo s relacijskim, noSQL baze podataka skalabilnije, pružaju bolje performanse i njihov model podataka može rješavati neke probleme za koje relacijski model nije dizajniran:

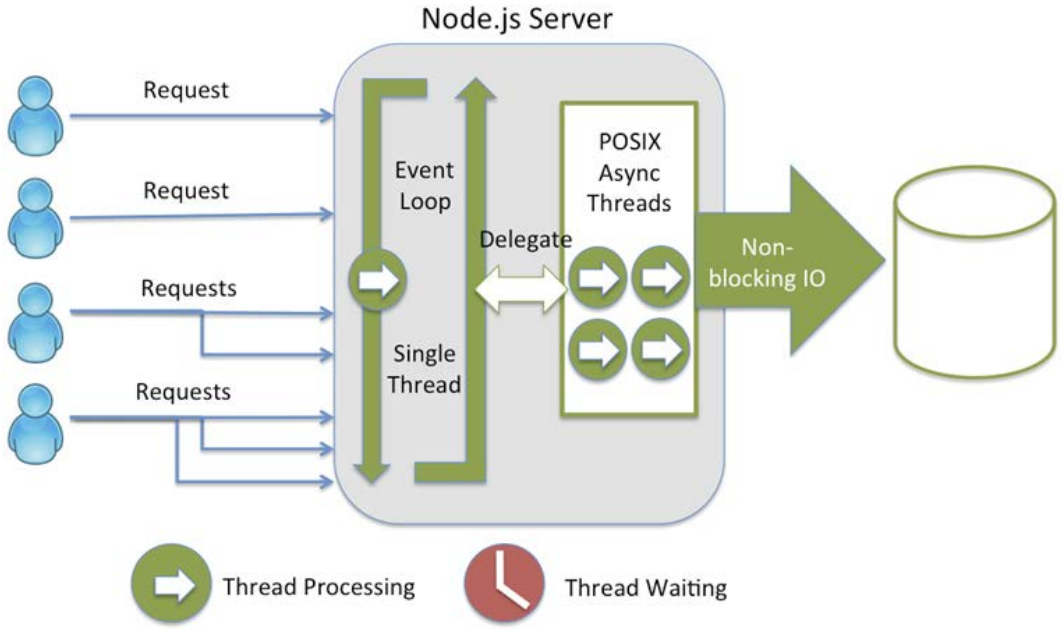
- Velike količine ubrzano rastućih strukturiranih, polustrukturiranih i nestrukturiranih podataka
- Izmjene na shemi, česte izmjene koda
- Objetno-orijentirano programiranje koje je fleksibilno i jednostavno za korištenje
- Zemljopisno distribuirane arhitekture umjesto skupih monolitnih struktura

6. Node.js

Node.js je JavaScript skalabilna programska platforma koja se odvija na serverskoj strani (eng. *server side*), a za svoj rad koristi JavaScript programski jezik. Razvio ju je Ryan Dahl 2009. godine. Definicija Node.js dana je u njihovoj službenoj dokumentaciji:

"Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices." (https://nodejs.org)

Dakle, osnovna ideja node.js platforme je implementacija neblokirajućeg sustava signalizacije čime se postiže skalabilnost i brzina, korištenje novih i bržih načina rada s ulazno-izlaznim sustavima te rad u jednoj petlji sa signalizacijom događaja.



Slika 9 Node.js arhitektura[11]

Jezgra Node.js-a napisana je koristeći module koji se automatski uključuju u svaki program napisan za ovo okruženje. Node.js sadrži veliki broj već ugrađenih modula s osnovnim, podrazumijevanim funkcionalnostima kao što su pristup datotečnom sistemu, HTTP serveru i slično. Kako bi proširio svoju funkcionalnost, Node.js također pruža bogatu biblioteku različitih Javascript modula koji olakšavaju razvoj web aplikacija. Za upravljanje paketima pomoću tim modulima koristi se NPM[12] alat koji podrazumijevano dolazi sa svakom Node.js

[11] Izvor: https://strongloop.com/wp-content/uploads/2014/01/threading_node.png
[12] NPM – alat za upravljanje JavaScript paketima (https://www.npmjs.com/)

instalacijom. Ideja je da osnova sustava bude mala i jednostavna, i da programeri na osnovu manjih stabilnih dijelova Node.js-a pišu module kojima će proširivati funkcionalnost. Ova ideja je uspješna, jer su danas dostupni deseci tisuća modula za Node.js.

Značajke Node.js-a[13]:

- *Front-end* + *back-end* - Programeri su umogućnosti koristiti isti programski kôd na klijentskoj i poslužiteljskoj strani što znači da se da kompletna aplikacija može realizirati pomoću jednog programskog jezika
- Asinkronost - Node.js platforma se temelji na ne blokirajućem (eng. *non-blocking*), asinkronom (eng. *asynchronous*) modelu. Npr. poslan je zahtjev za dohvaćanje određenog resursa sa nekog poslužitelja, i umjesto da dretva koja je pozvala metodu čeka, ona nastavlja dalje normalno sa radom.
- Brizina - izgrađen na Javascript Engine-u V8, Node.js bibiloteka je izuzetno brza kod izvršavanja kôda. Nekoliko puta je brži od ostalih skriptnih jezika poput Ruby-a i Python-a. Neblokirajuća arhitektura Node.js je idealna za pravljenje *real-time* web aplikacija.
- Javascript je korišten od strane mnogih (NoSQL) baza podataka, pa je tako komunikacija sa ovim bazama podataka putem Node.js-a prirodna
- Jedna dretva, visoka skalabilnost - Node. js za sve operacije koristi isključivo samo jednu dretvu.

Neki od popularnijih NPM modula su:

- express – Express.js, je brzi, minimalistiški web framework. Standard je za većinu Node.js aplikacija danas.
- connect – Connect je HTTP server framework za Node.js, koji omogućava kolekciju *pluginova* širokog spektra performansi, poznatijih kao *middleware*
- jade – Jedan od najpopularnijih templating engine-a, inspiriran HAML-om.
- mongo – Omogućava API za MongoDB objekte za bazu u Node.js
- mongoose – Dodatak koji omgućava jednostavnije modeliranje (izradu sheme) i validaciju za MongoDB bazu podataka

[13] Tutorials Point, *Node.js*, http://www.tutorialspoint.com/nodejs/

6.1 Express.js

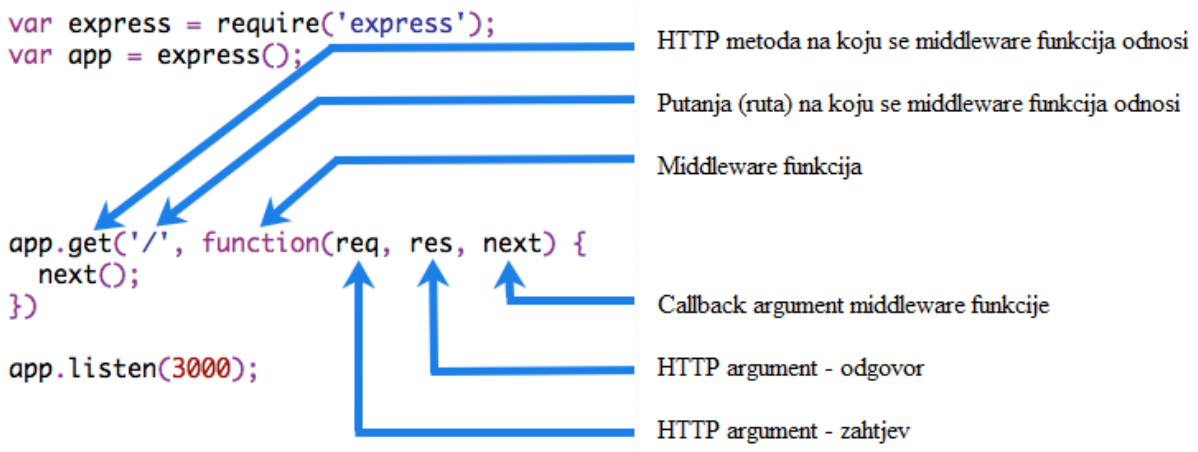
Express je minimalistički i fleksibilni Node.js web razvojni okvir (eng. *web application framework*) koji pruža močan set funkcionalnosti za razvijanje web i mobilnih aplikacija. Izdan ja kao besplatan, open-source softver pod MIT licencom. U principu, Express ubrzava razvoj Node web aplikacija, olakšava organizirati aplikaciju u MVC (*Model View Controller*) arhitekturu na serverskoj strani. Jednostavna definicija sa službene Express.js web stranice:

„Fast, unopinionated, minimalist web framework for Node.js.“ (http://expressjs.com/)

Osnovne značajke Express okvira:

- Dozvoljava postavljanje "*middlewarea* " koji odgovaraju na HTTP zahtjeve
- Definira tablicu putanja (eng. *routes*) koja se koristi za obavljanje različitih akcija baziranih na HTTP metodi i URL-u (*Uniform Resource Identifier*)
- Dozvoljava dinamično prikazivanje HTML stranica koje se temelji na argumentima koji su prenešeni u predloške (eng. *templates*) kao što je npr. Jade

Middleware funkcije su funkcije koje imaju pristup objektu s odgovorom (eng. *request object*), objektu sa zahtjevom (eng. *response object*). U principu ove funkcije mogu izvršiti bilo kakav kôd, raditi promjene nad objektima s zahtjevom/odgovorom, završiti ciklus zahtjeva/odgovora, pozvati sljedeću *middleware* funkciju u stog... [14]



Slika 10 Express.js *middleware* funkcija (Izvor: https://expressjs.com/en/guide/writing-middleware.html)

[14] Express.js, Writing middleware for use in Express apps, Izvor: https://expressjs.com/en/guide/writing-middleware.html

6.2 Jade

Jade je *template engine* koji omogućava korištenje statičnih predložaka u aplikaciji te je najčešće vezan uz Javascript aplikacije. *Template engine* je biblioteka ili razvojni okvir koji koristi neka pravila/jezike za interpretaciju podataka i prikazivanje pogleda (eng. *views*). U slučaju web aplikacija, pogledi su HTML stranice (ili dijelovi HTML stranica), ali mogu biti i JSON ili XML datoteke. U konceptu *model-view-controller*, predlošci spadaju u *view*. U vrijeme izvođenja, Jade zamjenjuje varijable u predlošku sa stvarnim vrijednostima i transformira predložak u HTML datoteku koja se šalje klijentu. Ovaj pristup olakšava dizajniranje HTML stranice.

Slika 11 Primjer Jade kôda i odgovarajući HTML *output* (Izvor: https://naltatis.github.io/jade-syntax-docs/)

7. Praktični rad

Dosada opisani koncepti, implementirani su unutar aplikacije za upravljanje dokumentima. Sustav je zamišljen da se koristi u firmi Textum d.o.o., međutim aplikacija koju ću opisati u nastavku nije finalni proizvod. Za praktično korištenje aplikacije bit će potrebno poraditi na nekim dodatnim funkcionalnostima i na sigurnosti.

7.1 Zahtjevi aplikacije

U nastavku će biti opisani osnovni zahtjevi aplikacije te će biti posebno označeni oni zahtjevi koji su implementirani u ovom radu.

Zahtjevi aplikacije		
Korištenje aplikacije putem web preglednika	Aplikacija mora biti dostupna bilo kada i bilo gdje, tako da je *web-based* aplikacija logično rješenje.	✓
Korisničko sučelje na engleskom jeziku	Tvrtka Textum posluje u cijelom svijetu, tako da sustav mora biti na engleskom jeziku.	✓
Prijava/odjava i registracija korisnika **Autentifikacija korisnika**	Forma za prijavu korisnika i registraciju korisnika. Pri registraciji, podaci o korisniku spremaju se u bazu podataka (MongoDB) te se korisniku omogućava pristup sustavu.	✓
Ograničenje i kontrola pristupa korisnika	Za svakog korisnika u sustavu mora biti definirano kakva prava ima, tj. koje dokumente može samo čitati, koje čitati i uređivati, a koje može čitati, uređivati i brisati.	✗
Ispis korisničkih podataka	Kada korisnik pristupi sustavu, mora postojati opcija pregleda podataka kao što su korisničko ime, email adresa i sl.	✓
Pohrana dokumenata	Sustav mora omogućiti korisniku	✓

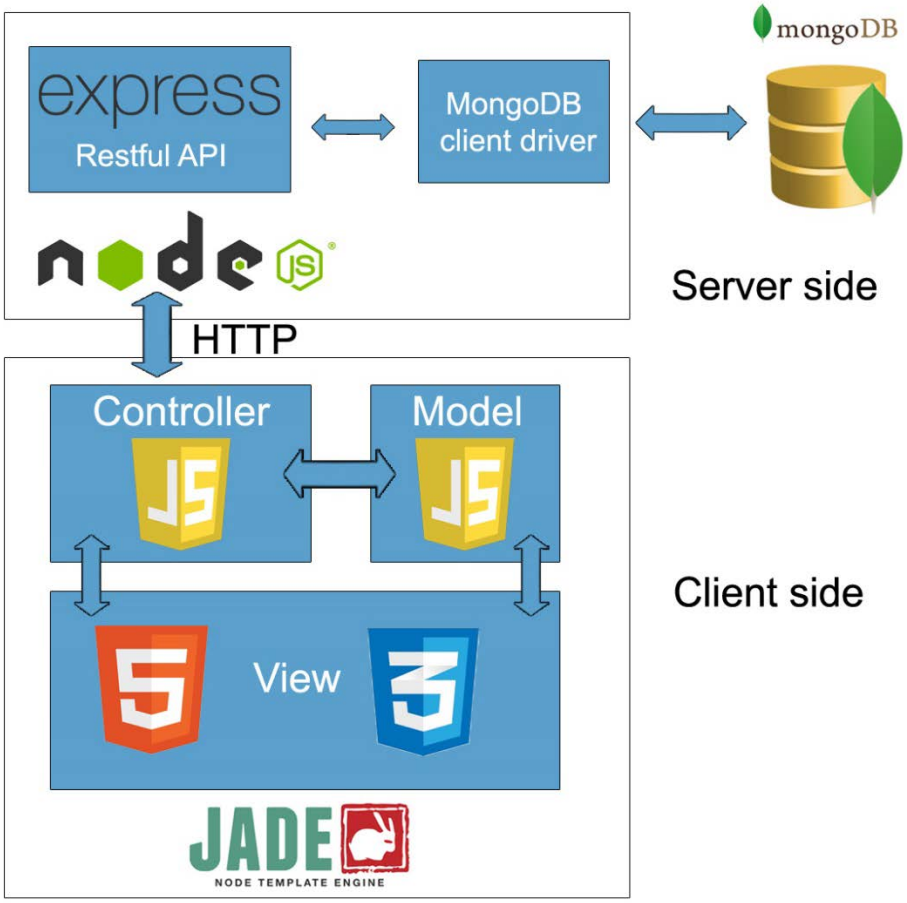
	pohranu podataka te podržavati različite tipove.	
Indeksiranje dokumenata	Pri svakom dodavanju dokumenata u sustav, dokumentu mora biti dodijeljen jedinstveni identifikator.	✓
Pregled dokumenata	Korisniku mora biti omogućen pregled svih dokumenata (onih nad kojima ima pravo pristupa)	✓
Praćenje dokumenata	Kada jednom dokument uđe u sustav, mora biti omogućena opcija za prikaz informacija o tome tko ga je kreirao i kada te tko ga je zadnji mijenjao i kada.	✓ / ✗
Verzioniranje dokumenata	Ako se jedan dokument mijenja i sprema više puta, mora biti omogućena opcija na prijašnje verzije te se mora vidjeti tko i kada je izvršavao promjene na dokumentu.	✓ / ✗
Pretraživanje dokumenata	Korisniku mora biri omogućeno pretraživanje dokumenata po određenim kriterijima	✓
Upravljanje dokumentima	Korisnik mora biti omogućeno brisanje dokumenata, preuzimanje dokumenata i uređivanje dokumenata.	✓

7.2 Implementacija aplikacije

Praktični dio rada, sustav za upravljanje dokumentima, implementiran je kao web aplikacija koristeći sve tehnologije koje su navedene u ovom radu. Na strani klijenta (eng. *front-end*) korištene su tehnologije HTML, CSS, Javascript, jQuery[15]. Na strani poslužitelja (eng. *back-end*) korišten je razvojni okvir Node.js. Za bazu podataka korišten je MongoDB sustav za upravljanje bazama podataka.

7.2.1 Arhitektura aplikacije

Na Slika 12 je prikazana arhitektura aplikacije za upravljanje dokumentima.



Slika 12 Arhitektura DMS sustava

[15] jQuery je JavaScript bibilioteka

7.2.2 Podešavanje razvojnog okruženja

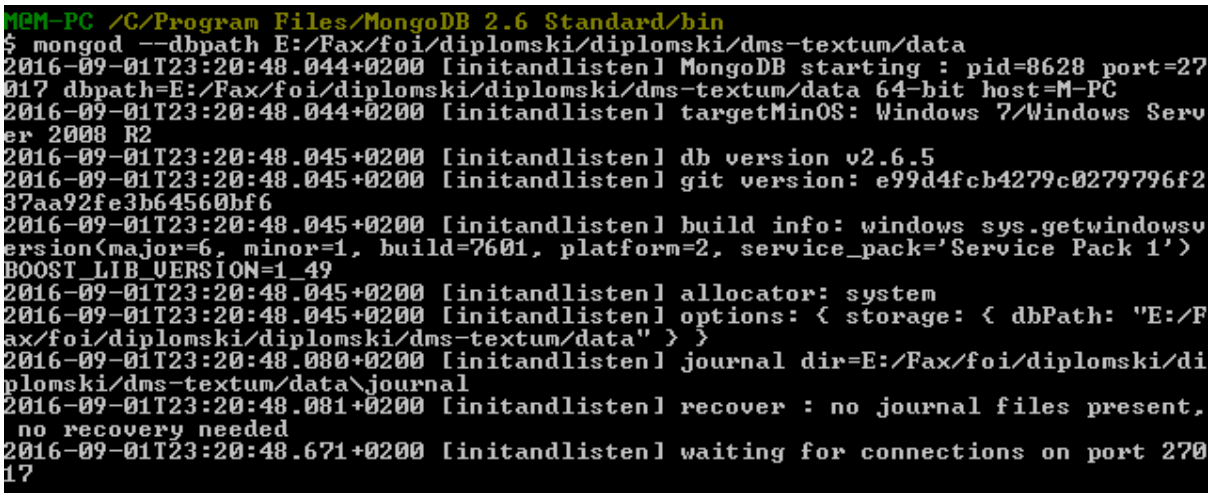
Prije početka izrade svake aplikacije potrebno je podesiti razvojno okruženje. Ovaj projekt rađen je na Windows operativnom sustavu, a za njegovu izradu bilo je potrebno instalirati:

1. MongoDB
2. Node.js

MongoDB instalacija

Sa službene stranice[16] može se preuzeti instalacijskih paket (.msi) za Windows. Nakon što je instalacija završena, potrebno je pozicionirati se u mapu gdje je Mongo instaliran, pokrenuti konzolu te podesiti direktorij u koji će se spremati svi podaci sljedećom naredbom:

```
mongod –dbpath E:/Fax/foi/diplomski/diplomski/dms-textum/data
```



Slika 13 Postavljanje MongoDB okruženja

Dobivamo poruku da je okruženje podešeno te da čeka na ulazu (eng. *port*) 27017.

Kako bi pokrenuli Mongo, u konzolu jednostavno upišemo naredbu:

```
mongo
```

Slika 14 Pokretanje MongoDB-a

[16] Službena MongoDB web stranica: https://www.mongodb.com/

Mongo zatim šalje poruku o inačici verzije te da je spojen sa bazom. Mongo je sada spreman za korištenje.

Node.js i Express.js instalacija

Node.js također na službenoj stranici[17] ima instalacijski paket za Windows. Nakon preuzimanja i instalacije potrebno je izvršiti sljedeću naredbu u direktoriju u kojem želimo da se aplikacija nalazi:

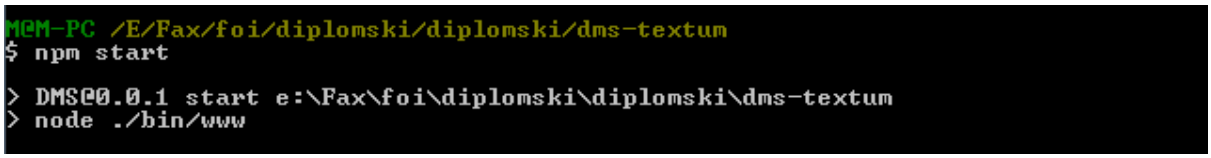
```
npm install -g express
```

Stvorila se struktura aplikacije. Sada moramo instalirati osnovne module naredbom:

```
npm install
```

Sve je spremno za pokretanje Nodea naredbom:

```
npm start
```



Slika 15 Poruka koju označava da je Node.js server pokrenut

Ovu poruku dobivamo ako je sve prošlo u redu. To znači da je Node.js server pokrenut i da se vrti u pozadini, skupa s Expressom i da je Jade HTML predprocesor instaliran. Ako sada u web pregledniku upišemo adresu http://localhost:3000 dočekat će nas poruka dobrodošlice na Express stranici.

7.2.3 Baza podataka

Kao što je već rečeno, za bazu podataka korišten je MongoDB sustav za upravljanje bazama podataka. U aplikaciji se koriste dvije kolekcije (eng. *collection*) a to su *users* i *documents*. Kolekcije se nalaze unutar baze koja se zove *app_db*.

[17] Node.js službena stranica: https://nodejs.org/

7.2.3.1. Kolekcija users

Unutar kolekcije *users,* nalaze se dokumenti o korisnicima. Dokument u NoSQL bazama podataka je ekvivalentan redu u SQL bazama. Iako znamo da MongoDB baza podataka ne zahtjeva strogu shemu, u ovom slučaju je ona potrebna radi validacije korisnika. Dokumenti korisnika imaju sljedeću shemu:

```
{
        id: String,
        username: String,
        password: String,
        email: String,
        firstName: String,
        lastName: String,
        department: String
}
```

Kada je korisnik spremljen u bazu, možemo pretraživati dokumente pomoću funkcije `find()`. Naredbom `use app_db` odabire se baza podataka nad kojom se žele izvršavati upiti. Upit koji se vidi na Slika 16 ima sljedeće stavke: `db` označava bazu u kojoj se trenutno nalazimo. `Users` se odnosi na kolekciju koju želimo pretraživati. Funkcija `find()` pretražuje kolekciju prema zadanim uvjetima. Funkcija `pretty()` uljepšava rezultat tako da je čitljiv ljudima.

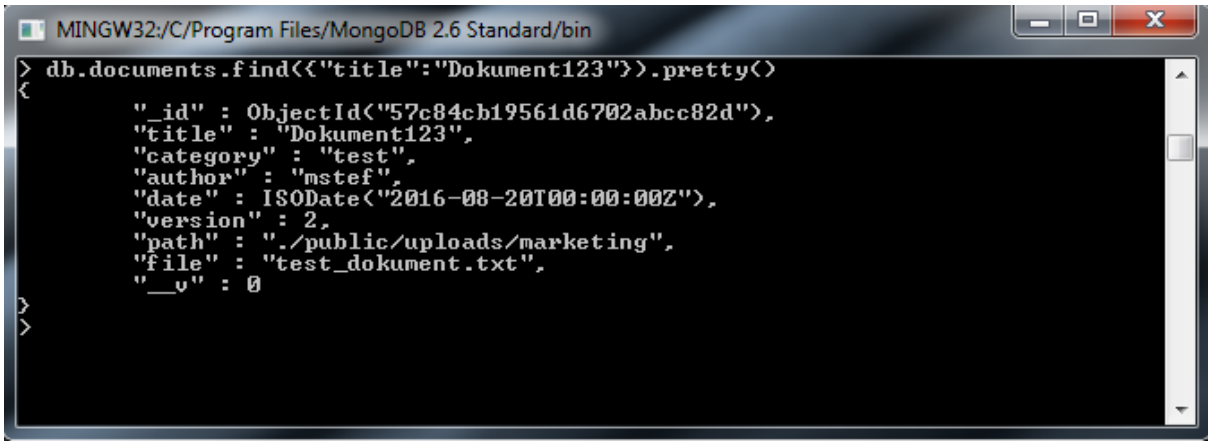
Slika 16 Upit traženja korisnika nad kolekcijom users

7.2.3.2. Kolekcija documents

Unutar kolekcije *documents*, nalaze se podaci o svim dokumentima koje korisnici šalju na server. Ovako izgleda njihova shema:

```
{
        id: String,
        title: String,
        category: String,
        author: String,
        version: Number,
        date: Date,
        path: String,
        file: String
}
```

Kao i kod kolekcije s korisnicima, prikažimo upit traženja u kolekciji *documents*:



Slika 17 Upit traženja dokumenta u kolekciji documents

7.2.4 Kreiranje prijave, registracije i provjere korisnika

Nakon instaliranja svih potrebnih alata, moguće je započeti s izradom prve funkcionalnosti ove aplikacije a to je kreiranje forme za prijavu i registraciju korisnika.

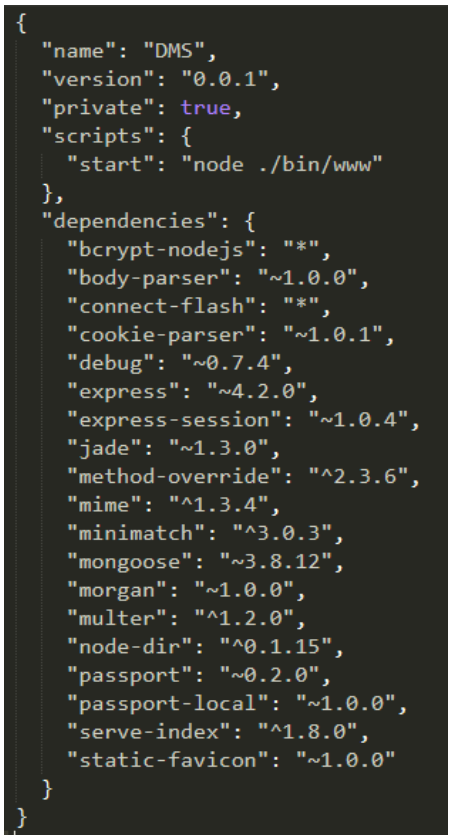
U prethodnom poglavlju, naredbom express, generirana je struktura direktorija ove aplikacije. U korijenskoj (eng. *root*) mapi nalaze se datoteke:

- `package.json` – datoteka koja sadrži sve informacije o paketima koji su potrebni za rad Node aplikacije, tj. paketima o kojima aplikacija ovisi.
- `app.js` – glavna datoteka, srce Node aplikacije gdje će biti konfigurirana cijela aplikacija

`package.json`

Ovako izgleda `package.json` aplikacije koja se opisuje u ovom radu. Pored naziva aplikacije, verzije i ostalog, najvažnije svojstvo su *dependencies* (ovisnosti) . Ako ti paketi

nisu instalirani, aplikacija se neće moći pokrenuti. Neki paketi dolaze podrazumijevano (eng. *default*) s instalacijom Nodea i Expressa, a neke su dodane naknadno što će biti opisano u nadolazećim poglavljima.



```
{
  "name": "DMS",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "bcrypt-nodejs": "*",
    "body-parser": "~1.0.0",
    "connect-flash": "*",
    "cookie-parser": "~1.0.1",
    "debug": "~0.7.4",
    "express": "~4.2.0",
    "express-session": "~1.0.4",
    "jade": "~1.3.0",
    "method-override": "^2.3.6",
    "mime": "^1.3.4",
    "minimatch": "^3.0.3",
    "mongoose": "~3.8.12",
    "morgan": "~1.0.0",
    "multer": "^1.2.0",
    "node-dir": "^0.1.15",
    "passport": "~0.2.0",
    "passport-local": "~1.0.0",
    "serve-index": "^1.8.0",
    "static-favicon": "~1.0.0"
  }
}
```

Slika 18 Package.json datoteka

`app.js`

Prvo što se mora definirati na početku ove datoteke su zahtjevi za paketima koji su definirani u package.json dokumentu. Na primjer:

```
var express = require('express');
```

Funkcija `require()` se koristi za učitavanje modula, zbog čega je njegova povratna (eng. *return*) vrijednost najčešće dodijeljena varijabli.

```
var app = express();
```

Ova linija je važna jer instancira Express i pridružuje mu varijablu `app`. Ova varijabla se dalje koristi kako bi se konfigurirale ostale Express funkcionalnosti:

```
// view engine setup
```

```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(favicon(__dirname + '/public/favicon.ico'));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);
```

Ovaj dio kôda govori aplikaciji gdje naći poglede (eng. *views*), koji *engine* koristiti za prikaz tih pogleda (Jade) i poziva još nekoliko metoda. Zadnja linija govori Expressu da poslužuje statične objekte iz /public/ direktorija, ali da izgleda kao da dolaze s najviše razine (isto radi i s /views/ direktorijem). Kao primjer, /images/ direktorij se nalazi u E:/Fax/foi/diplomski/diplomski/dms-textum/public/images ali mu se pristupa na http://localhost:3000/images.

Zbog izrade funkcionalnosti prijave i registracije korisnika, u app.js dodane su sljedeći moduli:

```
var passport = require('passport');
var bCrypt = require('bcrypt-nodejs');
var mongoose = require('mongoose');
```

Passport je *middleware* za provjeravanje autentičnosti korisnika. Mongoose je predložak za pisanje MongoDB validacije, provjeru valjanosti i poslovne logike. Iako znamo da MongoDB baza podataka ne zahtjeva strogu shemu, u ovom slučaju je ona potrebna radi validacije korisnika. Bcrypt omogućava *hashing* (sažimanje) lozinke.

7.2.4.1. Konfiguracija kolekcije korisnika

Kako bi se uspješno spojili s bazom, potrebno je navesti putanju do baze, naredbe koje omogućuju spajanje te kreirati shemu korisnika. To je napravljeno unutar tri datoteke:

`db.js` – url putanja do baze `app_db`

```
module.exports = { 'url' : 'mongodb://localhost/app_db'}
```

`app.js` – spajanje na bazu `app_db`

```
var dbConfig = require('./db');
var mongoose = require('mongoose');
mongoose.connect(dbConfig.url);
var dbConnection = mongoose.connection;
```

`user.js` – stvaraje mongoose sheme korisnika i prevođenje (eng. *compile*) u model.

```
var mongoose = require('mongoose');
module.exports = mongoose.model('User',{
        id: String,
        username: String,
        password: String,
        email: String,
        firstName: String,
        lastName: String,
        department: String
});
```

Nakon spajanja na bazu i kreiranja sheme za korisnike, kreirat ću *view* s formom za login i registraciju.

7.2.4.2. Prijava

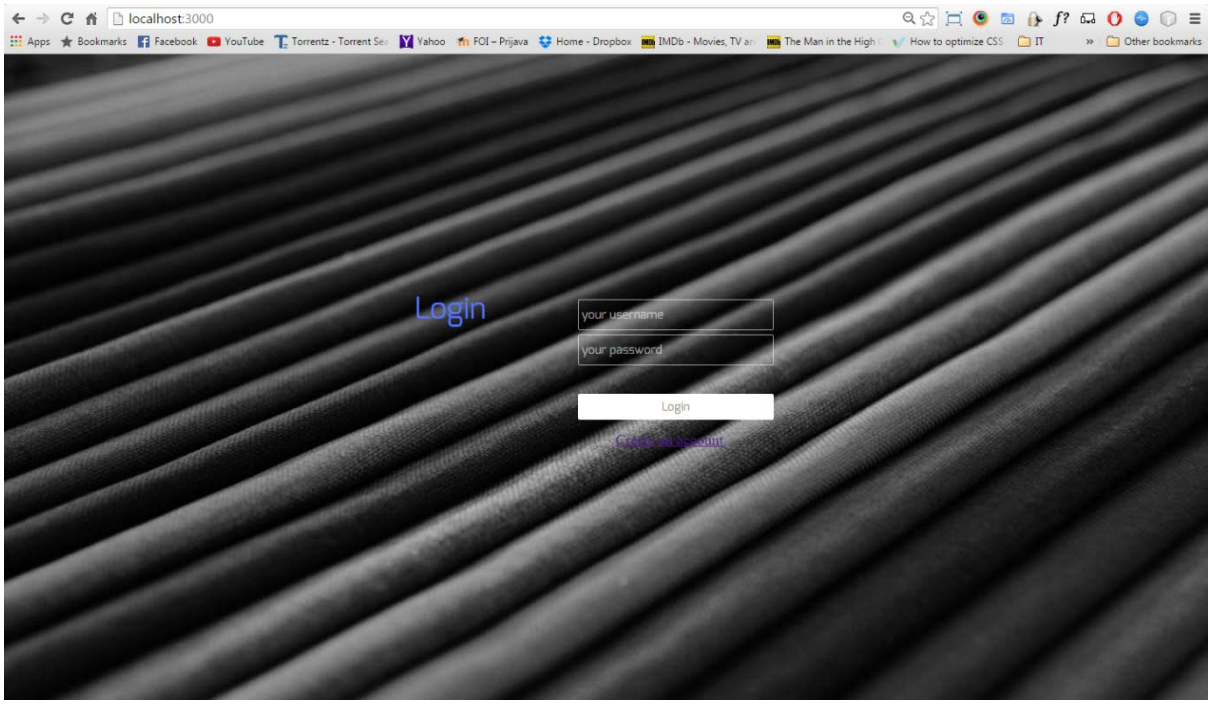
Kod kreiranje prijave korisnika, prvo sam napravila formu koja sadrži polja u koja korisnik upisuje svoje ime i lozinku. U formi se nalaze dva bitna atributa `action` i `method.` Vrijednost `post` dodaje podatke koje je korisnik upisao u formu, u tijelo HTTP zahtjeva te omogućuje njihovo daljnje korištenje. A `/login` označava naziv akcije. Pomoću ovih atributa, moći ćemo dalje manipulirati tim podacima.

`index.jade` – forma za prijavu

```
form.login.log(action='/login', method='post')
        input.logT(type='text' id="usern" name='username' placeholder='your
        username' required, autofocus)
        input.logT(type='text' id="userp" name='password' placeholder='your
        password' required)
        input.logB(type="submit" id="log" name="loginB" value='Login')
        br
        a(href='/signup') Create an account
        #message
                if message
                        h1.text-center.error-message #{message}
```

`index.js` – dohvaćanje i renderiranje `index.jade`

```
// GET login page
router.get('/', function(req, res) {
        res.render('index', { message: req.flash('message') });
});
```



Slika 19 *View* za prijavu korisnika u sustav

Dalje, treba kreirati funkciju za provjeru podataka koje je korisnik unio. U login.js datoteci se sprema model korisnika u varijablu `User`. Također se pozivaju paketi passport i bcrypt. Funkcija provjerava postoji li u bazi korisnik s korisničkim imenom i lozinkom koju

je korisnik upisao u formu. Ako ne postoji korisničko ime, ispisuje se odgovrajuća poruka. Za provjeru lozinke poziva se funkcija `isValidPassword()` .

`login.js`

```
var LocalStrategy   = require('passport-local').Strategy;
var User = require('../models/user');
var bCrypt = require('bcrypt-nodejs');

module.exports = function(passport){

    passport.use('login', new LocalStrategy({
            passReqToCallback : true
        },
        function(req, username, password, done) {
            // check in mongo if a user with username exists or not
            User.findOne({ 'username' :  username },
                function(err, user) {
                    // In case of any error, return using the done method
                    if (err)
                        return done(err);
                    // Username does not exist, log the error and redirect back
                    if (!user){
                        console.log('User Not Found with username '+username);
                        return done(null, false, req.flash('message', 'User Not found.'));
                    }
                    // User exists but wrong password, log the error
                    if (!isValidPassword(user, password)){
                        console.log('Invalid Password');
                        return done(null, false, req.flash('message', 'Invalid Password')); // redirect back to login page
                    }
                    // User and password both match, return user from done method
                    // which will be treated like success
                    return done(null, user);
                }
            );

        })
    );
    var isValidPassword = function(user, password){
        return bCrypt.compareSync(password, user.password);
    }
}
```

Slika 20 Login.js - funkcija za provjeru korisnika

Nakon što korisnik unese svoje podatke i pretisne gumb 'Login', pokreće se funkcija koja provjerava unešene podatke, te preusmjerava na /home ako su valjani. U suprotnom šalje poruku o pogreški.

`index.js` - provjera unešenih podataka

```
/* Handle Login POST */
router.post('/login', passport.authenticate('login', {
        successRedirect: '/home',
        failureRedirect: '/',
        failureFlash : true
}));
```

7.2.4.3. Registracija

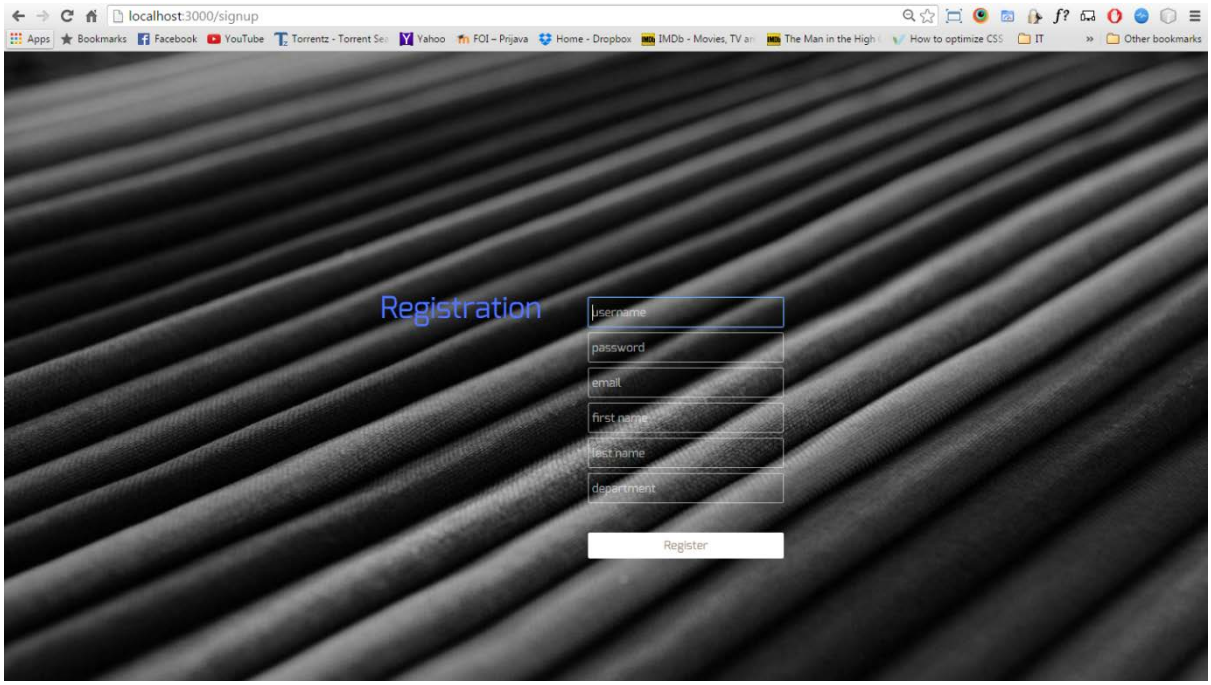
Slično kao i kod prijave, kreirala sam formu za registraciju koja sadrži polja u koja korisnik mora unijeti podatke te.

`register.jade` – forma za registraciju

```
form.login.log.reg(action='/signup', method='post')
 input.logT(type='text' id="usern" name='username' placeholder='username' required)
 input.logT(type='text' id="userp" name='password' placeholder='password' required)
 input.logT(type='text' id="usere" name='email' placeholder='email' required)
 input.logT(type='text' id="userfn" name='firstName' placeholder='first name' required)
 input.logT(type='text' id="userln" name='lastName' placeholder='last name' required)
 input.logT(type='text' id="userd" name='department' placeholder='department' required)
 input.logB(type="submit" id="log" name="loginB" value='Register')
  #message
   if message
    h1.text-center.error-message #{message}
```

`index.js` – prikazivanje stranice za registraciju

```
router.get('/signup', function(req, res){
        res.render('register',{message: req.flash('message')});
});
```



Slika 21 View za registraciju korisnika u sustav

Prilikom registracije treba provjeriti je li korisnik unio sve potrebne podatke, je li ih unio na ispravan način i postoji li taj korisnik već u bazi. Za svaki od tih slučajeva potrebno je korisniku poslati odgovarajuću poruku pogreške. To sve se odvija u datoteci signup.js.

Funkcija `findOrCreateUser()` kreira novog korisnika ili provjerava je li već postojeći. Jednom kada je korisnik spremljen u bazu, možemo pretraživati dokumente pomoću funkcije `findOne()`. Ta funkcija uzima za uvjet korisničko ime. Ako korisničko ime koje je korisnik upisao već postoji, ispisuje se odgovarajuća poruka. Ako korisnik ne

postoji u bazi, naredbom `req.parameters()` dohvaćaju se podaci koje je korisnik upisao koristeći svojstvo `'name'` i spremaju se u varijablu. Zatim se poziva već kreiran model `User` te se u njega spremaju podaci iz varijabli. Zatim se sve sprema u bazu.

`signup.js`

```
function(req, username, password, done) {

    findOrCreateUser = function(){
        // find a user in Mongo with provided username
        User.findOne({ 'username' :  username }, function(err, user) {
            // In case of any error, return using the done method
            if (err){
                console.log('Error in SignUp: '+err);
                return done(err);
            }
            // already exists
            if (user) {
                console.log('User already exists with username: '+username);
                return done(null, false, req.flash('message','User Already Exists'));
            } else {
                // if there is no user with that email
                // create the user
                var newUser = new User();

                // set the user's local credentials
                newUser.username = username;
                newUser.password = createHash(password);
                newUser.email = req.param('email');
                newUser.firstName = req.param('firstName');
                newUser.lastName = req.param('lastName');
                newUser.department = req.param('department');

                // save the user
                newUser.save(function(err) {
                    if (err){
                        console.log('Error in Saving user: '+err);
                        throw err;
                    }
                    console.log('User Registration succesful');
                    return done(null, newUser);
                });
            }
        });
    };
```

Slika 22 Funkcija findOrCreateUser()

Treba spomenuti i pomoćnu funkciju `createHash()`. Ona generira hash[18] ključ kojim će lozinka koju korisnik upiše biti kriptirana.

```
// Generates hash using bCrypt
var createHash = function(password){
    return bCrypt.hashSync(password, bCrypt.genSaltSync(10), null);
}
```

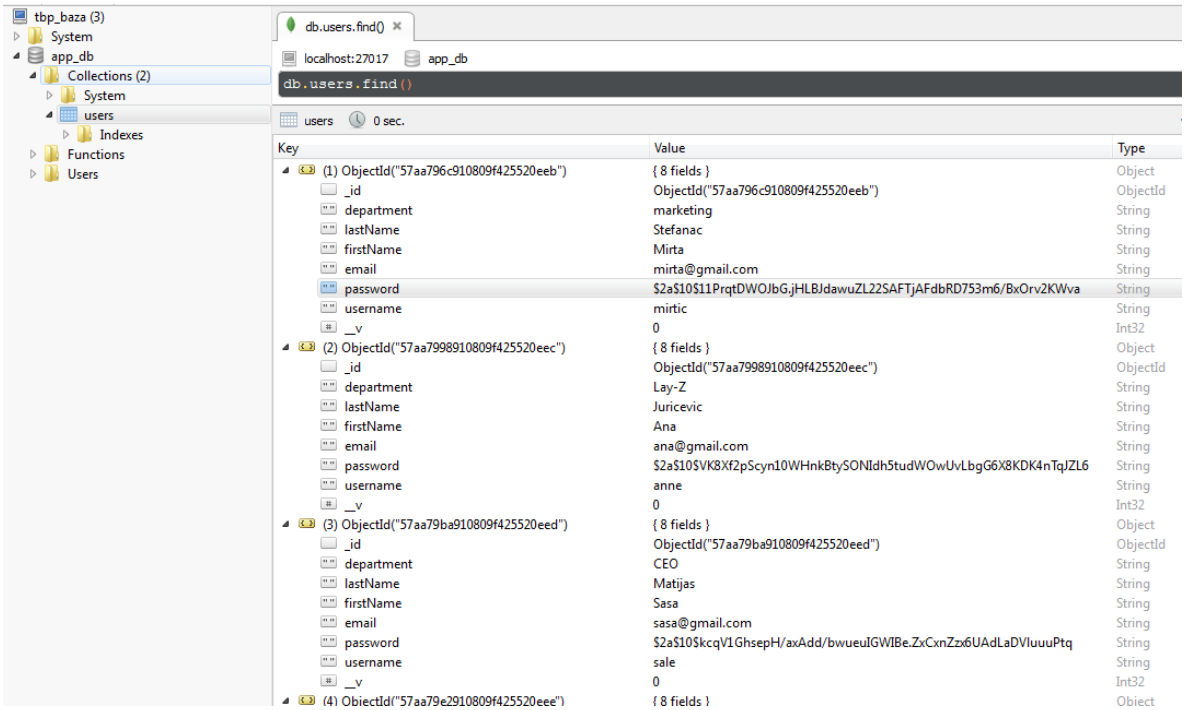
Slika 23 Funkcija za kriptiranje lozinke

[18] Kriptografske hash funkcije su funkcije s dodatnim sigurnosnim svojstvima kako bi ih se moglo koristiti za autentifikaciju i očuvanje integriteta podataka

Znači, kada su u formu za registraciju upisani su podaci, klikom na gumb „Register“ podaci se spremaju u bazu. Nije prije spomenuto da postoji program koji se zove Robomongo. Definicija sa službene stranice glasi ovako:

„Native and cross-platform MongoDB manager“[19]

Naime, to je alat koji nam pruža vizualni pregled MongoDB baze podataka (eng. GUI – *Graphic User Interface*), umjesto da za to koristimo konzolu.



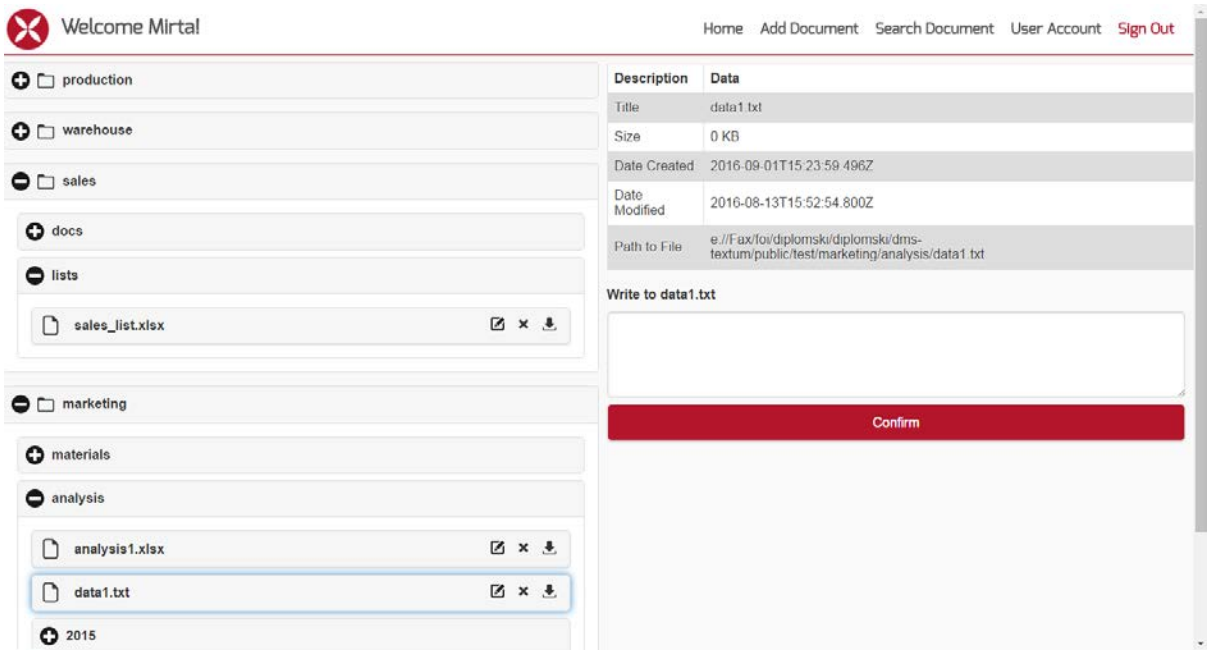
Slika 24 Robomongo - pregled kolekcije *users*

Na Slika 24 vidimo strukturu dokumenta, nazive atributa, njihove vrijednosti te tipove podataka. Na ovom prikazu se jasno može vidjeti i da je lozinka uspješno kriptirana korištenjem prije spomenute funkcije.

7.2.5 Kreiranje strukture

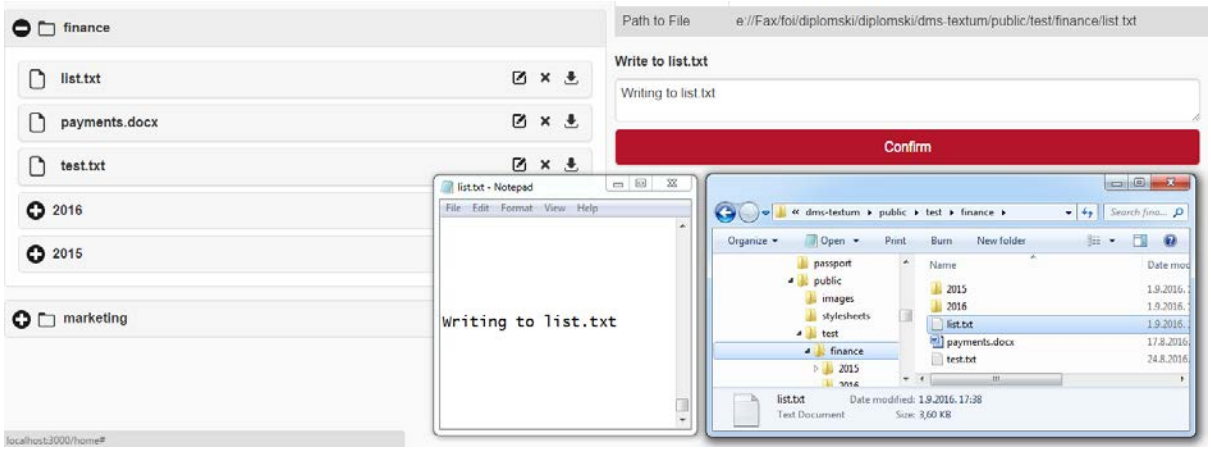
Kada je korisnik prošao autentifikaciju, ulazi u sustav. Na početnoj stranici dočekuje ga poruka dobrodošlice i navigacija u gornjem desnom kutu.

[19] https://robomongo.org/



Slika 25 Početna stranica nakon ulaska u sustav

Korisniku je omogućen pregled direktorija i datoteka s desne strane. Direktoriji su podijeljeni po odjelima, pa se ovdje tako nalaze proizvodnja, skladište, prodaja, marketing i računovodstvo. Kada korisnik odabere datoteku s desne strane će se prikazati tablica u kojoj se nalaze neki osnovni podaci o datoteci kao što su naziv, veličina, datum kreiranja, datum modificiranja i putanja do datoteke. Postoje još i opcije preuzimanja datoteke, brisanja datoteke i mijenjanja datoteke. S desne strane prozora nalazi se područje za upis teksta (eng. *textarea*). Ako se tu nešto upiše i pritisne gumb „Confirm“, vidjet ćemo promjene na toj datoteci. Trenutno aplikacija ima mogućnost modificiranja samo tekstualnih datoteka (.txt). Ovo je jedna od funkcionalnosti koja bi se trebala unaprijediti ako se aplikacija stvarno namjerava koristiti u poduzeću.



Slika 26 Pisanje u datoteku list.txt korištenjem aplikacije

Za prikaz strukture direktorija u web pregledniku, koristila sam Node *middleware* File System i Path. Kreirala sam modul *diretoryTreeToObj*, Slika 27. Ovdje se provjeravaju sve mape i datoteke koje se nalaze ispod zadane putanje. U petlji se za svaki dio strukture provjerava je li naišla na mapu ili na datoteku i dohvaćaju se detalji o njima kao što su datum kreiranja / modificiranja, putanja, veličina i sl.

```
var diretoryTreeToObj = function(dir, done) {
    var results = [];
    fs.readdir(dir, function(err, list) {
        if (err)
            return done(err);
        var pending = list.length;
        if (!pending)
            return done(null, {name: path.basename(dir), type: 'folder', children: results});

        list.forEach(function(file) {
            file = path.resolve(dir, file);
            console.log(file);
            fs.stat(file, function(err, stat) { //get file details
                if (stat && stat.isDirectory()) {
                    diretoryTreeToObj(file, function(err, res) {
                        results.push({
                            name: path.basename(file),
                            type: 'folder',
                            uid: stat['uid'],
                            size: stat["size"],
                            dateCreated : stat["birthtime"],
                            dateModified : stat["mtime"],
                            path: file,
                            children: res
                        });
                        if (!--pending)
                            done(null, results);
                    });
                }
                else {
                    results.push({
                        type: 'file',
                        name: path.basename(file),
                        uid: stat['uid'],
                        dateCreated : stat["birthtime"],
                        dateModified : stat["mtime"],
                        size: stat["size"],
                        path: file,
                    });
                    if (!--pending)
                        done(null, results);
                }
```

Slika 27 Modul za kreiranje strukture direktorija

Ovaj modul poziva se u index.js datoteci.

```
var dirTree = ('./public/test');                //directory which is going to be listed

var fileStructureObject;                        //global variables for json object in which file structure is going to be saved

diretoryTreeToObj(dirTree, function(err, res){  //calls the function from getDirectory.js (res is json object)
    if(err)
        console.error(err);

    var fileStructureString = JSON.stringify(res, null, 2); //converts json object to string
    console.log(fileStructureString);

    fileStructureObject = JSON.parse(fileStructureString);  //converts it back to object
});
```

Slika 28 Pozivanje funkcije diretoryToObj() i definiranje putanje do direktorija

Definira se putanja do direktorija koji će biti izlistan, proslijeđuje se kao parametar funkciji `diretoryTreeToObj()`. Rezultat te funkcije se *parsira* u JSON objekt te je spreman za korištenje. Pri otvaranju stranice /homepage, pogledu `homepage.jade` prosljeđuje se taj JSON objekt (`fileStructureObject`).

```
/* GET Home Page */
router.get('/home', isAuthenticated, function(req, res){
    //sends variables user and data(file structure) to home.jade
    res.render('home', { user: req.user, parentFiles : fileStructureObject});
});
```

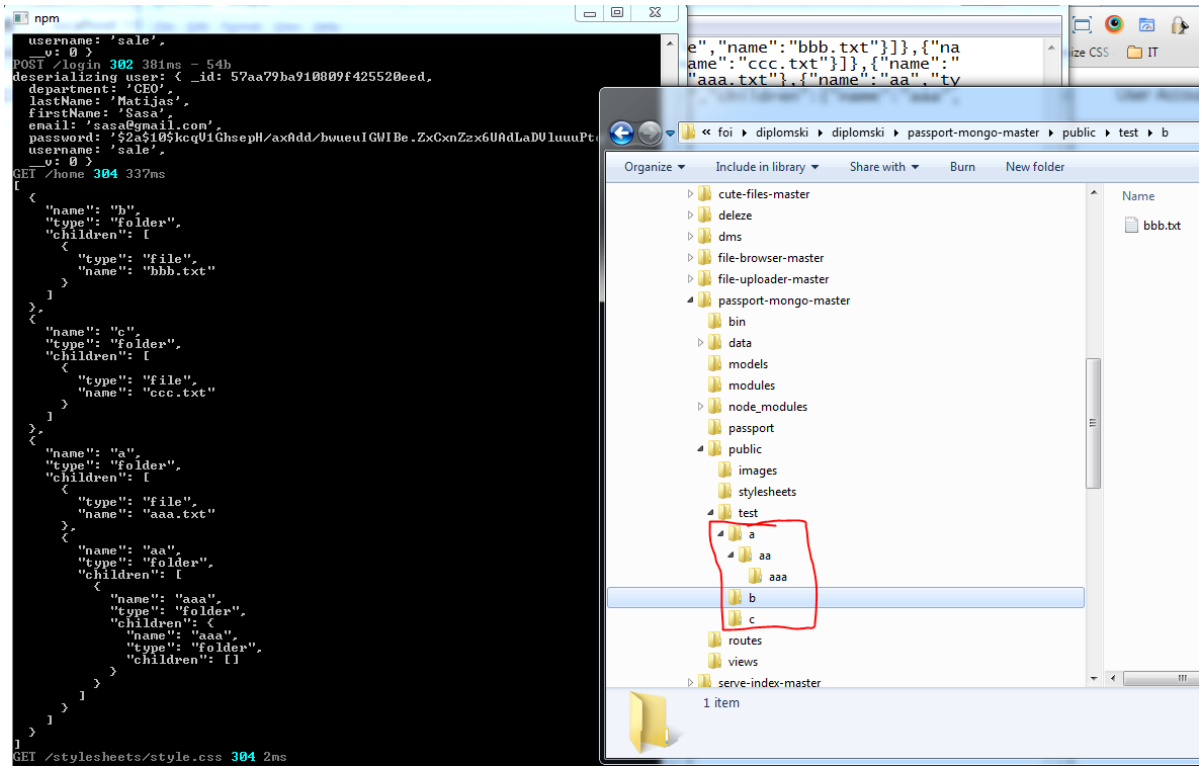
Slika 29 Renderiranje predloška home i prosljeđivanje parametra istom

Kada je objekt prosljeđen u predložak home.jade, nije komplicirano manipulirati s prikazom.

```
div.structure
    //first level
    - var numbOfParents = parentFiles.length; //counts number of 1st level folders
    - for (var j = 0; j < numbOfParents; j++) {
        div
            h1 !{parentFiles[j].name}
                img(src='http://megaicons.net/folder-icon.png' class="folder")

            //second level
            - var numbOfChildren = parentFiles[j].children.length; //counts number of 2nd level
            children
            - for (var i = 0; i < numbOfChildren; i++) {
                - if (parentFiles[j].children[i].type == 'folder'){
```

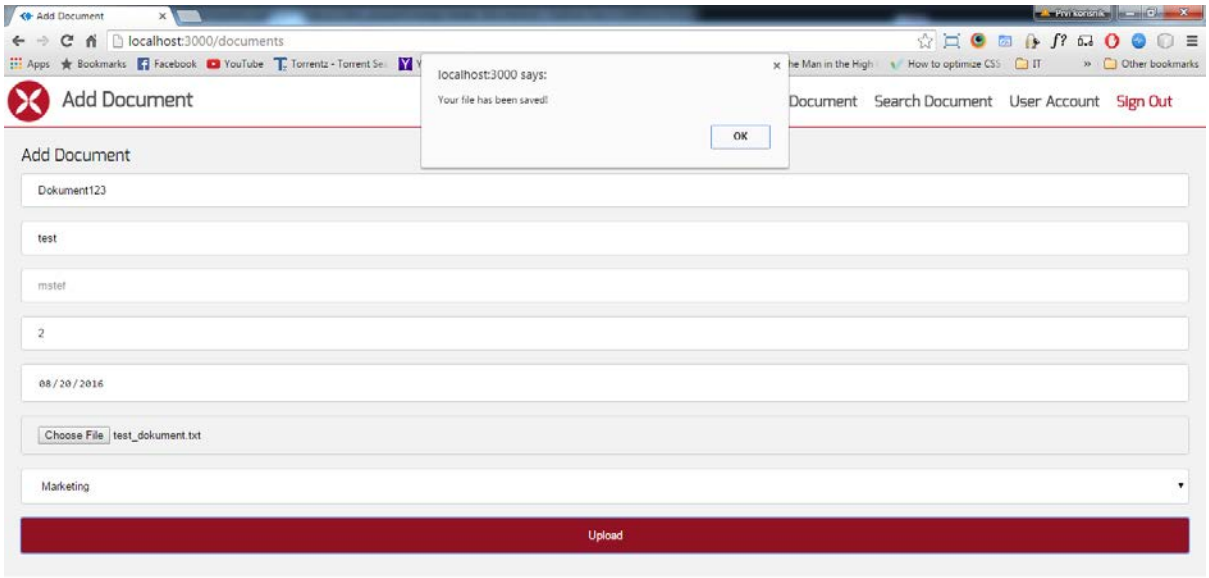
Slika 30 Isječak kôda home.jade datoteke - upravljanje json objektom i kreiranje struktura direktorija i datoteka



Slika 31 Lijevo - prikaz strukture direktorija u JSON formatu, Desno - prikaz strukture u datotečnom sustavu

7.2.6 Dodavanje dokumenta

U svakom sustavu za upravljanje dokumentima, dodavanje datoteka u sustav mora biti jedna od osnovnih funkcionalnosti. U ovoj aplikaciji, biti će omogućeno dodavanje dokumenta pomoću forme. Na Slika 32 je prikazan izgled te forme i poruka da je dokument uspješno poslan na server. Pogledajmo kako to učiniti u nastavku.



Slika 32 *View* documents.jade za dodavanje dokumenata u bazu i na server

`documents.jade`

```
form(action='/addDoc', method='post' enctype='multipart/form-data')
        input(type='text' name = 'title' placeholder='Title')
        input(type='text' name = 'category' placeholder='Category')
        input(type='text' name = 'author' value="#{user.username}" readonly)
        input(type='text' name = 'version' placeholder='Version')
        input(type='date' name = 'date' placeholder='Date')
        input(type='file' name = 'file')
        select(name="path_for_upload")
                option(value='./public/uploads/sales' selected='selected') Sales
                option(value='./public/uploads/production') Production
                option(value='./public/uploads/warehouse') Warehouse
                option(value='./public/uploads/finance') Finance
                option(value='./public/uploads/marketing') Marketing
        input(type="submit" value = 'Upload')
```

Kada se prikaže datoteka `documents.jade` u pregledniku, korisnik može ispuniti formu podacima i spremiti novu datoteku u sustav. Objasnimo kako:

Korisnik upisuje:

1. Naslov dokumenta

2. Kategoriju u koju dokument spada
3. Ime korisnika se ne može mijenjati – datoteka se sprema s ugrađenim imenom korisnika koji ju je *uploadao*
4. Broj verzije – ovu funkcionalnost također treba unaprijediti, ovdje je jednsotavno omogućeno korisniku da odabere bilo koji cijeli broj koji će označavati verziju dokumenata.
5. Datum upload-a dokumenta
6. Izabire datoteku sa svojeg računala koju želi upload-ati
7. Bira odjel u koji će dodati datoteku – ova funkcionalnost nije realizirana do kraja

Ako su svi podaci koje je korisnik upisao točni, oni se prosljeđuju u `index.js`. U formi se nalazi atribut `action` koji ima vrijednost `/addDoc`, atribut `method` koji ima vrijednost `post` i atribut `enctype` koji ima vrijednost `multipart/form-data` . Enctype atribut specificira kako će se podaci iz forme kodirati kada ih se pošalje na server. Može se koristiti samo u kombinaciji s metodom post. Post metoda šalje podatke iz forme u tijelo HTTP zahtjeva te se njima može manipulirati u sljedećoj funkciji:

```
//function for uploading file using 'multer'
router.post('/addDoc',upload.single('file'), function(req, res){
    // creates a new document by taking the user input and saves it to a database documents
    var doc = new Document({
    title: req.param('title'),
    category: req.param('category'),
    author: req.user.username, //saves the active user as author of the document
    date: req.param('date'),
    version: req.param('version'),
    path: req.param('path_for_upload'),
    file: req.file.originalname
    });

    doc.save(doc, function(err) {
        if (err){
            console.log('Error in Saving document: '+err);
            throw err;
        }
        console.log('Document inserted successfuly');
    });
});
```

Slika 33 Metoda koja prima HTTP *body request* s podacima iz forme za *upload* dokumenata i sprema te podatke

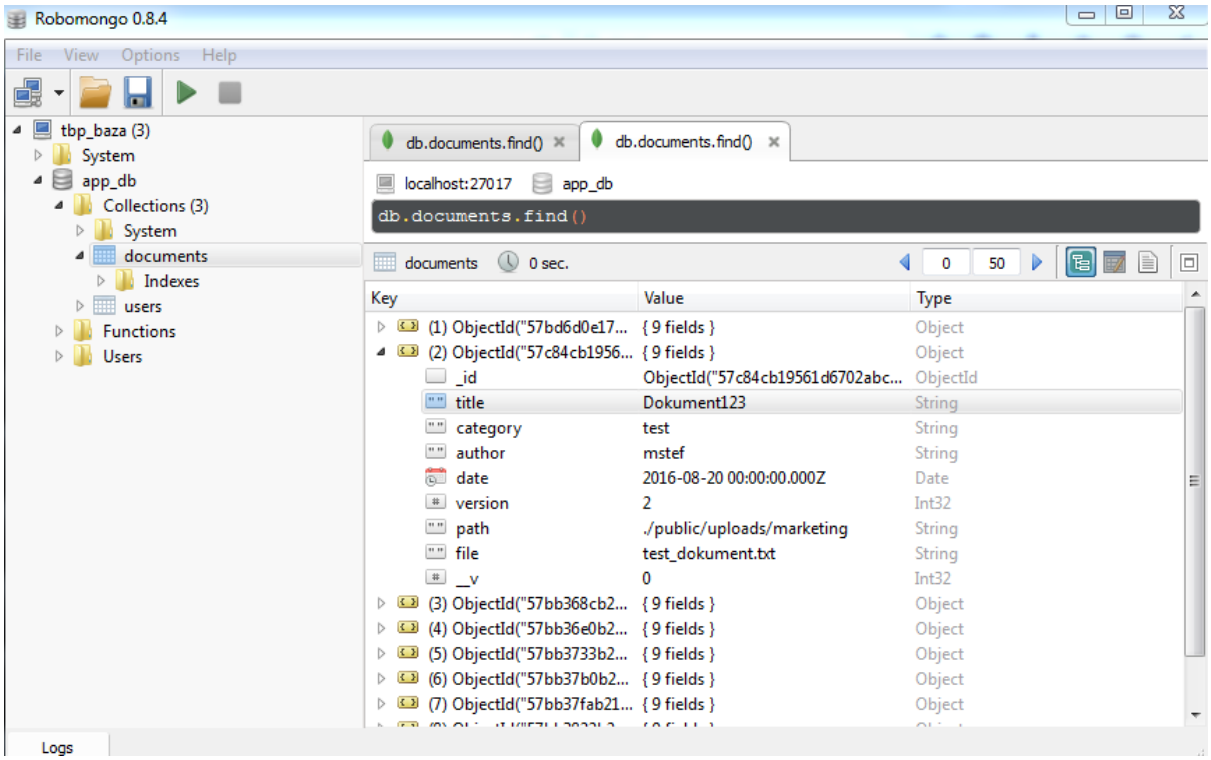
Funkcija prima te podatke, stvara novi objekt `doc` prema modelu `Document`. Dodaje atributima iz modela odgovarajuće podatke koje je korisnik upisao te ih sprema u kolekciju *documents*.

Ovako izgleda model `Document` koji se poziva u gornjoj funkciji:

```
module.exports = mongoose.model('Document',{
    id: String,
    title: String,
    category: String,
    author: String,
    version: Number,
    date: Date,
    path: String,
    file: String
});
```

Slika 34 Model Document

Sada je novi dokument dodan u bazu što možemo vidjeti na Slika 35.



Slika 35 Robomongo - kolekcija documents s prikazom dodanog dokumenta putem forme

Preostaje još samo spremiti taj dokument na server. To možemo postići korištenjem Multera. Multer je Node.js *middleware* za rukovanje s *multipart/form-data* podacima koji se najčešće koristi za *upload* datoteka.

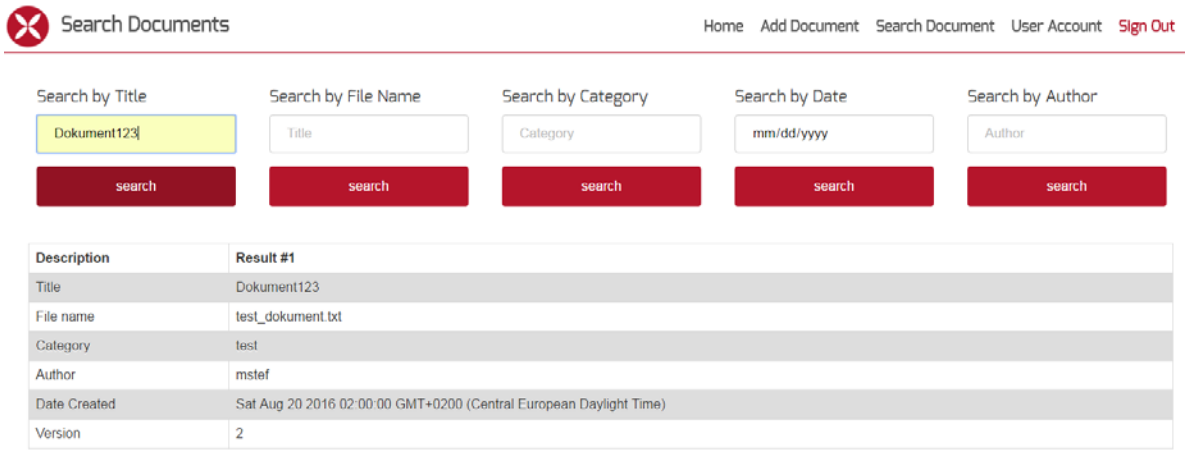
```
var multer = require('multer');
var storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, './public/uploads/')
  },
  filename: function (req, file, cb) {
      cb(null,  file.originalname );
  }
});
var upload = multer({ storage: storage });
```

Slika 36 Korištenje modula Multer

Dvije opcije, `destination` i `filename` određuju gdje će datoteka biti spremljena i pod kojim imenom. To je spremljeno u varijablu *upload* koja je prosljeđena funkciji na Slika 33.

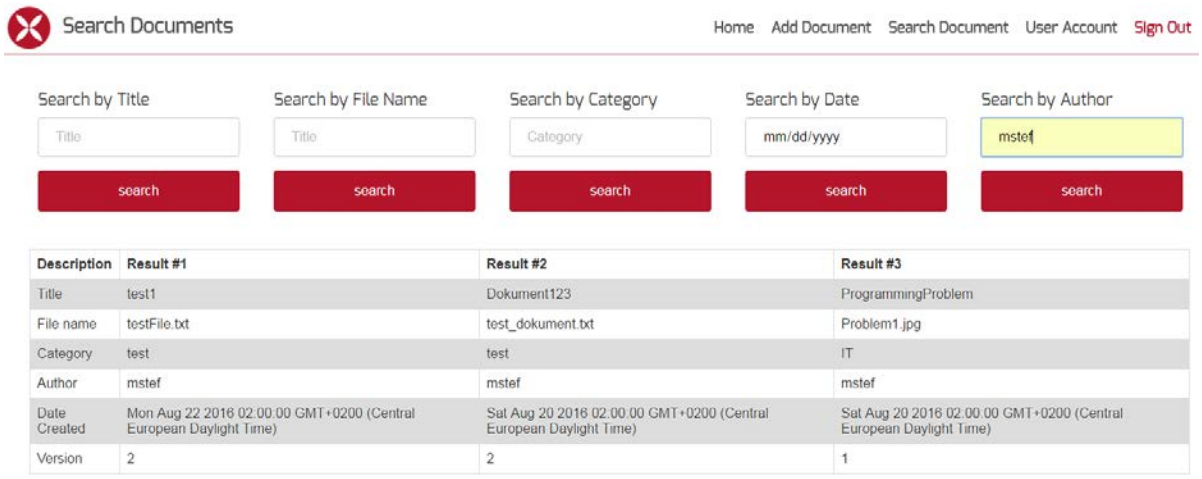
7.2.7 Pretraživanje dokumenata

Još jedna funkcionalnost koju svaki sustav za upravljanje dokuentima mora imati je pretraživanje istih. U ovoj aplikaciji korisniku je omogućeno pretraživanje prema naslovu dokumenta, imenu datoteke, kategoriji, datumu i prema autoru koji je uploadao datoteku. Na Slika 37 prikazano je pretraživanje dokumenta pod naslovom 'Dokument123', koji je u prethodnim poglavljima spremljen na server i u bazu.



Slika 37 Pretraživanje dokumenata prema nazivu

Na Slika 38 vidimo kako izgleda pretraživanje prema autoru, tj. prema korisniku koji je *uploadao* dokument.



Slika 38 Pretraživanje dokumenata prema korisniku

Svaka opcija predstavlja jednu formu u jade predlošku. Svaka forma ima svoj naziv atributa `action` koji se prepoznaje u index.js datoteci.

```
span Search by Title
        form(action='/searchByTitle', method='get')
                input(type='text' name ='docTitle' placeholder='Title')
                input(type="submit" value = 'search')
```

U index.js datoteci *router* prepoznaje akciju `/searchByTitle` te radi upit nad kolekcijom *documents* uzimajući kao uvjet naslov dokumenta kojeg je korisnik upisao u formu. Preusmjeravanje (eng. *routing*) se odnosi na definiranje krajnjih točaka (eng. *end points*) ili URI-a (*Uniform Resource Identifier*) aplikacije i na to kako treba reagirati na zahtjeve klijenata.

```
router.get('/searchByTitle', function(req, res){
    var title = req.param('docTitle');
    Document.find({ title: title }, function(err, doc) {
      if (err) throw err;
      res.locals.doc = doc;
      res.render('search');
    });
});
```

Slika 39 Funkcija koja pretražuje kolekciju *documents* prema zadanom naslovu

Rezultat koji se prikazuje u tablici ispod formi za pretraživanje kreiran je u jade predlošku. Linijom `res.locals.doc = doc;` se prosljeđuje JSON objekt koji sadrži zapis o dokumentu u jade s kojim je onda jednostavno upravljati i prikazivati ga, na primjer: `#{doc[1].title}` prikazuje naslov prvog dokumenta u tablici.

7.3 Moguće nadogradnje aplikacije

Ova aplikacija svakako ima prostora za napredak. Ukoliko će u budućnosti zaživjeti unutar jedne organizacije potrebno je napraviti nekoliko poboljšanja i dodati par funkcionalnosti. Funkcionalnosti koje bi trebalo implementirati:

- Mogućnost promjene lozinke – pošto je već implementirata funkcionalnost kriptiranja lozinke, bilo bi dobro napraviti još taj jedan mali korak koji bi uveliko olakšao život korisniku, ali i administratoru
- Ograničenja prava pristupa – ovo je možda najvažnija funkcionalnost koja nije implementirana. Bez ograničenja prava korisnika, ovakav sustav ne može funkcionirati kako treba. Kao što je već prije opisano, pri ulasku u sutav korisniku se prikazuje struktura datotečnog sustava koja je podijeljena po odjelima. Trebalo bi ograničiti pojedinog korisnika na odjel u kojem je zaposlen.
- Slanje notifikacija – umjesto da zaposlenici svakih nekoliko minuta ili sati provjeravaju sustav, bilo bi puno bolje da im dođe notifikacija na mobilni telefon ili poruka na e-mail. To bi implementirala tako da kad jedan od korisnika dodaje novi

dokument u sustav, mora odabrati kojim korisnicima ili kojim odjelima će stići notifikacija.

- Verzioniranje dokumenata – verzioniranje je djelomično implementirano u aplikaciju. Naime pri dodavanju novog dokumenta, korisnik mora izabrati broj verzije, ali se to ne odvija automatski tako da opet postoji mogućnost pogreške. Trebalo bi doraditi ovo svojstvo.

Ovo su samo neke od značajki koje treba nadograditi, a postoji i nekoliko sitnijih nedostaka i grešaka (eng. *bug*) u kôdu koje se moraju riješiti. Moram još spomenuti i da bi bilo dobro u potpunosti prilagoditi izgled aplikacije svim veličinama uređaja. Korisnici mogu ovakvom sustavu pristupati gdje god se oni nalazili tako da je svakako nužno prilagoditi izgled i manjim uređajima kao što pametni telefoni ili tableti.

8. Zaključak

Povećanjem količine podataka i dokumenata koje ulaze u firmu, tradicionalni način pohranjivanja istih jednostavno više nije adekvatan. Na primjer, razmjeri *big data* podataka su toliko veliki da ih čovjek ne može procesirati bez pomoći računala. A sve su veći i veći.

U pomoć dolaze NoSQL baze podataka i sustavi za upravljanje dokumentima. Pošto NoSQL baze podataka ne zahtijevaju shemu, savršene su za arhiviranje polustrukturiranih podataka.Također, ti sustavi su daleko skalabilniji i imaju bolje performanse od relacijskih sustava za upravljanje bazama podataka. Međutim, nije sve tako jednostavno. Postoje i neke negativne strane NoSQL baza podatka. Kao prvo, sustavi za upravljanje relacijskim bazama podataka postoje preko 40 godina, dakle zrelije su, vrlo su funkcionalne i stabilne, te za ove sustave postoji daleko veća podrška.

Zapravo je krivi pristup gledati na ove dvije tehnologije kao na konkurenciju, bolje je reći da su to alternative jer o tome kakav problem treba riješiti, ovisi koji sustav ćemo izabrati. Na primjer relacijski model je jedino rješenje ako moramo osigurati ACID (eng. *Atomicity, Consistency, Isolation, Durability*) svojstva transakcija tj. integritet podataka ili ako je zahtjev da podaci budu strogo strukturirani i ne promijenjivi. NoSQL baze, na primjer, žrtvuju ACID svojstva radi veće fleksibilnosti i brzine procesiranja. Također, nema potrebe za korištenjem NoSQL sustava ako organizacija ne doživljava goleme poraste koji zahtijevaju više servera i ako se samo radi nad podacima koji su dosljedni. Bilo bi bespotrebno trošenje resursa dizajnirati sustav koji može podržati različite tipove podataka i visoko opterećenje sustava ako se to ne zahtijeva.

Problemi s kojima se susrećemo ili ćemo se tek susretati su veliki izazov i smatram da je ovo područje interesantno i nepresušno. Osim velikog interesa koji imam prema ovoj temi odabrala sam razviti aplikaciju za upravljanje dokumentima jer tvrtka u kojoj sam odradila stručnu praksu i u kojoj ću nastaviti raditi ima veliku potrebu za jednim takvim sustavom. Aplikacija koja je prikazana u ovom radu nije, doduše, kompletno funkcionalna, jer za razvijanje jednog takvog sustava treba daleko više vremena i resursa, ali je jako dobar temelj.

9. Literatura

[1] Abiteboul S., Buneman P., Suciu D. (1998), *Data on the Web: from relations to semistructured data and XML*, Morgan Kaufmann Publishers, San Francisco, California

[2] Alfresco, (2016), *Document management*, Preuzeto s: https://www.alfresco.com/

[3] Brown E., (2014), *Web Development with Node and Express*, O'Reilly Media Inc., Prvo izdanje, SAD, California

[4] Chodorow K., (2013), *MongoDB: The Definitive Guide*, O'Reilly Media Inc., Drugo izdanje, SAD, California

[5] ECMA International, (2013), *The JSON Dana Interchange Format*, Standard ECMA – 404, Ženeva, preuzeto s: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[6] Fourny G. (2013), *JSONiq, The SQL of NoSQL*,

[7] Goldman R., McHugh J., Widom J. (1997), *Lore: A Database Management System for Semistructured Data*, Stanford University

[8] Goldman R., McHugh J., Widom J. (1999), *From Semistructured Data to XML: Migrating the Lore Data Model and Query Language*, Stanford University

[9] Hughes-Croucher T., Wilson M., (2012), *Node: Up and Running*, O'Reilly Media Inc., Prvo izdanje, SAD, California

[10] MongoDB, Inc. (2014), *Top 5 Considerations When Evaluating NoSQL Databases*, Preuzeto s: http://www.mongodb.com/lp/whitepaper/nosql-considerations

[11] MongoDB, Inc. (2016), *The MongoDB 3.2 Manual*, Preuzeto s: https://docs.mongodb.com/manual/

[12] Node.js Foundation, (2016), *Node.js v6.5.0 Documentation*, Preuzeto s: https://nodejs.org/api/documentation.html

[13] Schatten, M., Ivković, K., (2012), *Regular Path Expression for Querying Semistructured Data - Implementation in Prolog*, Central European Conference on Information and Intelligent Systems, FOI Varaždin

[14] Smith B., (2015), *Beginning JSON*, Apress, Prvo izdanje, SAD, California

[15] Suciu D., (1998), *Database Theory Column, An Overview of Semistructured Data*, AT&T Labs, San Diego, California

[16] Tehničko veleučilište u Zagrebu (2012), *Nastavni materijali iz kolegija: Baze podataka*, Zagreb

[17] Tehničko veleučilište u Zagrebu (2013), *Nastavni materijali iz kolegija: Napredne baze podataka*, Zagreb

[18] TutorialsPoint (2014). *JSON Tutorial*, Preuzeto s: http://www.tutorialspoint.com/json/json_data_types.htm

[19] Ujević I., (2003), *Sustav za upravljanje dokumentima*, Sveučilište u Zagrebu, Fakultet Elektronike i računarstva, Zagreb, Preuzeto s: http://161.53.18.5/zpr/Portals/0/Osobe/Kreso/Sustav%20za%20upravljanje%20dokumentima.pdf

[20] Horvat I., Sustavi za upravljanje sadržajem, MBU d.o.o, Zagreb

[21] eVision informacijski sustavi, *7 snažnih prednosti korištenja sustava za upravljanje dokumentima*, Preuzeto s: http://www.evision.hr/hr/Novosti/Stranice/7-snaznih-prednosti-koristenja-sustava-za-upravljanje-dokumentima.aspx

[22] Sullivan D., *Types of NoSQL databases and key criteria for choosing them*, TechTarget, Preuzeto s: http://searchdatamanagement.techtarget.com/feature/Key-criteria-for-choosing-different-types-of-NoSQL-databases

[23] MongoDB, *Document Databases*, https://www.mongodb.com/document-databases

Popis slika

Slika 1 Primjer polustrukturiranih podataka nepravilne strukture (JSON format) 4
Slika 2 Baza prikazana u obliku grafa [1] 6
Slika 3 JSON polje [5] 10
Slika 4 JSON objekt [5] 10
Slika 5 Prikaz ključ-vrijednost baze podataka 17
Slika 6 Prikaz dokument baza podataka u usporedbi s relacijskim 19
Slika 7 Prikaz stupčanih baza podataka 24
Slika 8 Prikaz graf baza podataka 24
Slika 9 Node.js arhitektura 28
Slika 10 Express.js *middleware* funkcija 30
Slika 11 Primjer Jade kôda i odgovarajući HTML *output* (Izvor: https://naltatis.github.io/jade-syntax-docs/) 31
Slika 12 Arhitektura DMS sustava 34
Slika 13 Postavljanje MongoDB okruženja 35
Slika 14 Pokretanje MongoDB-a 35
Slika 15 Poruka koju označava da je Node.js server pokrenut 36
Slika 16 Upit traženja korisnika nad kolekcijom users 37
Slika 17 Upit traženja dokumenta u kolekciji documents 38
Slika 18 Package.json datoteka 39
Slika 19 *View* za prijavu korisnika u sustav 42
Slika 20 Login.js - funkcija za provjeru korisnika 43
Slika 21 View za registraciju korisnika u sustav 44
Slika 22 Funkcija findOrCreateUser() 45
Slika 23 Funkcija za kriptiranje lozinke 45
Slika 24 Robomongo - pregled kolekcije *users* 46
Slika 25 Početna stranica nakon ulaska u sustav 47
Slika 26 Pisanje u datoteku list.txt korištenjem aplikacije 47
Slika 27 Modul za kreiranje strukture direktorija 48
Slika 28 Pozivanje funkcije diretoryToObj() i definiranje putanje do direktorija 48
Slika 29 Renderiranje predloška home i prosljeđivanje parametra istom 49
Slika 30 Isječak kôda home.jade datoteke - upravljanje json objektom i kreiranje struktura direktorija i datoteka 49

Slika 31 Lijevo - prikaz strukture direktorija u JSON formatu, Desno - prikaz strukture u datotečnom sustavu 49
Slika 32 *View* documents.jade za dodavanje dokumenata u bazu i na server 50
Slika 33 Metoda koja prima HTTP *body request* s podacima iz forme za *upload* dokumenata i sprema te podatke 51
Slika 34 Model Document 52
Slika 35 Robomongo - kolekcija documents s prikazom dodanog dokumenta putem forme 52
Slika 36 Korištenje modula Multer 52
Slika 37 Pretraživanje dokumenata prema nazivu 53
Slika 38 Pretraživanje dokumenata prema korisniku 53
Slika 39 Funkcija koja pretražuje kolekciju *documents* prema zadanom naslovu 54

Popis tablica

Tablica 1Tipovi brojevi u JSON formatu [18] 8
Tablica 2 Tipovi stringova u JSON formatu [5] 9
Tablica 3 Tipovi podataka korišteni u MongoDB bazi podataka [4] 22
Tablica 4 Usporedba SQL i NoSQL baza podataka (Izvor: https://www.mongodb.com/nosql-explained) 27