Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: **Машинско инжењерство**

Ниво студија: **Мастер академске студије**

Модул: **Информатика у инжењерству**

Предмет: **Пројектовање информационих система и база података**

Број индекса: **343/2013**

Данка Б. Басарић

NoSQL базе података и њихова примена

мастер рад

Комисија за преглед и одбрану: Датум одбране:_____

Оцена:_____

1. **Др Милан Ерић – ментор**
2. _____
3. _____

У оквиру мастер рада са темом "*NoSQL базе података и њихова примена***" кандидат треба да:**

Кроз развој конкретног софтверског решења за web окружење прикаже примену NoSQL база података. Рад треба да обухвати уводна разматрања везана за значај и примену NoSQL модела података, технологије NoSQL база података, упоредну анализу карактерисрике NoSQL и SQL база података, опис конкретног развојног окружења за web апликацију (софтвер) као и опис конкретно развијеног софтвера. На крају дати закључна разматрања и списак коришћене литературе.

1. Rainer K., Turban E.: **Introduction to Information Systems**, John Wiley & Sons, Inc., 2009
2. Лазаревић Б., ……: **Базе података**, ФОН Београд, Београд 2003.
3. Shashank Tiwari: Professional NoSQL, Wrox Press, 2011.
4. Robert T. Mason: **NoSQL Databases and Data Modeling Techniques for a Document-oriented NoSQL Database**, Proceedings of Informing Science & IT Education Conference (InSITE), Jun 29 - Jul 5 2015, Tampa, Florida, United States

Ментор:

Проф. др Милан Ерић

Крагујевац, Септембар 2015

Изјава о самосталној изради рада

Изјављујем да сам овај мастер рад урадила самостално, служећи се стеченим знањем и искуством као и наведеном литературом.

343/2013 Данка Басарић

_____

(потпис)

Резиме

MongoDB база података има велику улогу и значај у решавању многих проблема који се односе на чување велике количине података. Рад представља целокупан осврт на преглед основних појмова нерелационих база података, са акцентом на MongoDB базу података, предности у раду и карактеристика које доводе до једноставности и флексибилности употребе. Кроз рад је изложен значај примене MongoDB базе података и приказане су како теоријске, тако и практичне основе кроз креирање апликације.

Кључне речи: NoSQL база података, MongoDB, складиштење података, обрада података, блог.

Abstract

MongoDB database has a great role and importance in solving many problems related to storing large amounts of data. This paper presents an overview of the basic NoSQL concepts, with an emphasis on MongoDB database and its advantages and characteristics that lead to simplicity and flexibility of use. Furthermore, it is also covered importance of MongoDB in real-world applications and theoretical as well as practical basis through the creation of applications are presented.

Keywords: NoSQL databases, MongoDB, data warehousing, data processing, blog.

САДРЖАЈ

1. Увод

Дуг период многих професионалних каријера у области развоја софтвера, релационе базе података су биле подразумевани избор и професионални приступ проблему чувања података. Питање је у ствари било за коју се базу података, односно за ког произвођача се одлучити. Својим дуготрајним и доказним постојањем постале су важна карика рачунарског размишљања. Међутим, у последњих пар деценија технологија се развија великом брзином, а као последица тога јавио се проблем да релационе базе података нису увек одговарајуће за складиштење и обраду података и у томе лежи мотивација за настанак и развој NoSQL (*Not Only SQL*) база.

Наиме, релационе базе података у свом основном формату нису дизајниране и предвиђене за смештање и управљање подацима временских серија (*time-series*). Само коришћење података временских серија за пословне анализе није нови преокрет, али оно што је ново јесте могућност за прикупљање и анализу огромне количине података временских серија невероватно великом брзином како би се добила најјаснија слика у контексту прегледности и саме обраде.

NoSQL базе података обухватају скуп различитих врста технологија база података који су развијени према захтевима тржишта. Временом се показала потреба за складиштењем веће количине података о корисницима, услугама и производима уз омогућавање већег броја приступа тим подацима и смањење времена обраде истих. Са друге стране, релационе базе података нису дизајниране да се носе са изазовима и потребама прилагодљивости са којима се сусрећемо код развоја данашњих модерних апликација. Такође, релационе базе података нису замишљене да искоришћавају предности данашњег приступа великој процесорској снази и великој количини меморије за складиштење података.

Сам термин NoSQL уведен је 1998. године и односио се на релационе базе података које нису имале подршку SQL-а. Увео га је Carlo Strozzi и тако назвао свој пројекат који је користио базу која није подржавала стандардни SQL интерфејс. Термин NoSQL поново уводи Eric Evans 2009. године када је организовано окупљање ради дискусије о појави све већег броја складишта података која не воде рачуна о очувању атомичности, конзистентности, изолацији и трајности података (тзв. *ACID* особине). NoSQL, дакле, не представља један производ или технологију већ класу производа и колекцију различитих, понекад повезаних концепата о начинима складиштења и манипулације подацима.

Важни аспекти везани за домен NoSQL база података су још увек у фази турбуленције, мењају се у времену нудећи нова решења, нове могућности и нове базе података. Поред неоспорних чињеница предности NoSQL база, сама идеја и концепт NoSQL заслужује пажњу, јер нуди нешто ново и настаје из реалних потреба за новим начином примања и представљања модела података. NoSQL базе података могу се посматрати као потенцијално решење за уклањање проблема традиционалног начина пројектовања и коришћења релационих база података. Чињеница је да данашњи вид пословања црпи снагу из података насталих у све краћим и фреквентнијим временским

интервалима, што неминовно води ка великим количинама података. Концепт NoSQL база података настао је из потребе да се, између осталог, на адекватан начин задовоље захтеви обраде велике количине података.

У оквиру овог рада главни акценат стављен је на значај и примену MongoDB базе података. На самом почетку рада дат је увод о NoSQL базама података, при чему је каратко описан сваки од четири постојећа типа, односно: базе података засноване на графу, базе података кључ – вредност, базе засноване на матрицама и базе података засноване на документима чијој групи припада и сама MongoDB база података. У оквиру истог поглавља представљено је поређење релационих и нерелационих база података истицањем различитости у раду конкретно на примерима, затим су дате предности NoSQL база података, као и разлози њихове чешће примене код објектно оријентисаног програмирања због саме флексибилности и једноставности употребе.

У даљем наставку рада, у оквиру поглавља број три, реч је конкретно о MongoDB бази података, при чему су обухваћене основне карактеристике, структура, као и приступ подацима. Представљене су могућности њихове примене у различитим сферама, истицањем њихових предности и приказом лакшег начина рада. Такође су наведене позивне функције које су различите од функција које се користе у оквиру релационих база података и термини који се користе.

Након представљања концепта MongoDB базе података, у поглављу четири реч је о креирању личног блога коришћењем MongoDB базе података и Python web технологија, што представља практичан део самог рада. Детаљно је објашњен начин креирања апликације, почевши од дефиниције блога, предности MVC (енг. *Model – View – Controller*) архитектуре, инсталирања свега што је потребно за рад и јасног представљања кода писаног у сврху креирања личног блога кога корисници редовно могу посећивати, остављајући своја мишљења путем коментара, а који може бити ажуриран од стране једног корисника, односно администратора блога.

У оквиру закључка дат је осврт на комплетан рад, као и лично виђење предности у раду са MongoDB базом података у комбинацији са различитим веб технологијама.

2. NoSQL базе података

NoSQL обухвата широк спектар различитих технологија база података који је настао и развио се као одговор на значајан пораст обима података из разних типова извора, фреквенција којима се приступа овим подацима, као и захтева перформанси и потребних обрада.

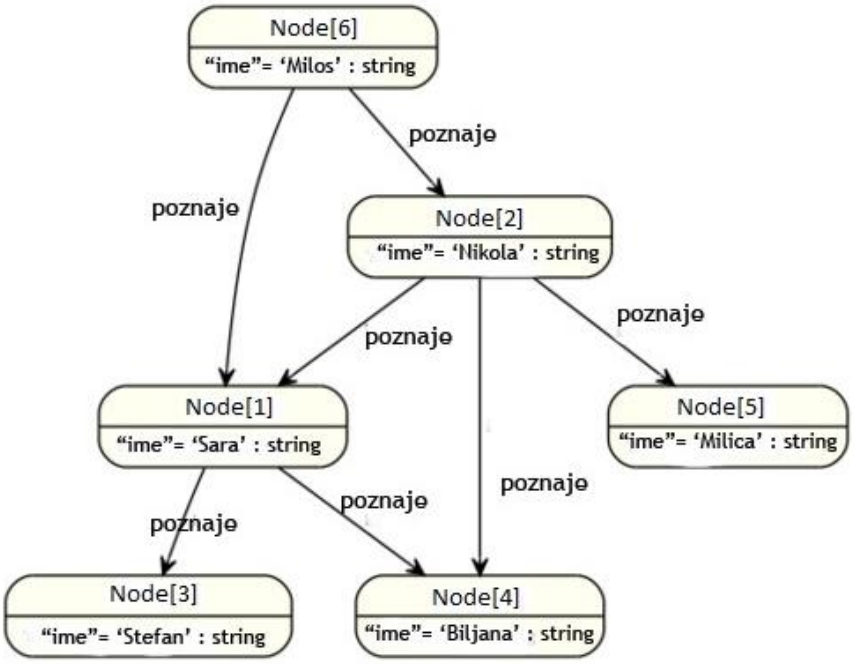
2. 1. Основни типови NoSQL база података:

- **Базе података засноване на графу** (енг. *graph databases*) се најчешће користе за чување података о мрежама, као што су на пример подаци о друштвеним мрежама. Најпознатији представници ове категорије база података су Neo4J, Titan и OrientDB.
- **Базе података кључ – вредност** (енг. *key – value databases*) су најједноставнија врста NoSQL база података. Сваки запис у бази чува се као име атрибута (тј. кључ) заједно са својом вредношћу. Неки од познатих представника ове групе су Redis, Riak и Memcached. Већина ових база имају веома мали број могућих функционалности које нуде, али неке од њих, попут Redis-a, омогућавају додељивање типа података вредностима (нпр. *"integer"*) чиме проширују додатне сет функционалности.
- **Базе података засноване на матрицама** (енг. *wide – column databases*), су заправо кључ – вредност базе података које за проналазак података користе скуп кључева. На тај начин групишу се подаци у један ред, што аутоматски подразумева да подаци кад се спреме на диск буду физички хомогени. Овај тип база података најчешће се користи за велике интернет услуге, као и за само претраживање интернета. Најзначајнији представници ове групе база података су Cassandra и Hbase.
- **Базе података засноване на документима** (енг. *document databases*) упарују сваки кључ са комплексном структуром података која се назива документ. Документи могу садржати више различитих парова кључ – вредности или чак могу имати друге угњеждене документе (кључ – документ). Најзначајнији представници ове категорије база података су MongoDB, CouchDB и Couchbase.

2.1.1. Базе података засноване на графу

Граф базе података користе структуру података типа *graf* (чворови, потези и својства) за репрезентацију и складиштење података. Код овог типа базе података сваки елемент може да садржи директан показивач ка другим елементима без потребе постојања индексне структуре, јер сваки чвор има везе са суседним чворовима. Граф база података се најчешће користи за упите који се односе на релације између података, док су за остале врсте анализа мање употребљиве. Најочигледнији пример примене граф база података јесу везе између људи на друштвеним, односно социјалним мрежама.

Елементи граф базе података су, дакле: својства, потези и чворови. Сваки чвор у оквиру базе може да има неограничен број својстава. Својства представљају податке који описују чворове и могу се изједначити са атрибуима, док потези представљају релације које повезују чворове међусобно, или чворове са својствима. На слици 1 приказан је пример граф базе података [9].



Слика 1. Пример граф базе података [9]

Граф база података може садржати већи број чворова, док најједноставнији вид ове врсте базе може имати само један чвор. Поред генералних граф база података које могу да складиште било који граф, постоје и специјализоване граф базе података као што су *triplestore* (складиштење података у облику субјекат – предикат – објекат, RDF) и мрежне базе података чији се принцип рада заснива на употреби мрежног модела [9].

Разлике између граф базе података и релационе базе података:

- Релационе базе података користе мање простора, јер не морају да памте све везе које памте граф базе.
- Релационе базе података су брже код упита при огромним количинама података јер не морају испитати сваки запис током упита.

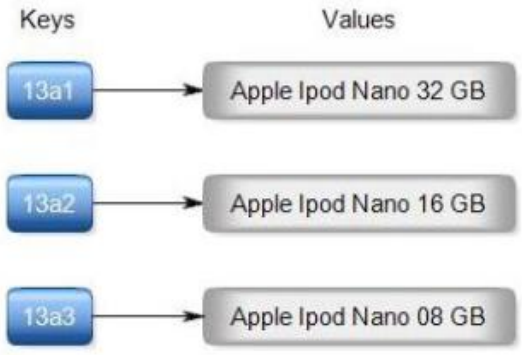
- Граф базе података су погодне за нередовне и сложене структуре, док су релационе базе погодне за редовне, релативно једноставне структуре које се већином и захтевају у реалној употреби, па се из тог разлога релационе базе података и више користе [9].

Међутим, граф базе података са собом носе и одређени број својих предности међу којима су и:

- Решење за асоцијативне скупове података,
- Директно мапирање из граф базе података на ОО (објектно - оријентисан) модел,
- Погодне су за операцијије типа: обилазак графа, провере да ли стоји пут између два чвора, одрђивање најкраћег пута између два чвора и слично.
- Погодне су за развој *ad – hoc* апликације које брзо еваулирају због своје флексибилне шеме.
- Примењују се и због велике количине података које не захтевају скупе операције споја [9].

2.1.2. Базе података кључ – вредност

База података кључ – вредност је заправо основни тип NoSQL базе података. Сваки запис у оквиру базе је сачуван као име атрибута, односно кључ са придруженом вредношћу. Основном предности овог типа база података сматра се врло брзо записивање и читање података без обзира на количину, док је главни недостатак то што су могућности претраге врло ограничене, тј. претрага се врши само по кључу. База података кључ – вредност своју примену највише налази за чување велике количине података код различитих видова имплементација. Пример типичне кључ – вредност базе података приказан је на слици 2 [2].



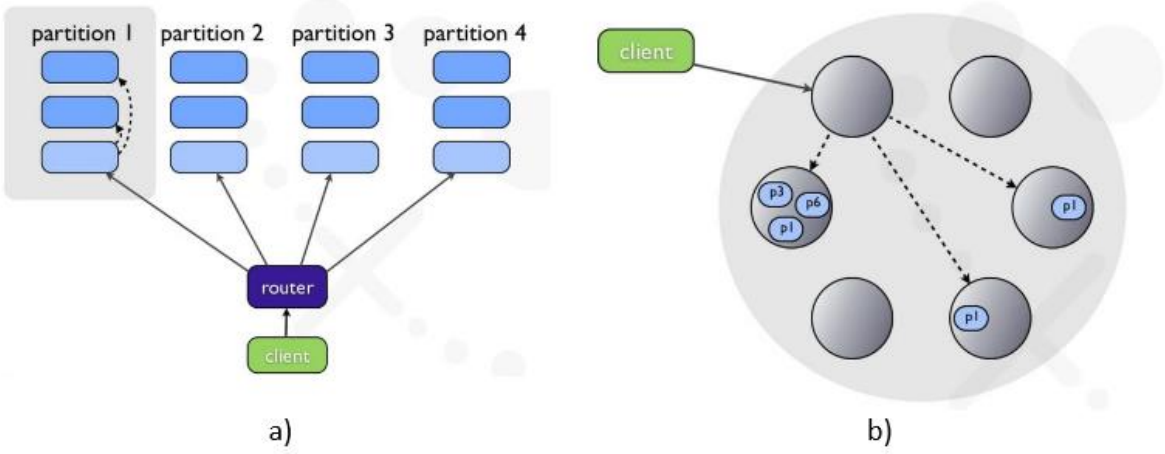
Слика 2. Изглед базе кључ – вредност [2]

Један од најчешћих случајева коришћења овог типа базе података јесте за чување података о корисничким сесијама [8].

2.1.3. Базе података засноване на матрицама

Базе података засноване на матрицама своју примену налазе за чување података дистрибуиране мулти – димензионалне сортиране мапе. Колоне могу бити угњеждене унутар других колона и такве колоне се називају „*super*“ колонама. За проналазак података се користи скуп кључева (нпр. врста, колона) на начин да се повезани подаци групишу у једну врсту. Код овог типа база података није потребно унапред одредити колико се колона налази у реду и подаци се претражују по кључу који једнозначно одређује ред. Највећа предност базе података засноване на матрицама је у томе што омогућују дистрибуираност уз непостојеће јединствене тачке испада, па имају линеарну скалабилност [2].

Најзначајнији представници овог типа база података су: Cassadra, HBase, Accumulo, Hypertable и Sqrrl.



Слика 3. а) Састав са јединственом тачком испада, б) Састав без јединствене тачке испада [2]

2.1.4. Базе података засноване на документима

Такозване базе података за чување докумената (енг. *document databases*) чувају податке у облику докумената, за разлику од релационих база података које податке чувају у облику врста и колона унутар табела. Документи који се чувају у базама података најчешће су структуирани помоћу JSON (*JavaScript Object Notation*) формата, који је врло популаран у последње време. Такав облик података је доста интуитиван и једноставан начин за моделирање података, а такође је значајно и то што се може користити и у оквиру објектно оријентисаног програмирања, где би сваки документ представљао један објекат [3].

Сваки документ се може састојати из једног или више атрибута, чије се вредности могу представити *string*-ом, *boolin*-ом или низом. Оно што је најважније јесте то што се сви подаци везани за документ најчешће чувају управо на том документу, чиме се обезбеђује смањење времена потребног за приступ подацима и готово у потпуности избацује потреба за комплексим спајањем табела. Међутим, јавља се питање ажурирања података, али се управо из тог разлога ентитети који су подложни променама ипак не

чувају директно на документу, већ се чувају на посебним местима. Када је у питању шема, може се слободно рећи да у оваквим базама података шеме заправо и нема, јер сваки документ може садржати различите атрибуте, чиме се постиже флексибилност, а олакшава моделирање полуструктуираних и полиморфних типова података. Такође, овим начином се значајно олакшавају будуће измене апликација у току развоја, јер се додавање атрибута документу може извести на крајње једноставан начин.

Чување података у оваквом облику има неколико предности [3]:

- Документи представљају независне јединице, а омогућују боље перформансе и дистрибуцију података на више сервера, чувајући их локално.
- Апликацијска логика је једноставнија за програмирање. Не морају се радити промене између објеката из базе података и упита, већ се објекат директно претвара у документ.
- Неструктуирани подаци се врло лако чувају. С обзиром да документ може садржати све што апликацијска логика дозвољава, а нема строго дефинисане шеме, неструктуиране документе је врло лако чувати.

Поред чувања, код оваквих типова база података значајна је чињеница да је и претраживање база такође врло једноставно. Подаци се могу претраживати према било ком атрибуту унутар документа. Управо због тога, базе података засноване на документима су врло погодне за управљање документима, где је потребно да се цели документи чувају унутар базе података. Такође, могуће их је применити за широк спектар апликација због саме флексибилности модела података као и могућности претраживања докумената по било ком атрибуту. Mongo база података чува документе у формату који је врло сличан *JSON (JavaScript Object Notatio*n), што омогућује лако чување података. Такође, састав одговара модерним објектно – оријентисаним методологијама и MongoDB подржава индексирање као и комплексне упитне механизме, за разлику од других који их не подржавају или је потребно користити додатне модуле за укључивање комплетног претраживања .

Базе података засноване на документима пружају могућност постављања упита на било ком атрибуту унутар документа. Неке базе података, као што је MongoDB пружају богат скуп операција за индексирање, тако да је могуће извршити различите упите над базом.

Подржани индекси су:

- *Compound indexes* – користи се где појединачна индекс структура има референце на више атрибута у скупу докумената;
- *Sparse indexes* – проверава да ли документ садржи поједини атрибут;
- *Time to live (TTL) indexes* – овим индексом се омогућава да се документ аутоматски избрише из колекције након одређеног периода;
- *Unique indexes* – омогућава брисање свих докумената који садрже атрибуте који имају више вредности за дефинисани индекс атрибут;
- *Test indexes* – подржава претраживање текста над стринг вредношћу у документу колекције;
- *Geospatial indexes* – могућава геопросторне индексе (у зависности од локације) итд.

Дакле, базе података засноване на документима су намењене за општу намену и широку примену код апликација, управо због своје флексибилности модела података, односно неструктуираности модела, способности генерисања упита над свим атрибутима у документу и подршке мапирања објеката модерно објектно оријентисаних језика у документе, тј. у документ модел података.

Најпопуларније базе података овог типа јесу: MongoDB, CouchDB, Couchbase, MarkLogic и RavenDB.

Слика 4. Документ у NoSQL бази података у односу на SQL базу података [2]

2.2. Поређење NoSQL-а у односу на SQL базе података

Када се упоређују могућности и карактеристике релационих база и NoSQL-а база података по перформансама и функционалностима, може се рећи да код развоја модерних апликација NoSQL базе података пружају бољу скалабилност и боље перформансе, па њихове структуре података решавају неке од проблема које структура релационих база података не решава, као што је:

- Могућност обраде велике количине података која може бити структуирана, делимично структуирана, или неструктуирана;
- Могућност брже интеракције, аналогни развој и честе измене кода;
- Флексибилност и једноставност саме употребе код објектно оријентисаног програмирања;
- Функционална и скалабилна архитектура уместо скупе и затворене архитектуре [2].

Дефинисање шеме

Као што је познато, релационе базе података захтевају дефинисање шеме базе података. Сваки пут када се уоче неке нове особине реалног система, шема базе података захтева промену. На пример, ако је потребно да се купцу прошири постојећи скуп атрибута, односно да се додају нове ставке (као нпр. врста робе, боја), поред уобичајених (идентфикационих података адресе, телефона итд.) неопходно је пре свега додати нову колону, а затим преместити целу базу података на новонаправљену шему. Уколико база

података садржи велики број информација, читав процес измене шеме може трајати дуго и довести до значајног застоја у раду. Поред тога, у случају коришћења релационих база података користе и неструктуирани подаци, или подаци чија структура није унапред позната.

Са друге стране, NoSQL базе података су креиране тако да дозвољавају уметање података без унапред дефинисане шеме и на тај начин је значајно олакшана реализација промене апликације у реалном времену, без бриге о ситуацијама прекида, што доводи до закључка да је развој бржи, код интеграције је поузданији и нема времена одвојеног за администратора базе података [1].

Дистрибуирано смештање података

Због начина на који су структуиране релационе базе података, када је у питању хардверска подршка, уобичајена је вертикална скалабилност, односно обично се захтева да један сервер буде домаћин целе базе података да би се осигурала поузданост и континуирана доступност бази података. На повећане потребе се углавном одговара доградњом серверских капацитета (уградња бржег процесора и већих хард дискова) што временом постаје скупо. Решење, односно солуција се налази у хоризонталној скалабилности, односно додавању нових сервера, уместо повећања капацитета једног сервера.

Дистрибуирано смештање података, или како је уобичајени термин у NoSQL базама, такозвани *sharing* базе података на више инстанци сервера може бити постигнуто SQL базама података, али се у пракси обично остварује у оквиру мрежа за спремање података (*SAN, Storage Area Network*) или другим комплексним аранжманима који омогућују да се сложен хардвер понаша као један сервер. Будући да релационе базе података не обезбеђују такву подршку, развојни тимови су задужени за имплементацију више релационих база података на одређени број машина. Подаци се чувају у свакој инстанци базе података аутономно. Апликациони код се развија да дистрибуира податке, дистрибуира упите, али и да обједињује резултантне податаке преко свих инстанци базе података. Поред тога, потребно је развити код за обраду грешака ресурса, за релацију спајања (*join*) широм различитих база података, за ребалансирање, репликацију и друге захтеве. Такође, многе предности у релационим базама података, као што је трансакцијски интегритет, су угрожени или се елиминишу у случају ручног партиционисања [1].

Ипак, NoSQL базе података са друге стране обично подржавају аутоматско партиционисање што значи да аутоматски шире податке преко произвољног броја сервера без потребе да апликација води рачуна о композицији серверског скупа (*server pool*). Подаци и учитавање упита су аутоматизовани широм сервера, а у случају пада сервера, он може бити замењен без прекида апликације.

Поред тога, „рачунарство у облаку“ (*cloud computing*) је знатно лакше, а услуге попут *Amazon Web Service*- а пружају виртуелан, готово неограничен капацитет на захтев и при томе воде бригу о свим неопходним пословима администрације базе података [1].

Многе NoSQL базе података такође подржавају аутоматску репликацију што у коначном исходу даје високу доступност без потребе увођења нових посебних апликација за управљање задацима тог типа. Окружење за смештање података значајно

је виртуелизовано из перспективе развојног програмера и та карактеристика представља још једну у низу предности NoSQL база података, чиме се омогућава лакша импровизација, имплементирање и сама примена различитих апликација.

У следећој табели дато је поређење SQL и NoSQL база података, укључујући типове података, модел чувања података, историју развоја итд.

	SQL базе података	**NoSQL базе података**
Типови (врсте)		
	Само један тип података са мањим разликама (SQL базе података).	Многе различите варијанте NoSQL база података, а главне категорије су: 1. *key value stores,* 2. *document databases,* *3. wide – column stores* 4. *graph databases*.
Историја развоја		
	Развијене 1970-тих година за апликације и чување података.	Развијене 2000-тих година да се носе са ограничењима која имају SQL базе података. Фокус им је на скалабилности, репликацији и неструктуираном чувању података.
Примери		
	MySQL, Postgres, Oracle Database, SQL Server	MongoDB, Cassandra, Hbase, Neo4j
Модел чувања података		
	Сваки појединиз запис се чува као ред у табели (нпр. особа, запослен). Свака колона у таквом реду представља одређени део податка о запису (нпр. име, презиме, врста посла). Различите врсте података се чувају у одвојеним табелама (нпр. подаци о особи у табели Особа, подаци о раднику у табели Радник), а затим се код сложенијих упита такве табеле спајају *JOIN* упитима како би се добиле све потребне информације.	Модел чувања података се разлукује у зависности од врсте NoSQL базе података. Нпр. тип базе података је *key value*, који функционише на сличан начин као SQL базе података,али је разлика у томе што постоје само две колоне („кључ“и„вредност“). Понекад колона „вредност“ може садржати сложеније типове. *Document model* тип базе података спрема све податке заједно у један документ који може бити *JSON, XML* или неки други формат.

	SQL базе података	**NoSQL базе података**
Шема		
	Структура и врсте података су већ унапред дефинисани и имају структуирани тип записа.	Структура и врста података су динамичке и имају неструктуирани тип записа. Нови атрибути се једноставно могу додавати како се појављују и могу се чувати по потреби.
Врста скалирања		
	Вертикално скалирање, што значи да се мора додавати снага (*RAM, CPU, HDD*) како би се носио са порастом захтева. Могуће је проширити SQL базе података на више сервера, али је то врло захтеван и скуп изазов.	Хоризонтално скалирање тј. додавање додатних чворова. Администратор базе података може на једноставан начин да дода више инстанци сервера, а база података ће аутоматски ширити податке по потреби.
Модел развоја		
	Модели развоја су различити, у зависности од администратора и потребе базе података, али углавном је то open – source (нпр. *MySQL*) или чак closed – source (нпр. *Oracle Database*).	*Open - source*
Подржаност трансакција		
	Операције трансакција су подржане на већини база података.	Операције трансакција су подржане само у специфичним случајевима (на већини докумената, а не на већини база података).
Манипулација подацима		
	Манипулација подацима се одвија коришћењем специфичног SQL језика (*select, insert, update*).	Манипулација подацима се одвија кроз апликацијеске програме (*API*).

Табела 1. Поређење SQL и NoSQL база података

3. MongoDB

Самим порастом интересовања за алтернативним системима за управљање базама података, који се разликују од традиционалних база података, појавио се концепт такозваних NoSQL база података, које не користе SQL за повезивање, нерелационе су, дистрибуиране, отвореног кода и хоризонтално скалабилне. MongoDB представља водећу NoSQL базу података развијену октобра 2007, године од стране компаније „10gen“.

Слика 5. MongoDB лого

MongoDB чува податке углавном као JSON (*JavaScript Object Notation*) документе са динамичким шемама. JSON је отворени стандард заснован на тексту, а осмишљен конкретно за размену података који су људима погодни за читање. MongoDB је прихваћен као *backend* софтвер бројних значајних веб сајтова и сервиса укључујући *New York Times, eBay, Foursquare* итд, управо из разлога што интеграцију података у многим апликацијама чини једноставнијом и бржом.

Чињеница да MongoDB има званичне драјвере за популарне програмске језике и развојна окружења само је једна у низу многобројних предности који доприносе самој његовој популарности и изузетно широкој примени. Такође, постоје и многи незванични драјвери који су подржани од стране отворених заједница, а који подржавају готово све програмске језике и различита окружења.

Оно што је такође изузетно важно када је реч о MongoDB јесте чињеница да су сви подаци везани на документ чиме се увелико смањује време приступа подацима и готово у потпуности избацује потреба за комплексним спајањем табела. У овом случају, потенцијално спорно питање је питање ажурирања података, међутим, најчешће се ентитети који су подложни променама ипак не чувају директно на документ, већ се чувају засебно, а сам документ има спремљену асоцијацију на тај ентитет [8].

Имајући у виду његове карактеристике и могућности, MongoDB своју примену налази у:

- Апликацијама за управљање садржајем (енг. *CMS – Content Management Systems*), с обзиром да није потребно унапред одредити шему података поједине збирке докумената,
- Записивању дневника рада апликација (енг. *Application Data Logs*),
- *Online* рачунарским игрицама код којих је потребан велики број записивања и читања од стране много корисника, а који са друге стране, не смеју утицати на брзину извођења,

- Развоју апликација јер није потребно унапред дефинисати шему базе података, што истовремено олакшава увођење нових атрибута и врста докумената током развоја апликација,
- Развоју мобилних апликација због управљања геопросторним подацима (*Google Maps*) [5].

3.1. Спецификације и карактеристике MongoDB-а

MongoDB спада у већ поменуту групу база података које су зановане на документима (*document databases*), а које складиште податке у облику докумената, за разлику од релационих база података које податке чувају унутар табела. У табели 2 су представљени упоредни називи терминологија релационих и NoSQL база података.

Колекција представља групу докумената који имају нека заједничка својства и слободно се може рећи да одговара термину табеле у концепту релационих база података [8].

Релационе базе података	**MongoDB**
Табела	Колекција (*Collection*)
Ред	*JSON* документ
Индекс	Индекс
JOIN	Референца или уграђени документ

Табела 2. Упоредни називи релационих база података и MongoDB

Код MongoDB-а, за разлику од SQL база података, шема је флексибилна, односно колекција не захтева тачну структуру докумената који се у њу смештају, али битно је нагласити да документи који се у њој чувају, односно документи који се чувају у оквиру исте колекције морају имати сличну структуру јер представљају сличне ентитете. Таквом флексибилношћу се знатно олакшава мапирање докумената у неки ентитет или објекат [4].

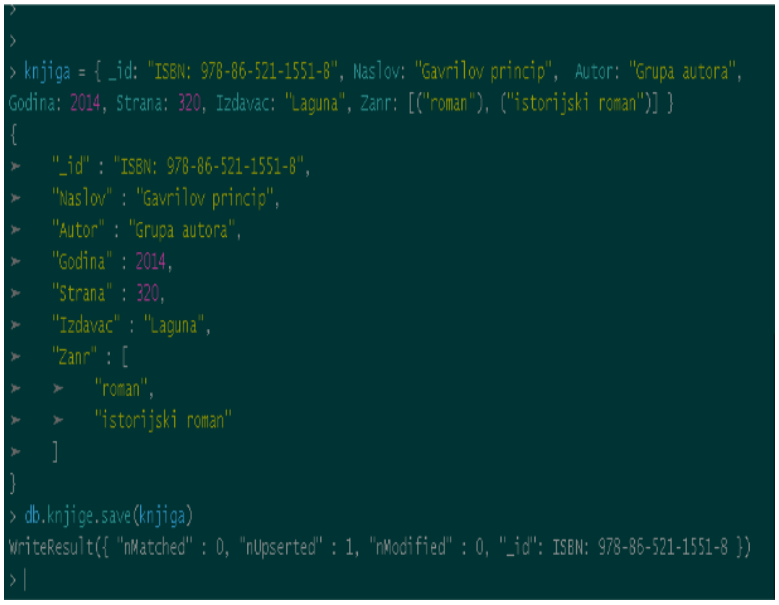
Када је у питању само дизајнирање модела података важна одлука јесте избор начина на који ће документи бити повезани једни са другима. Постоје два начина повезивања и то су референце и уграђени документи. Референце чувају однос између докумената коришћењем линкова или референци из једног документа у други (тзв. нормализовани модел података), док се уграђивањем докумената везе између података спремају унутар једног документа.

Као једноставан пример тога узет је случај неке књижаре у којој се за сваку књигу прате одређени подаци. У табели 3 дат је приказ једног од начина представљања:

```
{
_id: “ISBN: 978-86-521-1551-8”,
Naslov: “Gavrilov princip”,
Autor: “Grupa autora”,
Godina: 2014,
Strana: 320,
Izdavac: “Laguna”,
Zanr: [(“roman”), (“istorijski roman”)]
}
```

Табела 3. Приказ кода записа „Књига“ [6]

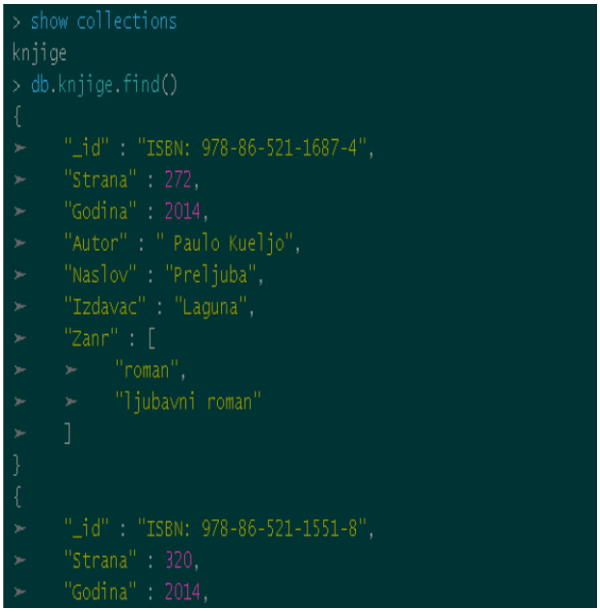
док је на слици 7 илустративно приказан начин креирања документа на примеру документа „књига“ и његово смештање у колекцију названу „књиге“, коришћењем MongoDB shell-a (интерактивни *JavaScript shell* интерфејс за MongoDB).



Слика 6. Приказ креирања документа „књига“ и његово смештање у колекцију „књиге“ коришћењем MongoDB shell-a [4]

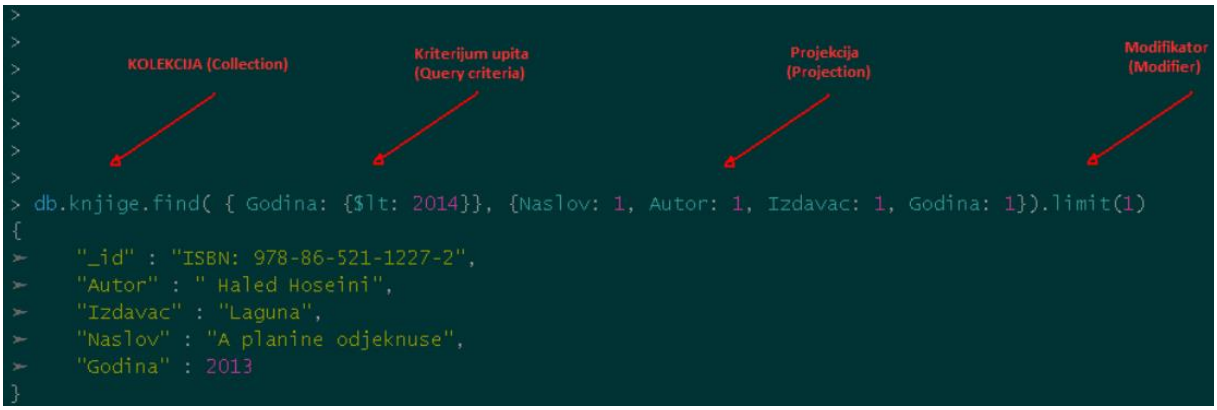
Сваки MongoDB документ је аутоматизован тако да добија јединствени идентификатор (*ID*), али такође постоји и могућност самосталног креирања истог од стране администратора базе података. У овом случају, као што се може видети из табеле 3 и са слике број 6, вредности ID је директно придружен међународни стандардни број *ISBN*, као „природан“ идентификатор сваке књиге [4].

Користећи команду *db.knjige.find()* врши се упит над колекцијом „књиге“, као што је приказано на слици 7, док команда *show collections* враћа називе колекција.



Слика 7. Упит над колекцијом „књиге“ коришћењем MongoDB shell-a [4]

Предходни упит би одговарао упиту „*select * from knjige*“ код релационих база података, који враћа све редове табеле „књиге“. Пример реализације комплексних упита који садрже критеријуме упита, пројекције и/или модификаторе приказан је на слици 8. У овом примеру потребно је наћи једну књигу *(limit(1))* која је издата пре 2014. године *($lt:2014),* при чему су тражени атрибути: Наслов, Аутор, Издавач, Година (Naslov: 1, Autor: 1, Izdavac: 1, Godina: 1).



Слика 8. Приказ примера упита са задатим критеријумом упита [4]

Дакле, MongoDB представља далеко бољи избор у односу на релационе базе података, јер нуди више могућности и једноставнији начин рада.

Неке од основних карактеристика MongoDB-а су:

- Смештање података је усмерено на документима – *JSON* документа са динамичким шемама нуде једноставност и снагу,
- Апсолутна подршка индексирању – индексирање било ког атрибута,
- Репликација и висока доступност,
- Аутоматско скалирање – хоризонтално скалирање без угрожавања функционалности. Хоризонтално скалирање дистрибуира један логички систем базе података између скупа машина.
- Упити – моћни упити базирани на документима,
- Брзи *update,*
- Мапирање/Редуковање – флексибилна агрегација и обрада података,
- *GridFS* – чува фајлове било које величине без компликација. Уместо чувања фајла у појединачном документу, *GridFS* дели фајл на делове, или блокове и сваки од тих докумената чува као посебан документ.
- *Online shell* пружа могућност испробавања MongoDB-а без инсталације истог,
- *Ad hoc* упити – MongoDB подржава претрагу по пољу, упите по опсегу и претраге по регуларним изразима. Упити могу да врате одређена поља документа, као и да обухвате кориснички дефинисане *JavaScript* функције.
- Извршавање *JavaScript* кода на страни сервера – *JavaScript* се може користити у упитима, агрегатним функцијама и директно се упутити бази података за извршавање [10].

3.2. Структура података

Сваки документ се састоји од једног или више атрибута чија вредност одговара типу података *JSON* формата (број, string, boolean или низ). MongoDB складишти документе и у *BSON(Binary Script Object Notation)* формату који је врло сличан *JSON* формату. Заправо, *BSON* формат је бинарни облик *JSON* формата [4].

JSON је формат записивања података у облику *JavaScript* синтаксе који је развијен са циљем да се лакше чита и да му структура заузима минимум меморијског простора. Дакле, основна јединица података су *JSON* – документи који се чувају у збиркама (енг. *collections*).

Када се MongoDB упореди са релационим базама података, документи представљају записе, односно *n*-торке (енг. *records*), док збирке предтављају релације. У релационим базама података сви записи исте релације имају исти број атрибута, док је код MongoDB докумената ситуација таква да сваки документ може имати своја поља невезана уз структуру збирке у којој се налази.

Као пример, може се навести то да свака особа не мора имати унапред предодређене информације о рођењу, што би код релационе базе података захтевало додавање новог атрибута у шему релације кад год би се појавио нови атрибут. У табели 4 дат је пример релационе базе података.

ID	Име	Презиме	Година рођења	Име мајке	Место рођења
1	Петар	Петровић	1963	Ана	Крагујевац
2	Марко	Марковић	1987	Јана	Београд

Табела 4. Пример релационе базе података

Исти пример имао би следећи запис у оквиру MongoDB документа:

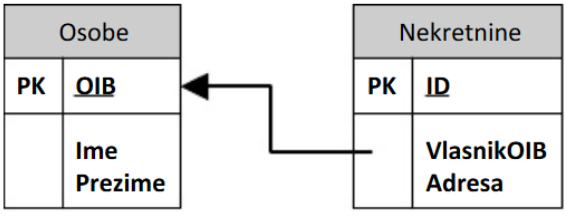
```
{
“_id” : ObjectID (“4efa8da223494afht47”),
“Ime” : ”Petar”,
“Prezime” : “Petrovic”,
“Godina rodjenja” : “1963”,
“Ime majke” : “Ana” ,
“Mesto rodjenja” : “Kragujevac”
},
{
“_id” : ObjectID (“4efa8da223494afht48”),
“Ime” : ”Marko”,
“Prezime” : “Markovic”,
“Godina rodjenja” : “1987”,
“Ime majke” : “Jana” ,
“Mesto rodjenja” : “Beograd”
}
```

Табела 5. MongoDB документ

Атрибут *“_id”* је обавезан и аутоматски га додаје MongoDB као кључ који једнозначно одређује сваки документ унутар комплетне базе података. Такође је могуће користити и додатни (властити) кључ над којим се изгради јединствени индекс (енг. *unique index*) [5].

3.3. Приступ подацима

Код релационих база података подаци су сачувани у нормализованом облику. На пример, када је одређеним особама потребно придружити некретнине које поседују, изглед релационе шеме Osobe – Nekretnine би изгледао као на слици 9.



Слика 9. Релациона шема Osobe – Nekretnine

Да би се ове две релације спојиле и да би се добио попис некретнина са припадајућим власницима, код релационих база података користио би се следећи SQL упит:

SELECT osobe.*, nekretnine.Adresa

FROM osobe **LEFT JOIN** nekretnine **ON** nekretnine.VlasnikOIB = osobe.OIB

Резултат предходног упита изгледао би на следећи начин:

OIB	Ime	Prezime	Adresa
214214214214	Petar	Petrović	Kragujevac 34000

Табела 6. Пример релационе базе података

За разлику од релационих база података, MongoDB не подржава спајање у упитима, па се управо из тог разлога користе друга алтернативна решења, као на пример:

а) када је потребно имати две збирке – колекције, у једну од њих сачувани Osobe, а у другу Nekretnine, која се повезује са особом преко *"_id"* кључа или OIB-а, што је слично као код релационих база података, али „спајање" података је потребно извести у следећем апликацијском облику:

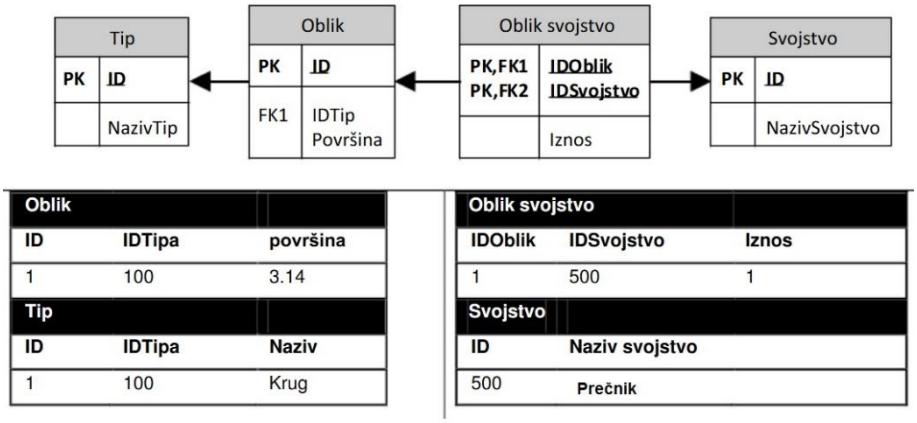
```
za svaku nekretninu {
vlasnik: = osobe.find( { OIB: nekretnina.VlasnikOIB } )
}
```

б) када је потребно унутар документа Osoba, направити поддокумент Nekretnina, без потребе за још једном збирком. Притом, треба имати на уму ограничење од 16Mb колико максимално може износити величина једног документа.

```
{
"_id" : ObjectID("424377251eft174"),
"OIB" : "214214214214" ,
„Ime" : Petar ,
„Prezime" : Petrovic ,
„Nekretnine" :
{
„Adresa" : „Kragujevac 34000" ,
}
        }
```

3.3.1. Скупови хетерогених ентитета

Уколико постоји потреба да се, на пример, у релационој бази података сачувају одређени геометријски облици и њихове физичке величине, то се може учинити тако да се заједнича својства ставе у једну релацију (Oblik), а да се остала својства сачувају у посебној релацији (Oblik – svojstvo), што би имало следећи изглед [5]:



Слика 10. Пример релационе базе података

Исти пример далеко је једноставније решити у оквиру MongoDB-a, у коме ће се такође чувати тип геометријских облика, али није потребно унапед предвидети и дефинисати шему за типове истих.

MongoDB

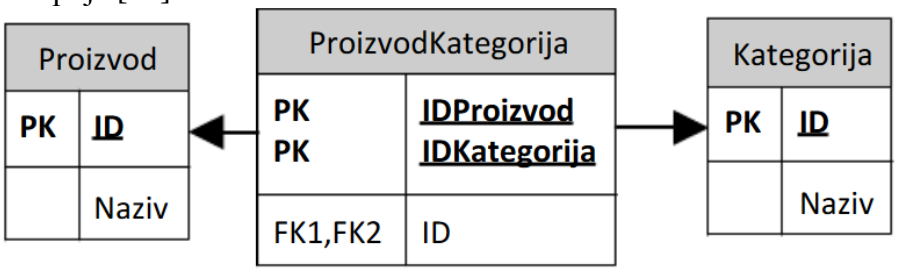
Геометријски облици

{
 “_id” : ObjectID(“424377251eft174”),
 “tip” : “krug”,
 “povrsina” : 3.14,
 “precnik” : 1,
 }, {
 “_id” : ObjectID(“424377251eft177”),
 “tip” : “kvadrat”,
 “povrsina” : 100,
 “precnik” : 10,
 }

Табела 7. Пример MongoDB-a

3.3.2. N – N везе

N – N (енг. *many to many*) везе или односи се у релационим базама података углавном изводе користећи три релације, на пример код повезивања производа и категорија (сваки производ може бити у N категорија, а свака категорија има N производа) користи се трећа релација ProizvodKategorija у којој се чува кључ производа и кључ категорије [11] :



Слика 11. Пример релационе базе података

Међутим, код MongoDB-а се у истом случају могу користити само две збирке, Proizvod и Kategorija, повезујући их тако да се унутар сваког производа чувају кључеви категорија у које спада одабрани производ.

```
Proizvodi:
{
_id: ObjectID (“1112474112dddas411”),
naziv: “MongoDB knjiga”,
kategorije: [ ObjectID(“123123228d2284dad221214a”),
                        ObjectID(“030477ejnfffksnjdx41152”)

}
Kategorije:
 {
_id: ObjectID (“123123228d2284dad221214a”),
naziv: “Baze podataka”
},
{
_id: ObjectID (“030477ejnfffksnjdx41152”),
naziv: “Mongo Baze podataka”
}
```

Табела 8. Пример MongoDB-а

3.3.3. MapReduce функције

MapReduce представља програмски модел који се користи за обраду великих количина података код подељених састава (енг. *distributed systems*). Модел је добио име по функцијама које се углавном употребљавају у функцијским језицима, иако се оне разликују од MapReduce функција које се користе у MongoDB и другим NoSQL базама података [5].

Трансформација података се одвија у следећа два корака:

1. *Map* – подразумева читање података, њихово филтрирање и груписање у парове кључ – вредност. *Map* функција се приликом рада са већим бројем чворова може извршавати на сваком чвору посебно.
2. *Reduce* – подразумева обраду удружених кључ – вредност података из *Map* функције (свих чворова) и генерисање излаза у облику парова кључ – вредност.

MapReduce функција која се користи у оквиру MongoDB-а има једноставан начин приступа и употребе, који је креиран тако да самим корисницима олакша рад и смањи потребно време за обављање активности.

3.3.4. Поређење SQL језика и рада са MongoDB JSON упитима

Код једноставних операција база података, попут претраживања, уноса и брисања, SQL упити се врло лако пресликавају у MongoDB JSON упите [5]. Преглед SQL и MongoDB JSON упита представљен је у табели 9.

SQL упит	MongoDB JSON упит
CREATE TABLE...	Стварање нових збирки није потребно
ALTER TABLE...	Сваки од докумената могуће је изменити без измене шеме.
INSERT INTO osobe („Petar“, „Petrović“)	db.osobe.insert({ ime: “Petar”, prezime: “Petrovic” });
SELECT * FROM osobe	db.osobe.find();

SELECT * FROM osobe
WHERE god>25 AND god<=40
	db.osobe.find(
	{
	{“god”} : {$gt: 25, $lte:40}
	}
	);

SELECT * FROM osobe WHERE
rbr=1 OR god=50
	db.osobe.find(
	{
	$or: [{rbr:1}, {god:50}]
	}
	);

UPDATE osobe SET ime = “Marko” WHERE ime = “Petar”	db.osobe.update({ ime: “Petar”}, {$set: {ime: “Marko”} });
DELETE FROM osobe WHERE ime = “Marko”	db.osobe.remove({ ime: “Marko” });

Табела 9. Преглед SQL и MongoDB JSON упита

SQL упити са груписањем података попут пребројавања, сабирања и слично се пресликавају у агрегацијске функције у следећим MongoDB JSON упитима:

SQL упит	MongoDB JSON упит
SELECT COUNT (*) FROM osobe	db.osobe.aggregate({ $group: { _id: null, count: { $sum: 1}} });
SELECT SUM (iznos) FROM racun	db.racuni.aggregate ({ $group : { id : null, total: {$sum: "$iznos"}} });

Табела 10. Преглед SQL и MongoDB JSON упита

Компликовани SQL упити се пресликавају у MapReduce функције у оквиру JSON упита. Први корак у креирању MapReduce функције је заправо писање исте. MapReduce функција се извршава над сваким документом из збирке и помоћу ње се одређују кључеви за груписање и прикупљање података који се касније користе. Као пример који показује функцију која групише износе по месецу и години, припрема број и износ рачуна, дато је следеће [11]:

MapReduce функција

map = function () {
var mesecIgodina = this.datum.getMonth () +
This.datum.getYear ();
var broj = 0, ukupanbroj = 0;
emit (mesecIgodina, {broj = 1, iznos = this.iznos});
}

Табела 11. Пример MapReduce функције

У оквиру следеће табеле (табела број 12) приказан је пример SQL упита, као и идентичан пример MapReduce функције у MongoDB-у. Наиме, за сваког појединачног студента је потребно израчунати колико је испита положио и која је његова просечна оцена.

SQL упит	MongoDB JSON / MapReduce функција

SELECT
COUNT (*) AS broj_ispita,
AVG (ocena) AS prosecna_oc
 	
FROM ispiti
 	
GROUP BY studentOIB
	db.ispiti.group ({
	key : {studentOIB : true },
	initial : {broj_ispita : 0, prosecna_oc:0},
	reduce : function (doc, aggregator)
	{
	aggregator.broj_ispita+=1;
	aggregator.sum_ocena+=doc.ocena;
	}
	analiza: function(doc)
	{
	doc.prosecna_ocena=
	doc.sum_ocena/doc.broj_ispita;
	}
	});

Табела 12. Пример SQL упита и MongoDB MapReduce функције

3.4. Атомичност операција

Постоји велика разлика између релационих база података и MongoDB-а, укључујући карактеристике, начин употребе, могућности, а самим тим и примену база података. За разлику од релационих база, које подржавају трансакције, односно омогућују да се обаве све, или чак ни једна операција, код MongoDB-а тако нешто не постоји. Атомичност операција се гарантује само код рада са једним документом, користећи на пример неки од следећих оператора: *$set, $unset, $inc, $push, $pushAll* итд. којима се може изменити постојећи документ.

Поред предходно наведене могућности, за измену документа постоји и други начин коришћењем "*update if current*" методе која прво приступа документу, затим мења локалну копију, па се након тога измењени документ враћа MongoDB-у, који га успрешно чува само уколико је документ у збирци остао у међувремену непромењен. У табели 13 дат је пример измене стања артикла на складишту.

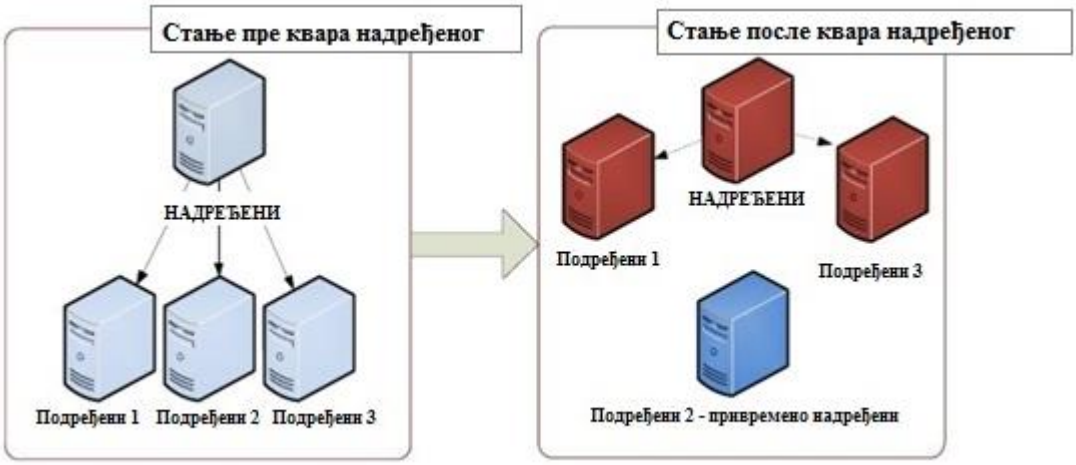
```
privremeni = dbv.stanjeArtikla.findOne ({ sifra: "123"});
stara?Kolicina = privremeni.kolicina;
privremeni.kolicina +=1;
db.stanjeArtikla.update({sifra: "123", kolicina : staraKolicina}, privremeni);
// Proverava da li je izmena podataka uspela i vraca gresku ako je u mdjuvremenu privremeni
podatak promenjen.
db.$cmd.findOne( {getlasterror: 1} );
// Ukoliko je izmena uspesno izvrsena, poslednji upit daje sledeci rezulat:
{"err" : , "updatedExisting" true, "n" : 1, "ok" :1}
db.stanjeArtikla.update({sifra: "123", {$inc: {{kolicina : 1} }} )
// Operator $inc sabira/oduzima broj u argumentu
```

Табела 13. Пример измене стања артикла на складишту

3.5. Репликација

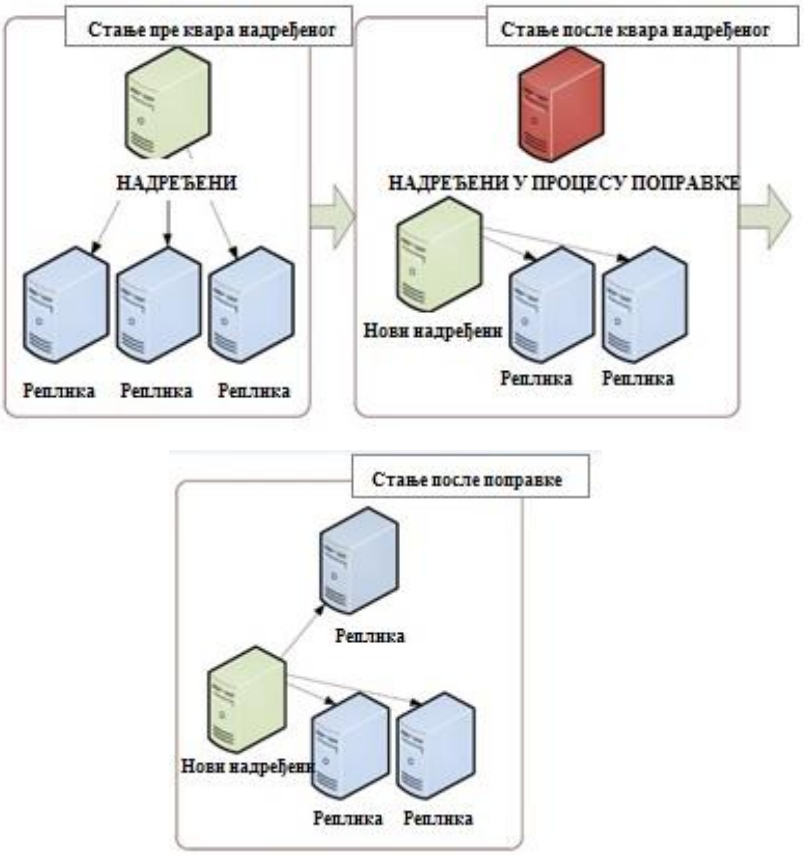
Под појмом репликације подразумева се стварање копија базе података на два или више сервера, како би се повећала поузданост приступања и отпорност на грешке. У случају да на једном од сервера дође до одређене грешке, други сервер на коме су сачуване копије базе података може преузети улогу првог сервера и рад са истом базом података би се наставио без икаквих сметњи.

MongoDB омогућује две врсте репликација: надређени – подређени (енг. *master – slave*) и скупови репликација (енг. *replica sets*). Прва врста репликације јесте уобичајени начин репликације података који је подржан код већине традиционалних, релационих база података. У случају квара надређеног сервера, рад са базама података се може наставити са само једним од подређених сервера. Да би се вратио надређени сервер, он се прво мора ускладити са подацима привременог подређеног сервера, како би могао наставити са радом као надређени сервер [5].



Слика 12. Процес након квара надређеног сервера код репликације надређени – подређени

Међутим, за разлику од репликације надређени – подређени, код репликације скуповима се унапред не одређује који је сервер надређени. Уколико дође до квара на одређеном серверу, један од преосталих сервера унутар истог скупа аутоматски преузима улогу надређеног сервера. Када се сервер који је био у кварy натраг враћа у скуп, прво се мора синхронизовати са осталим серверима како би ускладио податке и тада он ради као један од подређених сервера у скупу реплика, док улога надређеног остаје ономе ко ју је преузео током квара.



Слика 13. Процес поправке квара надређеног сервера код скупа реплика

Додатном предношћу репликације наводи се и повећање перформанси састава који имају велики број читања из базе података. У том случају, читања се могу поделити на све подређене сервере, али то је могуће само у случају када конзистентност података није битна, с обзиром да подаци на подређеним серверима не морају бити једнаки подацима на надређеним серверима. Подаци између надређеног сервера и осталих подређених сервера могу каснити ако је репликација асинхрона, односно ако надређени сервер не чека потврду подређених сервера да су се подаци заиста записали на њих. Због пада перформанси код синхроне репликације, данас се углавном користи асинхрона репликација [6].

3.6. Подела на одломке

У случају када један сервер не може обрађивати више захтева за читањем и записивањем података једно од решења јесте додати још један сервер који би аутоматски преузимао део података, а самим тим и обрађивао захтеве над тим подацима. Такав поступак назива се подела на одломке (енг. *sharing*) [11].

Подела на одломке се може извести на страни клијентске апликације или унутар саме базе података. У већини случајева, подела на одломке на страни клијентске апликације се изводи када база података нема те могућности. Тада је потребно имати

табеле у којима се записује који се подаци налазе на ком серверу. Сваки захтев за приступ података подразумева прво приступ таквој табели, затим проналажење на ком се серверу налазе потребни подаци и тек тада следи прослеђивање упита том серверу. Овим начином приступа долази до проблема када се један или више одломака преоптерети, јер се премештај података на нове додатне одломке мора обавити искључиво на страни апликације, или ручно.

У оквиру MongoDB-а, подела на одломке је подржана и за њихово коришћење нису потребне никакве измене унутар клијентске апликације. Премештај података и њихова миграција између одломака решени су на страни MongoDB-а.

3.6.1. Компоненте састава са поделом на одломке код MongoDB-а

MongoDB који омогућује поделу на одломке састоји се из неколико компоненти:

- Одломак (енг. *shard*) – један одломак може бити један сервер или чак сет реплика који се састоји од већег броја сервера.
- Mongos усмеривач – на кога се спаја клијентска апликација, њему упућује захтеве за читањем и записивањем, исто као што то ради када ради директно само са једним сервером без поделе на одломке. Може постојати већи број Mongos усмеривача, на пример да сваки апликацијски сервер има свој Mongos усмеривач.
- Конфигурацијски сервер – на коме се налазе подаци о физичким локацијама података, односно информације на којим се тачно серверима (или скуповима реплика) налазе подскупови комплетне базе података [5].

Слика 14. Компоненте MongoDB састава са поделом на одломке

Подаци се на одломке деле према кључу који се задаје у оквиру базе података. Врло је битно одабрати погодан кључ како би се подаци могли делити, јер уколико кључ не нуди довољну распршеност, подаци ће се груписати у превелике групе које не би могле да се премештају са сервера на сервер.

3.7. Ограничења и недостаци MongoDB-а

Иако своју примену налази у различитим областима и омогућава једноставнији и бржи начин рада, MongoDB због својих специфичности није погодан за примену у следећем:

- Рад са финансијским и рачуноводственим подацима, као и банкарским трансакцијама код којих је битно извршити неколико акција као једну акцију,
- Рад са подацима где је потребно генерисати ознаке попут редних бројева и слично. У оваквим случајевима потребно је користити помоћне табеле за ефикасно чување следећег редног броја,
- Рад са подацима код којих је потребно спајати више збирки у један резултат, с обзиром да MongoDB нема уграђене такве наредбе,
- Анализа велике количине непромењивих података, или складишта података [11].

Додатни, али и тренутни недостатак MongoDB-а је тај што је MongoDB релативно нов за разлику од релационих база података, па самим тим има мање документације о начину примене и мањи број особа са искуством рада са MongoDB-ом, али се ипак разним анкетирањем и истраживањем дошло до закључка да број корисника MongoDB-а расте невероватно великом брзином и да ће његова примена у будућности бити далеко изнад рада са другим типовима база података.

4. Креирање личног блога коришћењем MongoDB базе података и Python web технологија

У оквиру практичног дела рада биће представљен начин израде личног блога којим ће се употпунити предходно наведене могућности употребе MongoDB базе података са различитим веб технологијама.

Наиме, блог представља низ хронолошки организованих уноса текста који се приказују на веб страницама путем аутоматизованог софтвера који омогућује веома једноставно креирање, али и вођење истог. Реч „блог“ је настала од енглеске речи *„web”*, што у преводу значи мрежа и речи *„log“* која има значење уноса, или дневника [12].

Сам блог се може упоредити са *online* дневником или албумом, где се унос садржаја врши у стилу журнала и приказан је у обрнутом хронолошком реду, односно при врху странице приказани су нови садржаји, док се старији садржаји налазе на дну саме странице. На тај начин блог омогућава исказивање сопственог мишљења без икаквих ограничења. Како се из дана у дан повећава број корисника који почињу да деле своја мишљења са пријатељима и странцима, тако се повећава и број блогова везаних за мноштво различитих тема. Данас блогови, поред писаног, односно текстуалног садржаја углавном имају и постављене фотографије и/или видео туториал, као и аудио садржај.

Постављање садржаја и такозваних постова на блог врши једна или више особа у зависности од тога о каквом начину израде блога је заправо реч. Особа која уређује блог на било који начин носи име блогер, или администратор блога. Администратор има задатак да уређује блог. Уместо да своје мисли бележи у физичком дневнику, администратор поставља своје идеје, мисли и закључке на Интернет, надајући се да ће читаоци погледати оно што има да каже.

Сви садржаји одлазе у тзв. блогосферу, па се блогови могу сврстати у три основне категорије [12]:

1. Лични блог – већину блогова чине лични блогови који су фокусирани на тему у вези чега је блогер пасиониран, као што су технологија, спорт, мода, или чак неки хоби.

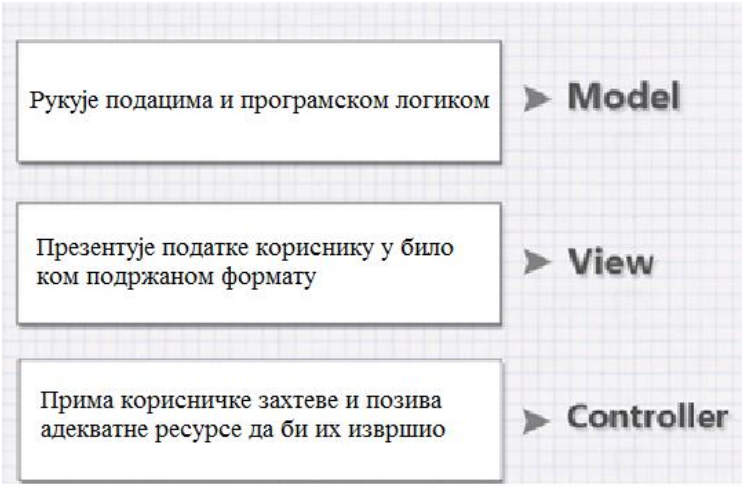
2. Организацијски блог – чине они блогови који се уређују од стране организација или компанија, као начин комуницирања са својим члановима и спољним светом ван саме организације (клијенти, спољни сарадници, читаоци).

3. Бизнис блог – представља трећу врсту блогова који су направљени у сврху зарађивања новца, било да је реч о рекламирању различитих производа и услуга, или путем продаје и промоције истих. Лични блог се такође делимично може сврстати и у ову групу када особа која га користи покушава да оствари приход или као саставни део свог посла.

Блог који је израђен у оквиру мастер рада сврстава се искључиво у врсту личног блога и апликација је писана праћењем MVC архитектуре.

MVC (енг. *Model – View – Controller*) је у последњих неколико година најчешће коришћени патерн у свету програмирања. MVC је архитектура која описује начин како да се структуира апликација и које су све одговорности и интеракције сваког дела те структуре, али и решава проблем у организацији самог кода [7].

MVC је први пут представљен 1979. године, али је имао другачији концепт од онога што има данас, управо из разлога што у то време концепт веб апликација није постојао, тако да је патерн који се данас користи у веб програмирању његова адаптација. До популаризације овог патерна у веб програмирању довела су два најкоришћенија framework – а, *Struts* и *Ruby on Rails*. Идеја иза MVC архитектуре је прилично једноставна – поделити одговорности између различитих слојева апликације.



Слика 15. Подела апликације на три главне команде где свака од њих обавља различте функције

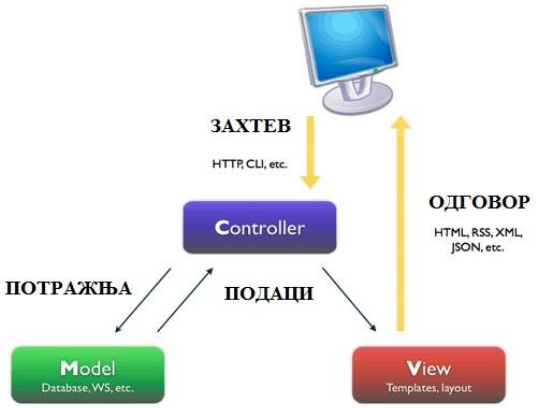
Дакле, апликација је подељена на три главне целине при чему свака од њих има задатак да обавља различите функције.

Контролер садржи главну контролу програма и одговоран је за његов ток. У веб апликацијама је први слој који се позива када *Browser* позове *URL*. Такође, контролер има задатак да управља корисничким захтевима (*HTTP, GET* или *POST* захтевима када корисник кликне на неки *GUI* елемент). Обично контролер позива одређени модел за задатак и затим изабере одговарајући поглед (*View*) [7].

Модел садржи главне програмске податке попут информација од објеката из базе података и SQL/NoSQL упита. Сви подаци се добијају од модела, али се он не може директно повезати, већ је контролер тај који од модела захтева одређене податке, модел тада обрађује захтеве и затим враћа податке контролеру.

View представља последњи слој MVC архитектуре који садржи корисничко окружење апликације, односно обезбеђује различите начине за презентовање података

које добија од модела. У веб апликацијама *View* садржи *HTML*, *CSS*, *JavaScript*, *XML* или *JSON*. *View* је видљив од стране корисника, док су сам модел и контролер скривени [7].



Слика 16. Шема главних компоненти апликације

Како корисници често имају допунске захтеве за дораду система, које се најчешће односе на измене на графичком интерфејсу, јасно је да сам дизајн представља проблем са становишта одржавања. Уколико се томе дода и захтев за обраду података, поред једноставног складиштења у базу, онда одржавање тако дизајнираног система добија на комплексности.

Разлози због комплексног одржавања су:

- Кориснички интерфејс се чешће мења него пословна логика (посебно код веб апликација),
- Ако се логика за кориснички интерфејс и пословну логику налази у једној класи, за њено одржавање је потребно више времена него када су одговорности разграничене,
- У неким сценаријима исти подаци се могу приказати на различите начине (нпр. статистички подаци се могу приказивати у форми графикона, табеле, попуњавањем неког предефинисаног *MS Word* или неког другог шаблона итд).

Међутим, предности које укључују коришћење MVC патерна у имплементацији су:

- Омогућава да се модел може приказати на више начина,
- Лакше је додати нови *View* (нпр. нову веб страницу која приказује постојеће податке или део њих),
- Омогућава да се интеракција са корисником лако промени,
- Омогућава да више програмера паралелно раде на различитим слојевима (што је посебно важно у тимовима који су специјализовани за поједине слојеве, као на пример тим задужен за имплементацију и одржавање корисничког интерфејса, пословну логику итд).

У овом раду је описан процес креирања личног блога коришћењем популарног Python веб фрејмворка – Flask, и MongoDB базе података.

Лични блог, под називом Danka – Blog, ће се састојати из два дела:

1. Јавног дела, који омогућава посетиоцима да читају текстове аутора, гледају видео материјале и износе своје мишљење везано за садржај писањем коментара,
2. Администраторског дела, који само аутору омогућава додавање или измену садржаја блога.

Python је мултиплатформски, мултифункционални, интерпретирани програмски језик високог нивоа са великом и свеобухватном стандардном библиотеком. Може се користити како за писање скрипти, тако и за процедурално, функционално и објектно оријентисано програмирање. Његова филозофија се заснива на једноставности и лакој читљивости самог кода. Једноставан програмски језик са малим основама, али и великом и проширљивом стандардном библиотеком уграђеном у сам интерпретатор.

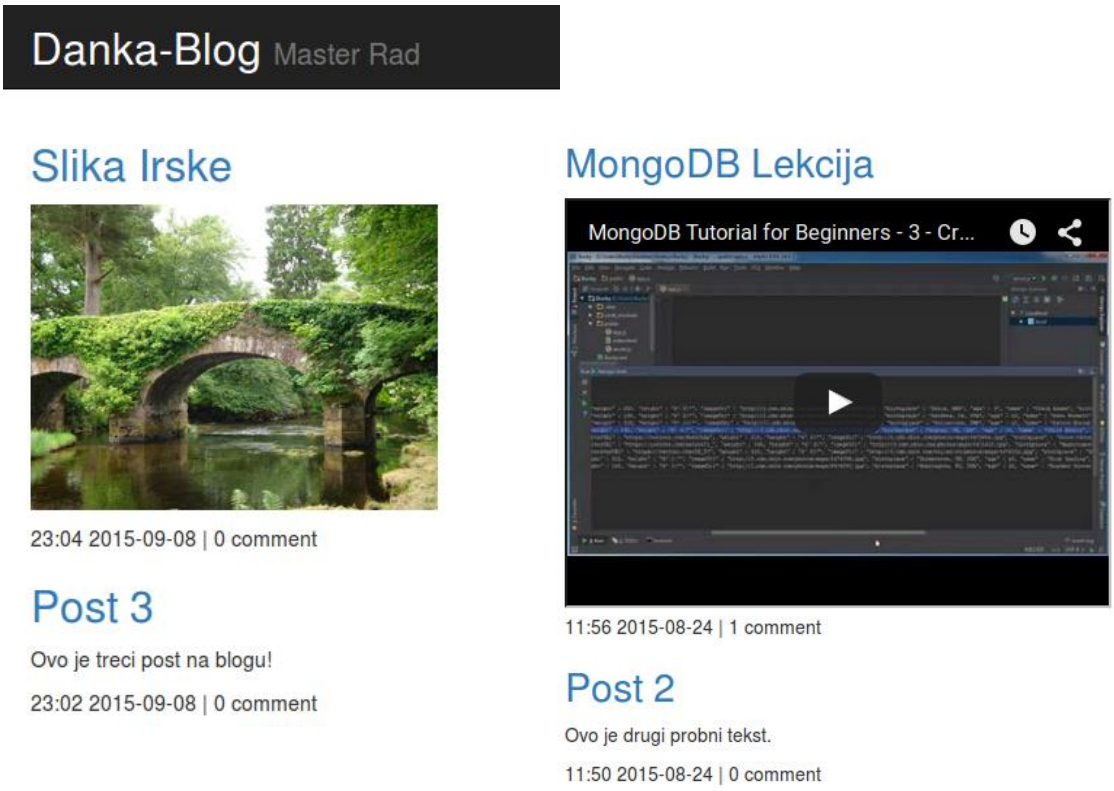
Python је динамички програмски језик са аутоматским управљањем меморије. Python обезбеђује начин додавања (на пример) C кода и његово штампање са остатком програма писаног у Python-у. Управо у томе се огледа његова модуларност. Такође, веома лако се могу укључити модули писани у C-у, C++-у, Java-и и осталим подржаним језицима [13].

4.1. Покретање и опис садржаја блога

Да би се покренуо лични блог, креиран у сврху практичног дела рада, потребно је пре свега покренути виртуелну машину, а онда у терминалу уписати следећи редослед команди:

```
cd master_rad/
source venv/bin/activate
python manage.py runserver
```

Након што се у оквиру терминала појави и линк самог блога, истовременим притиском на тастер *Ctrl* и кликом на линк 0.0.0.0:5000, отвара се јавна страна Danka – Blog-а, чији је изглед приказан на слици 17.



Слика 17. Јавна страна Danka – Blog-а

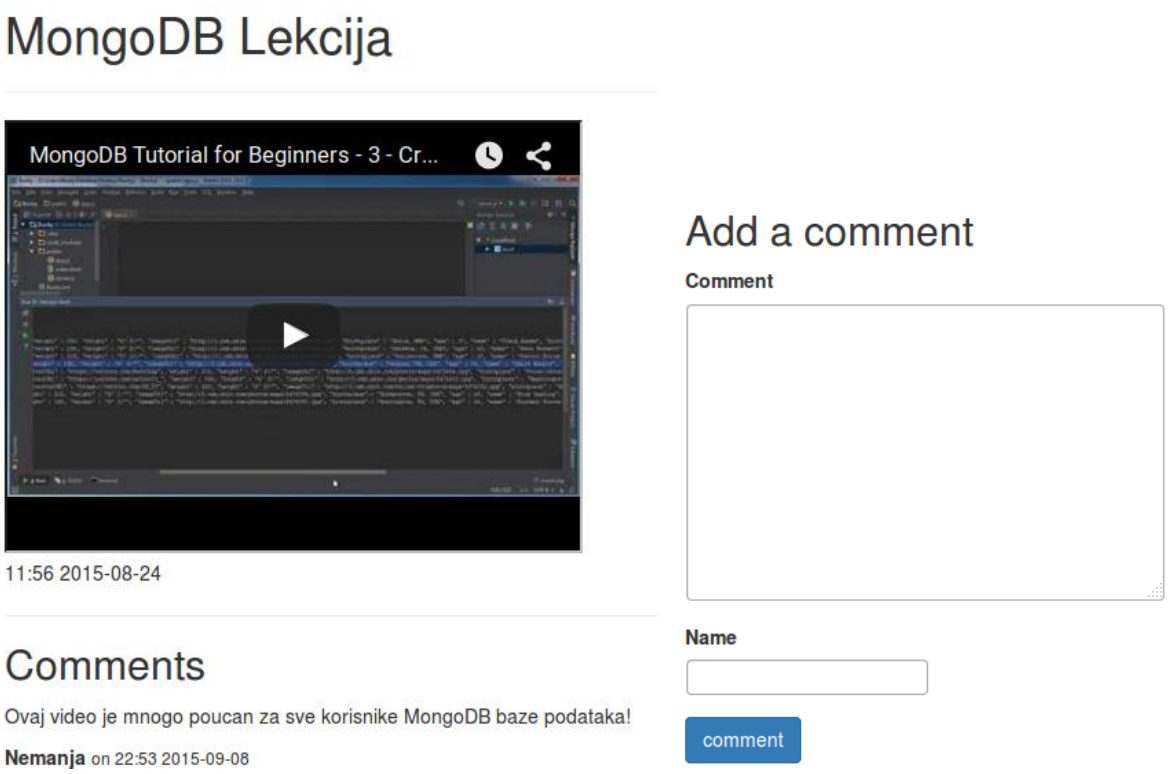
Блог се састоји из:

1. Наслова блога,
2. Дела садржаја блога и
3. Коментара сваког од постова у оквиру кога је дат и број укупних коментара на одређени пост, као и последње време коментара.

Блог тренутно садржи неколико постова, међу којима су: Slika Irske, Post 3, MongoDB Lekcija и Post 2, а сваки од њих има јединствену URL адресу која се динамички прави и о којој ће бити више речи у наредном излагању у раду када ће бити објашњена и администраторска страна блога.

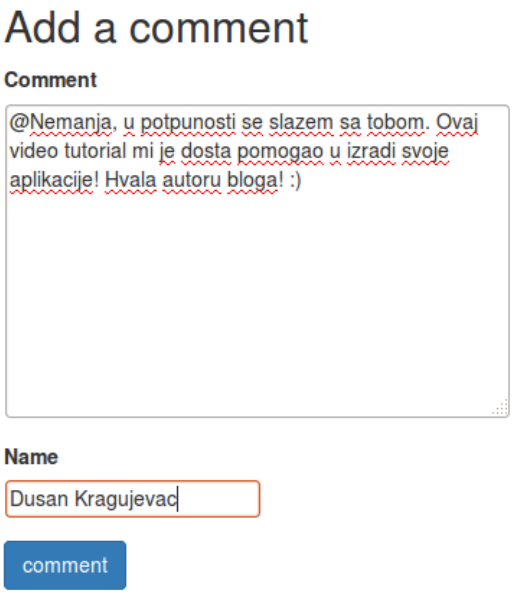
Јавна страна блога има URL адресу 0.0.0.0:5000 и у оквиру те стране се налази комплетан садржај постова од стране аутора блога. Самом блогу може приступити било који корисник, пратити свако ажурирање блога, прегледати текстове, слике, аудио материјал и гледати било који од постављених видео материјала у складу са његовим интересовањем. Такође, сваки посетилац блога има могућност дељења мишљења и искустава са осталим посетиоцима блога путем коментара.

Остављање коментара на блогу се врши на врло једноставан начин. Да би корисник посетио одређени садржај блога, потребно је само да кликне на линк онога што жели да види. На пример, кликом на MongoDB Lekcija, отвара се страница као на слици 18.



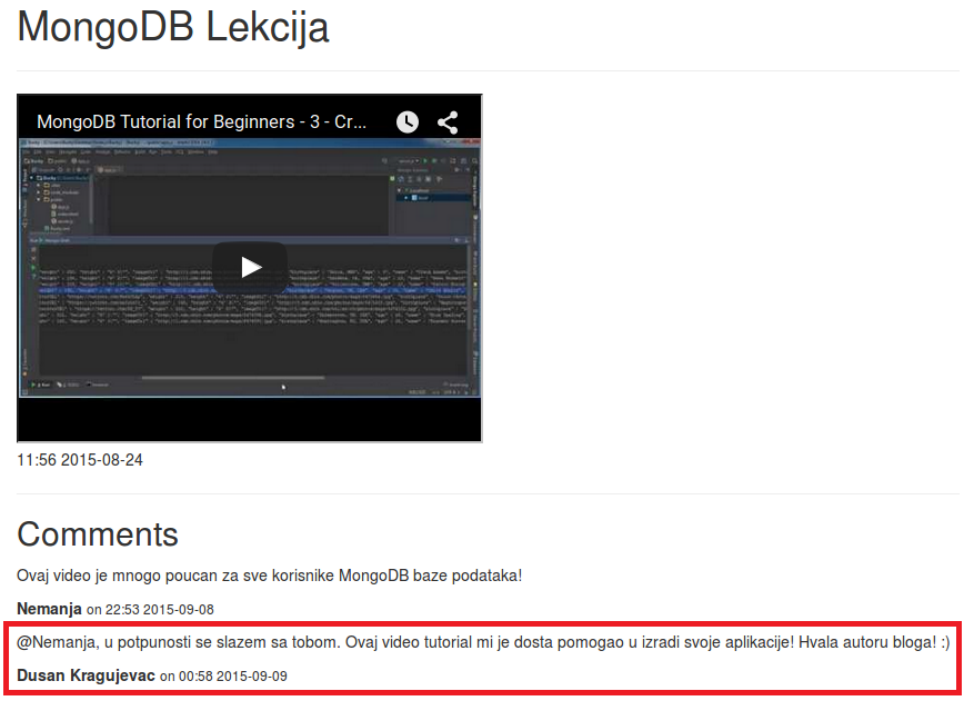
Слика 18. Пост MongoDB Lekcija

На слици 18 уочава се постављен видео материјал који се може погледати кликом на дугме *Play*, коментари од стране других корисника, тачно време и датум остављања коментара и име корисника који је коменатар писао. Сваки нови посетилац блога такође може оставити своје мишљење, савет или препоруку тако што ће у тело предвиђено за писање коментара написати своје мишљење, а у простор испод оставити своје име.



Слика 19. Писање коментара на блогу

Кликом на дугме *comment*, корисник је успешно објавио свој коментар, што се може видети на слици 20.



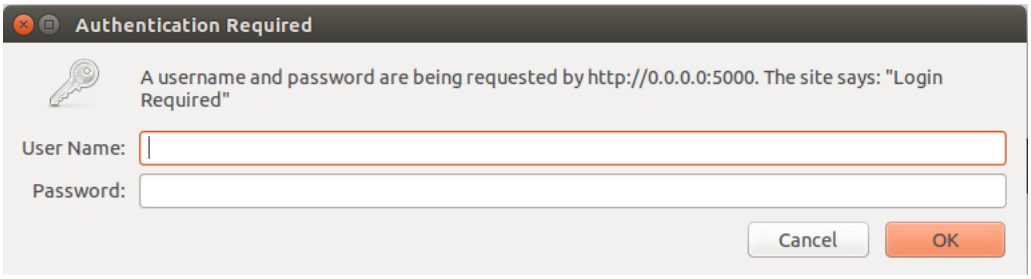
Слика 20. Постављање коментара на блог

На исти начин посетиоци блога могу остављати своје мишљење за било који од постова на блогу.

Лична – администраторска страна блога

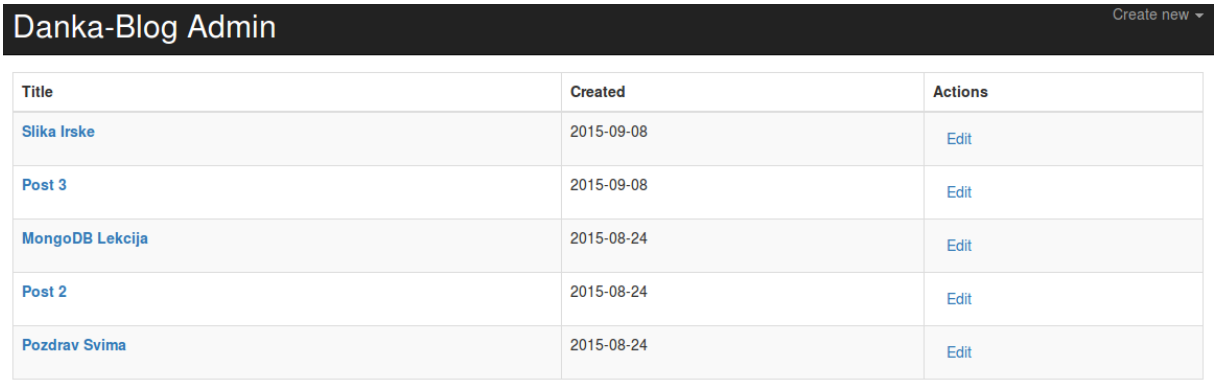
С обзиром да је у оквиру рада креиран блог искључиво за личну употребу, долази се до закључка да постоји само један администратор блога, али је такође потребно нагласити да то не значи да сваки блог има само једног администратора, већ да блог може бити уређиван и од стране више лица.

Да би се приступило администраторској страни блога, потребно је у новој картици претраживача уписати следећу URL адресу: 0.0.0.0:5000/admin/. Након што се приступи тој страници, појавиће се прозор као на слици 21.



Слика 21. Регистровање админа

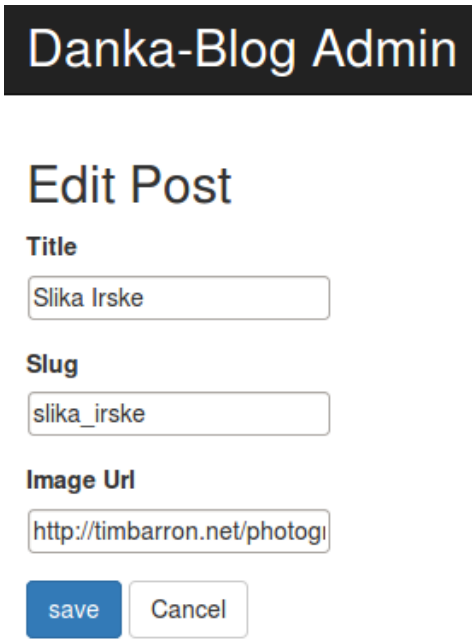
Дакле, да би се приступило личној страни блога, пре свега је потребно уписати корисничко име и лозинку администратора. User Name и Password овог блога су *admin* и *secret*, респективно. Кликом на дугме OK, отвара се администраторски део блога чији је изглед дат у облику табеле, као што је приказано на слици 22.



Слика 22. Лична – администраторска страна блога

Са предходне слике јасно се види наслов личне стране блога (Danka-Blog Admin), затим наслови сваког објављеног поста, датуми када су они креирани и опција *Actions* у оквиру које се могу вршити измене већ постављених садржаја.

Администратор блога има могућност измене било ког садржаја на страници личног блога. Да би се тај принцип измене објаснио, узеће се, на пример, пост под именом Slika Irske. Кликом на линк, отвара се страна као што је приказано на слици 23.

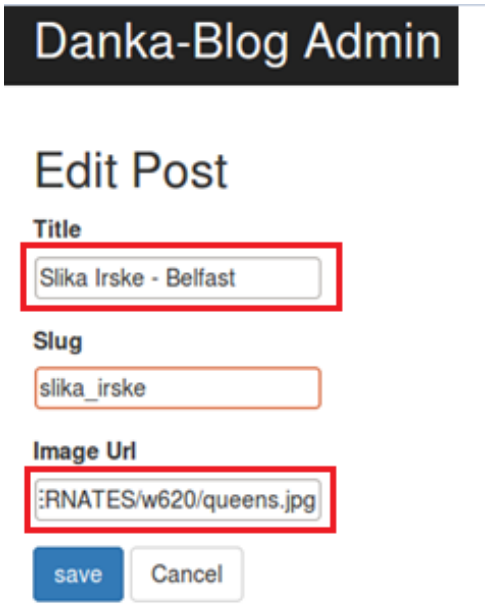


Слика 23. Измена одређеног поста блога

Са подацима (*Title*, *Slug* и *Image Url*) који су приказани на предходној слици, на јавној страни блога тај пост има следећи изглед:

Слика 24. Изглед поста пре измене од стране администратора блога

Међутим, администратор може изменити наслов поста (*Title*), затим *Slug* или чак променити комплетну слику (*Image Url*) која се налази у оквиру поста Slika Irske. Наслов Slika Irske ће заменити наслов Slika Irske – Belfast, као и нови линк, који ће водити ка слици која ће бити приказана у новој, измењеној страници блога.

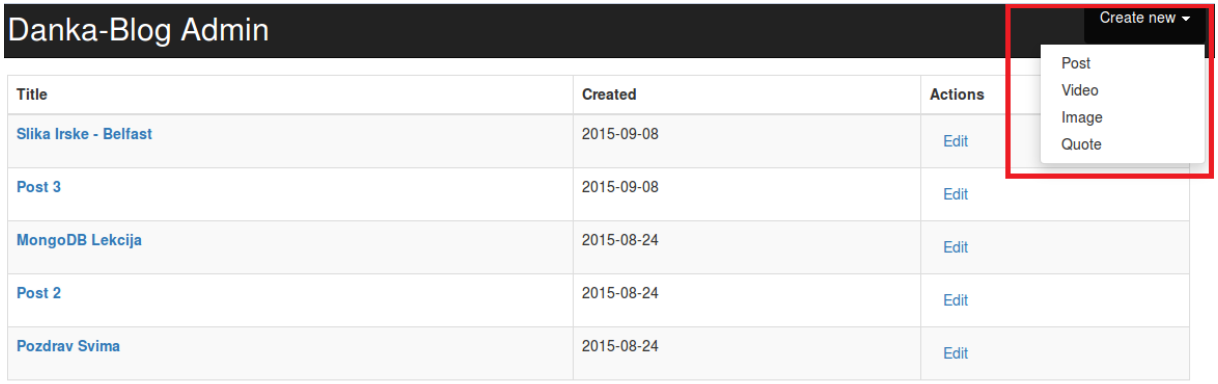


Слика 25. Измена предходно постављеног поста

Кликом на дугме *save* измене ће бити сачуване и пост ће имати нови изглед који је приказан на слици 26.

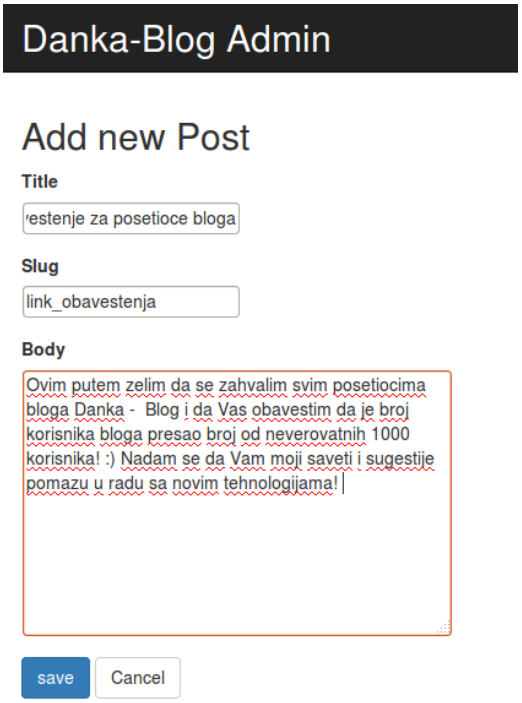
Слика 26. Изглед поста након измене од стране администратора блога

Поред ажурирања већ постојећих садржаја на блогу, администратор има могућност додавања нових постова, укључујући текстуални садржај, видео материјал или нове фоторафије. Да би се то постигло, потребно је на личној страни блога одабрати одређен садржај који ће се додати.



Слика 27. Додавање нових постова на блог

Једноставним одабиром из листе *Create new* са предходне слике, може се додати нови садржај. За пример ће се узети додавање новог поста. Кликом на Post отвара се нови прозор у оквиру кога је потребно написати наслов поста, поставити линк који ће водити до истог и уписати жељени текст.



Слика 28. Додавање новог поста на блог

Пост који је додат на блог носи назив *Obavestenje za posetioce bloga*. Када се новом посту приступи са јавне стране, тада исти пост има следећи изглед:

Слика 29. Приказ новог поста са јавне стране блога

Као и на раније постављеним постовима, и на нове постове се могу остављати коментари од стране посетиоца блога.

Слика 30. Додавање коментара на новопостављени садржај блога

На начин сличан овом, могу се обајвити и било који други садржаји на блог, фотографије, аудио и видео материјал и слично. Администратор блога се труди да својим креативним и занимљивим начином уређења блога помогне корисницима да лакше превазиђу проблеме на које наилазе.

4.2. Опис основних команди израде блога

Први корак у писању Flask апликације јесте дефинисање „модела“. Модел који одговара представљеној структури блога и којим се чувају подаци у MongoDB бази садржан је у *model.py* фајлу и има следећу структуру:

```
class Post(db.DynamicDocument):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=True)
    title = db.StringField(max_length=255, required=True)
    slug = db.StringField(max_length=255, required=True)
    comments = db.ListField(db.EmbeddedDocumentField('Comment'))

    def get_absolute_url(self):
        return url_for('post', kwargs={"slug": self.slug})

    def __unicode__(self):
        return self.title

    @property
    def post_type(self):
        return self.__class__.__name__

    meta = {
        'allow_inheritance': True,
        'indexes': ['-created_at', 'slug'],
        'ordering': ['-created_at']
    }

class BlogPost(Post):
    body = db.StringField(required=True)

class Video(Post):
    embed_code = db.StringField(required=True)


class Image(Post):
    image_url = db.StringField(required=True, max_length=255)

class Quote(Post):
    body = db.StringField(required=True)
    author = db.StringField(verbose_name="Author Name", required=True,
max_length=255)

class Comment(db.EmbeddedDocument):
    created_at = db.DateTimeField(default=datetime.datetime.now, required=True)
    body = db.StringField(verbose_name="Comment", required=True)
    author = db.StringField(verbose_name="Name", max_length=255, required=True)
```

Пример *View* – а који одговара почетној страници блога, на којој се налазе сви постови, садржан је у фајлу *list.html* и има следећу структуру:

```
{% extends "base.html" %}

{% block content %}
   {% for post in posts %}
     <h2><a href="{{ url_for('posts.detail', slug=post.slug) }}">{{ post.title }}</a></h2>
     {% if post.body %}
       {% if post.post_type == 'Quote' %}
         <blockquote>{{ post.body|truncate(100) }}</blockquote>
         <p>{{ post.author }}</p>
       {% else %}
         <p>{{ post.body|truncate(100) }}</p>
       {% endif %}
     {% endif %}
     {% if post.embed_code %}
       {{ post.embed_code|safe() }}
     {% endif %}
     {% if post.image_url %}
       <p><img src="{{ post.image_url }}" /><p>
     {% endif %}
     <p>
       {{ post.created_at.strftime('%H:%M %Y-%m-%d') }} |
       {% with total=post.comments|length %}
          {{ total }} comment {%- if total > 1 %}s{%- endif -%}
       {% endwith %}
     </p>
   {% endfor %}
{% endblock %}
```

Апликациона логика која одговара аутентификацији и ауторизацији администратора и у којој је дефинисано корисничко име (*admin*) и лозинка (*secret*), налази се у *auth.py* фајлу:

```
from functools import wraps
from flask import request, Response


def check_auth(username, password):
    """This function is called to check if a username /
    password combination is valid.
    """
    return username == 'admin' and password == 'secret'


def authenticate():
    """Sends a 401 response that enables basic auth"""
    return Response(
    'Could not verify your access level for that URL.\n'
```

```
    'You have to login with proper credentials', 401,
    {'WWW-Authenticate': 'Basic realm="Login Required"'})


def requires_auth(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        auth = request.authorization
        if not auth or not check_auth(auth.username, auth.password):
            return authenticate()
        return f(*args, **kwargs)
    return decorated
```

Контролер који садржи главну контролу програма, управља корисничким захтевима и комуницира са MongoDB базом података дефинисан је у фајлу *views.py*:

```
from flask import Blueprint, request, redirect, render_template, url_for
from flask.views import MethodView

from flask.ext.mongoengine.wtf import model_form
from models import Post, Comment

posts = Blueprint('posts', __name__, template_folder='templates')


class ListView(MethodView):

    def get(self):
        posts = Post.objects.all()
        return render_template('posts/list.html', posts=posts)


class DetailView(MethodView):

    form = model_form(Comment, exclude=['created_at'])

    def get_context(self, slug):
        post = Post.objects.get_or_404(slug=slug)
        form = self.form(request.form)

        context = {
            "post": post,
            "form": form
        }
        return context

    def get(self, slug):
        context = self.get_context(slug)
        return render_template('posts/detail.html', **context)
```

```
    def post(self, slug):
        context = self.get_context(slug)
        form = context.get('form')

        if form.validate():
            comment = Comment()
            form.populate_obj(comment)

            post = context.get('post')
            post.comments.append(comment)
            post.save()

            return redirect(url_for('posts.detail', slug=slug))
        return render_template('posts/detail.html', **context)


# Register the urls
posts.add_url_rule('/', view_func=ListView.as_view('list'))
posts.add_url_rule('/<slug>/', view_func=DetailView.as_view('detail'))
```

Основни конфигурациони параметри неопходни ради повезивања Flask фрејмворка са MongoDB базом података налазе се у *__init__.py* фајлу:

```
from flask import Flask
from flask.ext.mongoengine import MongoEngine

app = Flask(__name__)
app.config["MONGODB_SETTINGS"] = {"DB": "localhost:27017"}
app.config["SECRET_KEY"] = "KeepThisS3cr3t"

db = MongoEngine(app)

def register_blueprints(app):
    from views import posts
    from admin import admin
    app.register_blueprint(posts)
    app.register_blueprint(admin)

register_blueprints(app)

if __name__ == '__main__':
    app.run()
```

Све инструкције неопходне за покретање апликације написане у Flask фрејмворку, налазе се у *manage.py* фајлу:

```
import os, sys
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from flask.ext.script import Manager, Server
from tumblelog import app

manager = Manager(app)
manager.add_command("runserver", Server(
    use_debugger = True,
    use_reloader = True,
    host = '0.0.0.0')
)

if __name__ == "__main__":
    manager.run()
```

Покретање апликације се врши следећом наредбом:

```
python manage.py runserver
```

5. Закључак

Напредак технологије и сам раст количине података за обраду довео је до потребе за стварање бржег начина обраде истих. Сама чињеница да се олакша сређивање велике количине података, до које се све чешће неминовно долази, створила је нову врсту база података коју одликују боље перформансе од старијих, релационих база података.

NoSQL базе података обухватају скуп различитих врста технологија база података који су развијени према захтевима тржишта. MongoDB представља базу података засновану на документима, која је уједно и најчешће употребљавана база података када је реч о нерелационим базама.

Разлоге брзог раста броја корисника MongoDB базе података карактерише низ многобројних предности, нарочито у време када се јавља потреба за чувањем велике количине података за обраду, укључујући податке о корисницима, услугама и производима, где је уједно потребно омогућити већи број приступа тим подацима, као и смањење времена обраде истих.

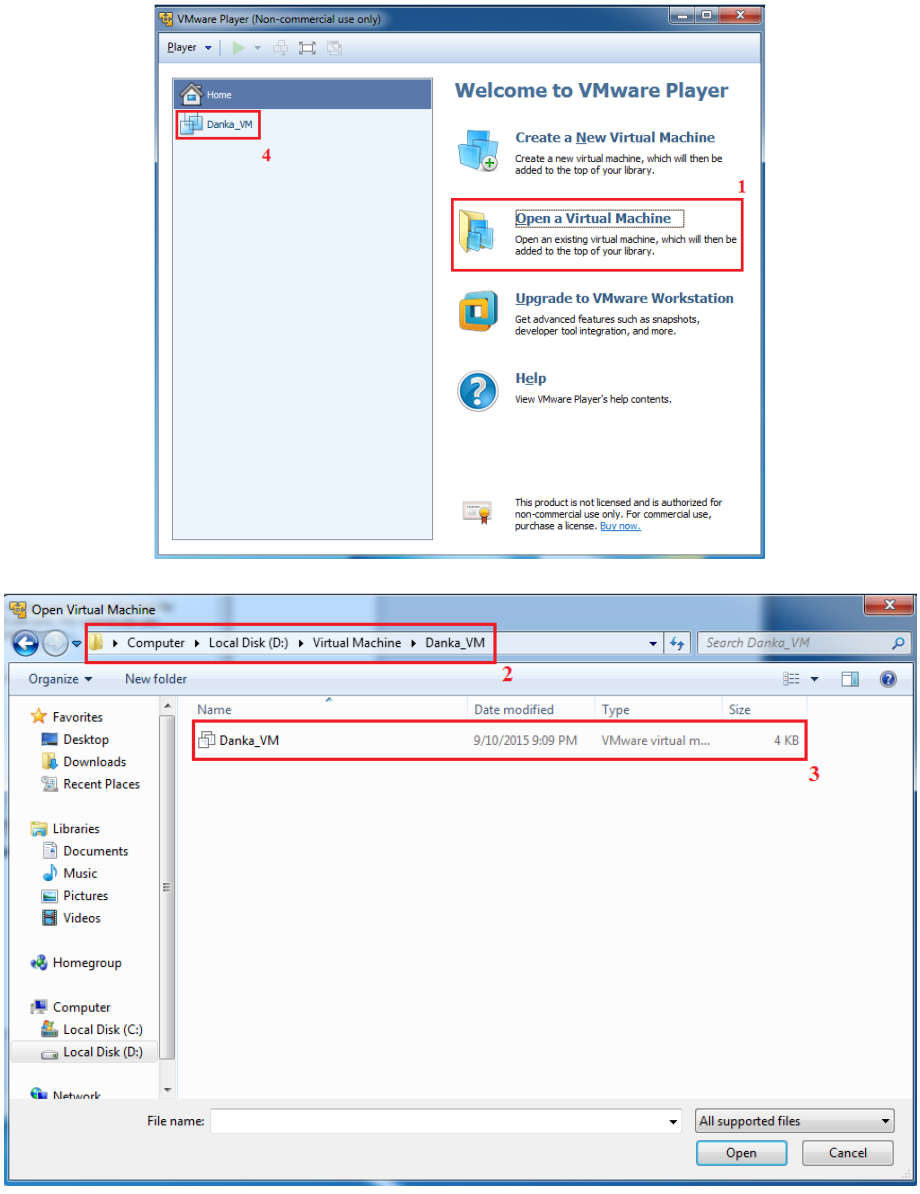
MongoDB своју примену налази у различитим гранама привреде. Због своје флексибилности и једноставности употребе запажа се све већа примена и код објектно оријентисаног програмирања и креирања апликација за управљање садржајем. Примена MongoDB база података знатно је олакшала креирање *Online* рачунарских игрица, код којих је потребан велики број записивања и читања од стране много корисника, а да се при том не утиче на брзину извођења.

Из личног виђења, будућност и мотивација MongoDB базе података лежи у ширењу броја корисника, управо из разлога што се сваким даном на тржишту јављају нове апликације које брзо налазе своју примену, па се самим тим јавља и велика количина података. Пример тога јесте велика заступљеност друштвених мрежа, где милиони корисника истовремено објављују мноштво постова, аудио и видео записа и коментара и где је потребно складиштити податке невероватно великом брзином. Такође, примена MongoDB базе података програмерима знатно поједностављује развој мобилних апликација за управљање геопросторним подацима као што је *Google Maps,* где су потребна честа ажурирања и приступи подацима смештеним у базу података.

Додатак А – Покретање радног окружења

Комплетно радно окружење које садржи све оно о чему је било речи у овом раду, односно инсталирану и конфигурисану MongoDB базу података и целокупан изворни код апликације, инсталирано је на Ubuntu виртуелној машини која се налази на пратећем *DVD* диску. За покретање виртуелне машине неопходно је да се има инсталиран *Vmware Player*, који је доступан за бесплатно преузимање на следећој веб адреси: http://vmw.re/1izV0je.

Поступци за учитавање и покретање пратеће виртуелне машине дати су на слици 31.



Слика 31. Учитавање и покретање виртуелне машине

Додатак Б – Инсталација MongoDB базе података на Ubuntu оперативном систему

Инсталација MongoDB базе података подразумева задавање скупа команди у терминалу Ubuntu оперативног система, у следећем редоследу:

- Увоз јавног кључа (*public key*) који користи систем за управљања пакетима

Инсталирање MongoDB-а подразумева увоз јавног кључа који користи систем за управљање пакетима. Потребно је да систем за управљање пакетима (*Ubuntu package management tools*) обезбеде доследност и аутентичност захтевајући да дистрибутери изврше потписивање пакета са GPG кључевима.

Потребно је издати следећу команду да би се увезао јавни GPG кључ MongoDB-а:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
```

- Креирање листе фајлова за MongoDB

Овај корак подразумева креирање листе датотека користећи команду која одговара верзији Ubuntu-а.

За Ubuntu 12.04 верзију потребно је изабрати следеће:

```
echo "deb http://repo.mongodb.org/apt/ubuntu precise/mongodb-org/3.0 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list
```

док је за Ubuntu 14.04 потребно нешто другачије:

```
echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.0 multiverse" | sudo
tee /etc/apt/sources.list.d/mongodb-org-3.0.list
```

У оквиру рада коришћена је верзија Ubuntu 14.04.

- Поновно учитавање система за управљања пакетима

Потребно је издати следећу команду да би се поново учитала база података:

```
sudo apt-get update
```

- Инсталирање MongoDB пакета

Може се инсталирати најновија верзија MongoDB-а, или нека друга, одређена верзија. Због своје стабилности, у раду је коришћена најновија верзија MongoDB базе података. Њена инсталација врши се издавањем следеће команде:

```
sudo apt-get install -y mongodb-org
```

Покретање MongoDB – а

MongoDB чува своје фајлове у */var/lib/mongodb* и покреће се помоћу MongoDB корисничког налога, што доводи до закључка ако се промени корисник који води MongoDB процес, мора се изменити контрола приступа права на */var/lib/mongodb* и */var/log/mongodb* директоријума да би кориснички приступ био омогућен.

- Покретање MongoDB базе података

Да би се стартовао MongoDB издаје се следећа команда:

```
sudo service mongod start
```

- Провера да ли је MongoDB успешно започео

Следећом командом се проверава да ли је сам процес успешно започео, тако што се проверава *log* фајл на */var/log/mongodb/mongod.log* за линију читања:

```
[initandlisten] waiting for connections on port <port>
```

где је <port> подразумевано конфигурисан у */etc/mongod.conf.*

- Заустављање – искључивање MongoDB базе података врши се издавањем следеће наредбе:

```
sudo service mongod stop
```

- Рестартовање MongoDB – а врши се командом:

```
sudo service mongod restart
```

5. Литература

[1] **Јанковић О.**, (2015), *NoSQL граф база података, база података од домена до модела преко упита – први део*, Бијељина, БиХ

[2] **ФЕР – Завод за телекомуникације**, (2014/2015), *Нерелационе базе података*, Сплит

[3] **Пероковић М.**, (2014), *Примена NoSQL база података на систем за управљање документима, дипломски рад,* Вараџин

[4] **Јанковић О.**, (2015), *NoSQL граф база података, база података од домена до модела преко упита – други део*, Бијељина, БиХ

[5] **Хабјанец Љ.**, (2012), *NoSQL системи за управљање базама података – завршни рад,* Загреб

[6] **Hows David P.**, (2013), *MongoDB Basic*, London

[7] **Маравић С.**, (2014), *Едукација, технологија и IT, MVC за почетнике,* Београд

[8] **Пушић И.**, (2014), *Интеграција хетерогених база података на примеру комплексне Real – Time апликације – дипломски рад*, Вараџин

[9] http://www.it-modul.rs/04/2013/baze-podataka/ , *приступљено сајту, јули, 2015.*

[10] https://sr.wikipedia.org/sr/MongoDB , *приступљено сајту, август, 2015.*

[11] https://www.mongodb.org/ , *приступљено сајту, јули, 2015.*

[12] https://sr.wikipedia.org/wiki/blog , *приступљено сајту, септембар, 2015.*

[13] http://www.it-modul.rs/09/2012/python-programski-jezik/ , *приступљено сајту, септембар, 2015.*