Садржај:

[1 NoSQL у дигитализацији књига 2](#_Toc463274544)

[2 NоSQL базе података 5](#_Toc463274545)

[2.1 Карактеристике NoSQL база података 5](#_Toc463274546)

[2.2 Подела NoSQL база података 8](#_Toc463274547)

[2.3 Упоређивање NoSQL база са релационим базама 10](#_Toc463274548)

[3 MongoDB 12](#_Toc463274549)

[3.1 Модел података – *JSON/BSON* 12](#_Toc463274550)

[3.2 Модел упита 14](#_Toc463274551)

[4 NoSQL у дигитализацији књига 15](#_Toc463274552)

[4.1 Кориснички захтеви за апликацију 15](#_Toc463274553)

[4.2 Увод у практичну израду пројекта 15](#_Toc463274554)

[4.2.1 Постављање и коришћење MongoDB-a 16](#_Toc463274555)

[4.2.2 GridFS 17](#_Toc463274556)

[4.2.3 Модел података 18](#_Toc463274557)

[4.3 Практична израда 19](#_Toc463274558)

[4.3.1 Унос фајла у базу 19](#_Toc463274559)

[4.3.2 Претрага базе и приказивање резултата 22](#_Toc463274560)

[4.3.3 Приказивање фајла 27](#_Toc463274561)

[4.3.4 Ажурирање фајла 28](#_Toc463274562)

[4.4 Корисничко окружење 30](#_Toc463274563)

[5 Закључак 33](#_Toc463274564)

[6 Литература 35](#_Toc463274565)

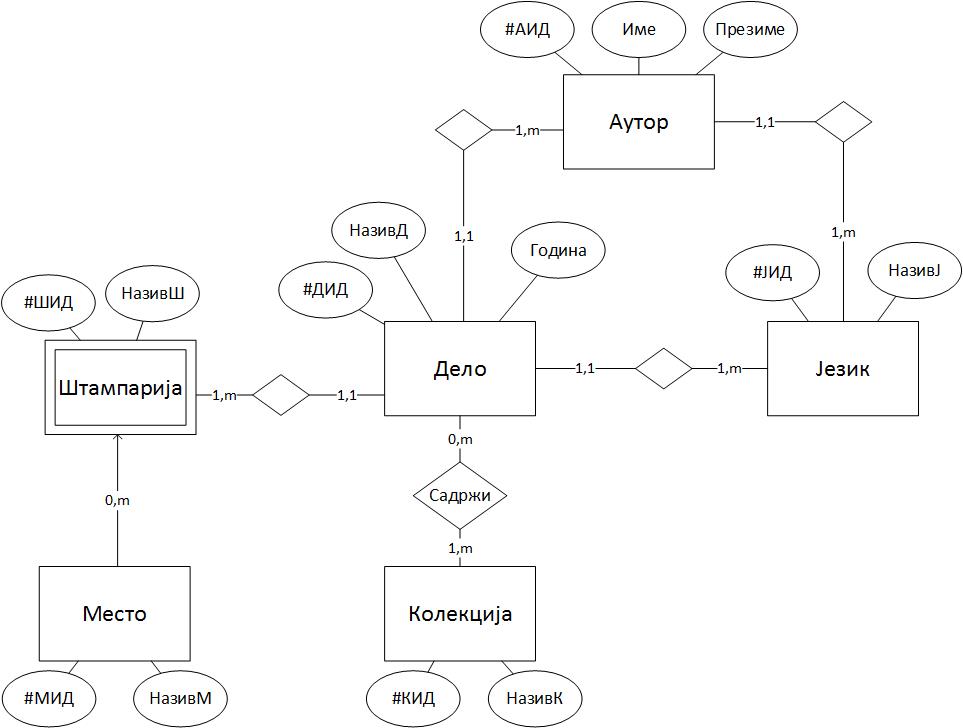
# NoSQL у дигитализацији књига

Дигитализација у контексту овог рада представља поступак обраде физичке копије књиге на такав начин да се на крају поступка добије њена електронска верзија. У Србији се дигитализација спроводи углавном у библиотекама, и већина библиотека поседује своју online дигиталну библиотеку (Библиотека града Београда, Народна библиотека Србије, библиотека САНУ, Библиотека Матице српске,...) која је доступна корисницима без икакве надокнаде.   
 Разлог због којег библиотеке све више приступају покрету дигитализације својих фондова није само омогућивање доступности грађе корисницима, већ је примарна сврха очување културне баштине. Стога се примарна грађа за дигитализацију узима из области старе и ретке књиге, које по тренутном закону представљају рукописи, штампане књиге до 1867. године и ретка издања књига из ратних периода (1912-1913, 1914-1918, 1941-1945). На овај начин се, као што је већ наведено, чува културно добро Србије, и омогућава се бесплатан приступ свим корисницима који поседују конекцију на интернет..

Процедура дигитализације креће од избора наслова (постоје утврђени критеријуми), а затим оператер за дигитализацију приступа њеној обради. Алати које се користе су скенер (обичан А3 формат или специјализован за ову врсту посла) и софтверски Adobe пакет. Од страница књиге се праве .jpg фајлови у резолуцији од 300 тпи, затим се обрађују у Photoshop и InDesign-у, након чега се праве две верзије PDF-а: високог квалитета 300 тпи и мањег квалитета од 72/96 тпи који је стандардизован за web приказ. Праксе се донекле разликују па тако у различитим дигиталним библиотекама можемо наћи наслове окачене као засебне .jpg странице или спаковане у један целовит PDF документ.

Како би се значај и смисао овог рада сместио у контекст, неопходно је на примеру објаснити конкретан проблем који ће се даље решавати.

Након што је обрада завршена, дигитални објекат се смешта у базу, како би му се могло приступити ради добијања информација о објекту или како бисмо прегледали сам објекат.  
Приликом уноса неопходно је осим објекта, унети и његове метаподатке у циљу његовог претраживања и класификовања. С обзиром на то да су у питању дигитализоване књиге тражени подаци обухватају наслов, годину издања, писца и још неколико специфичних детаља.   
У већини случајева једини заиста јединствен податак представља наслов дела, па би у случају уношења у релациону базу то био примарни кључ, док би остала поља као што су име и презиме аутора, година издања, штампарија, место издавања, врста грађе, колекција којој припада и језик на коме је написано представљало колоне којe зависе од примарног кључа.   
 Када се ова почетна табела разврста по нормалним формама, добијамо следећу ситуацију: основна табела "Дело" која садржи податак о наслову и години издавања, табела "Аутор" која садржи име и презиме аутора, табела "Врста" која садржи физички опис грађе (књига, фотографија, часопис...), табела "Колекција" која садржи податке о tоме које је дело смештено у коју колекцију, табела "Штампа" која представља слаб објекат табеле "Место", и на крају табела "Језик" у којој је наведен језик на коме је објављено дело (уколико представља писано дело).



*Слика 1.1 Пример изведбе ПМОВ-а у релационој бази података*

Преведено у релациони модел добићемо следеће табеле:

Дело (#ДИД, НазивД, Година, *ШИД, МИД, АИД, КИД, ЈИД*)

Аутор (#АИД, Име, Презиме, *ЈИД*)

Језик (#ЈИД, Језик)

Место (#МИД, НазивМ)

Штампарија (#ШИД, #МИД, НазивШ)

Колекција (#КИД, НазивК)

Садржи (#ДИД, #КИД)

Као што се може приметити добијамо велики број међусобно повезаних табела за овакав вид објекта, што и није баш толико оптимално. Стога је овај рад посвећен истраживању са неким другим типом база, које нуде другачији тип складиштења.   
Са порастом броја информација и података који се деле преко интернета (*Big Data*) појавила се потреба за новим типом база података које ће бити прилагодљивије толикој количини података и њиховим разноврсним, неструктуираним типовима. Тако се дошло на идеју *NoSQL* база података, које нуде неколико нових начина складиштења и извршавања упита. Њих постоји неколико врста, од које је свака настала са посебном наменом, и које полако постају популарне широм индустрије.   
Циљ рада је да се наведени проблем са традиционалним релационим базама у овом случају, пренесе у *NoSQL* складиште података и да се истражи да ли нам пружа оптималније решење од овога до сада понуђеног.

# NоSQL базе података

Релационе базе података имају вишегодишњу примену у свим могућим гранама индустрије, а њихов екосистем се ослања на широк опсег корисних алата као и велики број људи који је обучен за обављање послова везаних за њихову имплементацију и одржавање.

Међутим, информационе технологије су у константној експанзији и развоју па се стога компанијама нуде нека иновативнија решења која су више прилагођена њиховим потребама. Неки се на овај корак одлучују како би једноставно могли да складиште компликованије податке или како би проширили своје складишне капацитете (узрок томе је појава *Big Data*), док друге води жеља ка уштеди новца који одвајају на скуп софтвер и хардвер релационих база података. Наравно, постоје и они који једноставно желе да иду у корак са временом и да прихвате агилније методологије развоја.

Како би смо ово боље представили могу се дефинисати следећи разлози: [1]

* Све чешће се појављују апликације и програми који користе и креирају огромне количине структуираних, полу-структуираних, неструктуираних и полиморфних података
* Агилно програмирање се све чешће користи у пракси, па се стога и функционалности и захтеви проширују
* Прелазак на *open source* и *cloud* сервисе постаје тренд
* Апликације су постале сервиси и неопходно је да буду доступне константно

Термин *NoSQL* је настао 2009. године и обједињује све базе података које не следе основне принципе релационих база података. Заправо, сам термин *NoSQL* база није прецизно дефинисан и представља класу производа и колекцију различитих концепата о начинима манипулације и складиштења података који се разликују од традиционалних релационих база.

Разлог појаве и почетка развоја концепта *NoSQL* база података је огромна експанзија интернета која је проузроковала проблем великих количина података (*Big Data)* као и све већи број корисника и апликација који их генеришу и приступају им. Са овим променама од релационих база података су све више захтевани високи нивои скалабилности и расположивости, што оне у овом случају нису могле у потпуности да испуне. Тако се јавила потреба за развојем новог, прикладнијег решења.[1]

## Карактеристике NoSQL база података

Постоји неколико кључних разлика када повлачимо линију између *NoSQL* база и релационих база података, а то су флексибилнији модели података, боља скалабилност и боље пеформансе. Оно што такође одваја *NoSQL* базе јесте чињеница да не поседују неке од основних карактеристика које су учиниле релационе базе толико општеприхваћеним и које их чине скоро незаменљивим већ дуги низ година. То укључује употребу изражајног упитног језика, секундарне индексе и изразиту доследност. [2]

Од *NoSQL* база података се и даље захтева да поседују *ACID(Atomicity, Consistency, Isolation, Durability)* особине, међутим код система који складишти дељене податке долази до појаве *CAP (Consistency, Availability, Partition tolerance)* теореме која нам говори да услови конзистентности, расположивости и толеранције раздвојености се ни у ком случају не могу истовремено задовољити.

-Конзистентност захтева да се кориснику пружају најновије верзије података при сваком упиту. Постиже се ажурирањем неколико чворова пре наредног читања.

-Расположивост подразумева да база података увек буде доступна корисницима у гарантованим временским оквирима (постиже се великим бројем чворова и репликацијом података). Постиже се репликацијом података кроз мрежу.

-Толеранција раздвојености захтева да ниједан скуп отказа, осим потпуног отказивања не сме да произведе неисправан одзив система базе података.

Дакле, уколико постоји раздвојен систем (неколико партиција на неколико различитих локација) и кроз њега се шаље порука постоје три могућа догађаја уколико порука не стигне на одредиште: [3]

1. трансакција се поништава јер систем не прихвата раздвојеност система
2. трансакција се наставља али систем није козистентан
3. транскација чека на успешно слање поруке, систем није расположив

Компромиси који се могу начинити како би се разрешио овај проблем су следећи:

-Одбацивање прихватања раздвојености– на овај начин обезбеђујемо конзистентност и расположивост, али је комплетан систем на једном рачунару и веома је скупо решење

-Одбацивање расположивости – обезбеђујемо конзистентност и толеранцију раздвојености, али се не гарантује време одзива.

-Одбацивање конзистентности – Обезбеђују се расположивост и толеранција раздвојености, али се не гарантује читање најновије верзије података.

Ово суштински нарушава *ACID* особине, стога је дефинисан нови скуп особина под називом *BASE* особине (*Basically Available, Soft estate, Eventually consistent)*.

Дефинисање *BASE* особина:

1. Суштински расположива – већина података је доступна већину времена
2. Неконзистентно стање – база података не мора бити конзистентна у сваком тренутку
3. Конвергентна конзистенција – додатком новог чвора у систем садржај се може реплицирати на њега, не постоји гаранција да ће у сваком тренутку сви чворови садржати идентичне копије података

*MapReduce* алгоритам омогућава управљање великим количинама података посредством дистрибуиране мреже чворова. Његова сврха је да дели задатак на мање делове, састоји се из два корака:

*Map* корак – главни чвор преузима улазне податке, дели их на мање делове и дистрибуира у радне чворове. Радни чворови процесирају додељене задатке и враћају одговор главном чвору.

*Reduce* корак – главни чвор преузима резултате радних чворова и комбинује их са циљем добијања траженог резултата. [4]

Особине и предности:

* Не поседују формалну специфицирану шему базе података  
  Може да садржи и неструктуиране и полуструктуиране податке
* Не ослањају се на релациони модел података (нема *Join* операција)
* Унапређене перформансе у раду са великом количином података
* Хоризонтална скалабилност. Додавањем чворова у дистрибуирану мрежу се повећавају капацитет и перформансе
* Архитектура локализованих ресурса – сваки сервер се ослања на локалну меморију
* Партиционисање базе података – укупан број записа је дистрибуиран у партиције, а партиције се реплицирају
* Асинхрона репликација – подаци нису реплицирани оног момента када су записани
* Избегавање објектно-релационог мапирања – подаци се чувају у посебним структурама података
* Често представља систем за управљање датотекама
* Толеранција на отказе
* Даје предност перформансама на штету конзистентности
* Приступ путем АПИ-ја – не ослања се на SQL
* Доминантно отвореног кода

Недостаци:

* Непостојање стандарда
* Непостојање стандардног упитног језика – UnQL (*Unstructured Query Language)*

## Подела NoSQL база података

Иако постоји много различитих нерелационих база података, по типу података које складиште их можемо поделити у следеће категорије:

-Базе података оријентисане ка документима

Главна карактеристика ових база која их разликује од релационих је складиштење података у виду засебних докумената, док се у релационим базама подаци складиште по табелама (редови и колоне). Формат који ове базе користе за складиштење својих докумената је *JSON* (*JavaScript Object Notation*) стога сваки документ у суштини представља објекат као у објектно оријентисаном програмирању, па је овај формат популаран код већине развијача апликација.

Ови документи су конструисани тако да се уместо смештања података по различитим међусобно повезаним табелама, они смештају унутар једне целине у виду индексираних поља и подпоља којима се једноставно може приступити. С обзиром на то да су сви подаци садржани у једном објекту, операције попут спајања табела и сложене трансакције могу да се избегну, што нам умногоме олакшава посао.

Шема овог типа базе је динамична, односно сваки објекат у шеми може имати неодређени број различитих поља што је изразито погодно за развој апликација које се надограђују у кратким временским роковима јер на овај начин подржавају неструктуиране и полиморфне податке. Такође пружају и робусност упита који су уобичајени у релационим базама, стога омогућују извршавање упита на основу било које комбинације поља унутар документа.

Што се тиче упита, они се могу извршавати над било којим пољем унутар документа. *MongoDB* нам на пример пружа богате могућности индексирања (текстуални индекси, геолокациони индекси, јединствени индекси...) што помаже при оптимизацији упита.Такође је присутан *Aggregation Framework* за пружање анализа у реалном времену (попут функционалности *GROUP BY* унутар *SQL*-a), и нативно уграђен MapReduce алгоритам за друге типове софистицираних анализа.

Примена: примена овог типа база је општег типа, дакле примењује се у многим различитим типова апликација јер пружа могућност извршавања упита над различитим пољима документа. [1][2]

Примери: *MongoDB, CouchDB.*

-Базе података оријентисане ка графовима

У овом типу база података се подаци моделују на основу мреже веза између одређених елемената, док се упити извршавају на основу структуре графова са њиховим чворовима и својствима.

Доступни су и упитни модел где се везе између елемената могу испитати како би се добила дирекнта и индиректна преклапања података у систему. Овај тип база је оптимизован само за ову врсту упита, над графовима, док су друге врсте упита мање ефикасне.

Користе се углавном у случајевима где су везе између објеката кључне за саму апликацију, као што су друштвене мреже, ланци снабдевања и мрежне топологије.[1][2]

Примери: *Neo4j, Titan*.

-Базе података типа кључ-вредност

Оне представљају основни тип нерелационих база података где се свака ставка смешта у базу као кључ заједно са својом вредношћу. Саме вредности су нечитљиве унутар система и подаци се могу добити само путем кључа, а пошто база не прати стандардне шеме могу се користити неструктуирани и полиморфни подаци.

Претрага и ажурирање се извршавају на основу примарног кључа, док се корисницима препоручује креирање сопствених индекса за извршавање специфичнијих упита. Извршавање операција над овом базом није могуће одрадити јединственим упитом, стога је неопходно одрадити неколико (проналажење, ажурирање, ажурирање индекса) упита.

Овај тип база података се примењује у специфичним ситуацијама које траже податке само на основу примарног кључа. Једноставност података који се смештају унутар ње, пружају бази изразите могућности проширења капацитета и побољшања перформанси.[1][2]

Примери: *Riak, Cassandra.*

-Базе података оријентисано ка колонама

Уместо смештања података у редове, ове базе података су дизајниране за складиштење табеларних података у виду одвојених колона. Ово омогућује смештање огромних количина динамичних колона, а пошто називи колона и записи нису фиксни а број колона није ограничен, оне се могу посматрати попут дводимензионалних кључ-вредност база.

Поседују високе перформансе приликом извршавања упита што их чини погодним за *OLAP*, док је упис података нешто спорији па нису погодне за *OLTP*.

Поседују тродимензионалну структуру (идентификатор реда, фамилија колона и идентификатор колоне, временска ознака), а један у складишту је аналоган документу. Фамилије колона се дефинишу по принципу категорија колона у које се оне смештају, и оне играју улогу секундарног кључа.

Примену налазе, попут база кључ-вредност, у уском подручју апликација где је основни циљ претрага по примарном кључу, и такође поседују високе перформансе и скалабилност.

Примери: *HBase, BigTable*

## Упоређивање NoSQL база са релационим базама

|  |  |  |
| --- | --- | --- |
|  | SQL | NoSQL |
| Типови база | Један тип, уз одређене варијације | Неколико типова (кључ-вредност, документи, колоне, графови) |
| Историјат | Развијене `70-их година прошлог века, примењују се у свакој грани индустрије | Развој траје мање од 10 година, настале као потреба за већом скалабилношћу и складиштењем неструктуираних података |
| Примери | MySQL, Postgres, Oracle Database, Microsoft SQL Server | MongoDB, Cassandra, Hbase, Neo4j |
| Модел складиштења података | Записи се чувају као редови у табелама, где свака је колона предодређена за одређени тип податка. Постоји више табела које се међусобно спајају приликом извршавања сложених упита | Кључ-вредност базе имају сличан начин складиштења као и SQL базе, међутим постоје само две колоне (кључ и вредност). Базе за документа чувају сваки запис у виду јединственог објекта |
| Шеме | Структура и тип податка су унапред дефинисани. Како би се унео нови тип податка, база се скида са сервера и уносе се измене | Динамичне шеме, обично се са уносом записа креирају нова поља, мада се ипак поштују одређена правила |
| Скалабилност | Вертикална скалабилност која захтева сервер са могућношћу надоградње у циљу уношења све већег броја података. Постоји могућност распоређивања на неколико сервера, али то захтева додатне послове, при чему се губе неке од основних карактеристика | Хоризонтална скалабилност, омогућује додавање нових сервера, односно клауд сервиса |
| Развојни модел | Мешавина open-source и closed source система | Open-source |
| Манипулација подацима | Специфичан упитни језик (SQL) | Објектно оријентисани API |
| Конзистентност | Могу се оспособити за снажну конзистенцију | Зависно од базе до базе |

*Табела 2.1 Упоређивање каратеристика релационих и NoSQL база*

Како би се стекао утисак о могућностима различитих база, на конкретнијем нивоу од приказаног у претходно наведеној табели, пружамо и следећи увид:



*Табела 2.2 Упоређивање различитих типова NoSQL са релационм базама*

# MongoDB

*MongoDB* је *NoSQL* база података која није замишљена, као већина других, да одбаци карактеристике релационих база које су их учиниле незаменљивима последњих неколико деценија, већ представља следећи логичан корак у еволуцији складишта података. Заснована на такозваном *Nexus* (Чвориште) моделу, обједињује карактеристике класичних релационих база (изражајан упитни језик и секундарни индекси, добра конзистенција, солидна интеграција са већ постојећим пословним процесима) и карактеристике које модеран технолошки свет захтева (флексибилан модел података, скалабилност и перформансе, доступност широм света) што је чини једном од најпопуларнијих *NoSQL* база.

*MongoDB* такође поседује и флексибилну структуру за складиштење података, односно дозвољава истовремено коришћење неколико енџина (скуп алата) за складиштење истовремено, што представља огромну предност јер су на тај начин компаније у могућности да конфигуришу своје базе у складу са својим специфичним потребама. Комплексност развоја апликације и самих оперативних процеса се на овај начин значајно смањује, јер није потребно покретати неколико различитих решења за смештање података, већ се то самостално извршава на нивоу самих имплементираних енџина.

Уз базу (верзија 3.2) укључена су 4 енџина, тако да постоји могућност да се без већих потешкоћа они комбинују ради задовољења потреба апликације

Задовољење потреба апликације за овим типом разноврсности је омогућено извести без већих потешкоћа, јер постоје 4 енџина који су укључени уз базу:

1) *WiredTiger* – пружа грануларну контролу конкурентности и уграђену компресију, што значајно утиче на ефикасност.

2) *Encrypted* *storage* – само име нам говори да је задужен за енкрипцију осетљивих података (укључен је уз *MongoDB Enterprise Advanced*)

3) *In-Memory* – пружа захтевним апликацијама анализу у реалном времену и пружа добре перформансе

4) *MMAPv1* - ....

## Модел података – *JSON/BSON*

База је конструисана тако да се подаци складиште у формату *BSON*, који представља продужетак веома популарног формата *JSON*. Начин на који је ово остварено је следећи: корисник, односно програмер податке пакује у *JSON* и такав објекат шаље у базу, док се на пријемној страни он трансформише у *BSON* (најчешће како би се документу доделило идентификационо поље) и смешта као такав.

*JSON* (*JavaScript Notion Object*) представља формат за размену података чија је сврха да буде подједнако разумљив људима и машинама. Потпуно је независтан у односу на програмске језике, али је у потпуности компатибилан са скоро свим верзијама што га чини идеалним форматом..

Своју компатибилност остварује тиме што је заснован на две структуре података које сви програмски језици подржавају:

- Колекција парова име/вредност (у различитим језицима: објекат, запис, речник, асоцијативни низ)



*Слика 3.1 Парови име-вредност у JSON-у*

- Поређана листа вредности (у различитим језицима: низ, вектор, листа, секвенца)



*Слика 3.2 Листа вредности у JSON-у*

Типови података које *ЈСОН* подржава осим наведених су број (цели и децимални), *String* (низ знакова), *Boolean* (true/false вредности), низ (сортирани низ подржаних елемената) и вредност null. [5]

*BSON* (*Binary JSON*) у суштини представља проширени тип *JSON* -а. Садржи форму и поља идентичне као и *JSON*, међутим *BSON* подржава и додатне типове података који укључују *Date* (датум), *TimeStamp*(временска ознака) и *BinData* (низ битова).  
Особинa која га чини погоднијим у односу на *JSON* су ефикасност и искоришћеност меморије (у неким случајевима степен искоришћености варира услед дужине префикса). [6]

*JSON* је популаран међу програмерима због чињенице да његова структура највише подсећа на структуру објекта у објектно оријентисаним програмским језицима

Овако спаковане објекте посматрамо као документе, јер је то тип података за који је ова база намењена, док документе који садрже сличну структуру организујемо у колекције. Као успоредбу са релационим базама података можемо се послужити следећим објашњењем: „У односу на релационе базе података, колекције можемо посматрати као табеле, документе као редове, а поља као колоне“.

Предност коју *NoSQL* базе имају је та што уместо да податке смештају у неколико табела које је касније неопходно спајати како би се дошло до потпуног резултата упита, оне их смештају у много мањем броју колекција које се састоје од објеката и на којима је много лакше извршити упит због мање разуђености информација.[1][2]

## Модел упита

Како би се *MongoDB* приближио ширем спектру корисника, развијени су драјвери за све познатије програмске језике (*Java, Javascript, Python, Ruby…*), а у томе учествују и сами стручњаци који раде на овој бази и заједница програмера која је користи. За разлику од *SQL* језика, за комуницирање са базом се користи *API* програмског језика у који се смештају методи и функције.

Један од начина приступа бази представља и *Shell* *Query Sintax* (specifican samo za *MongoDB*), на бази *JavaScript*-а који је моћан, брз и помоћу њега се релативно лако управља целокупном базом.  
Сама флексибилност извршавања упита над базом је постигнута подршком за различите типове упита, које друге *NoSQL* базе обично одбацују. На овај начин се могу извршити сложенији упити, агрегације и секундарни индекси. [1]

Врсте упита које су подржане:

- Кључ-вредност упит

- Упит на основу опсега

- Геоспацијални упит

- Упит на основу текста

- *MapReduce* упит

- *Aggregation Framework* упит

Секундарни индекси које база подржава:

* Јединствени индекс
* Сложени индекс
* Индекс низа
* ТТL индекс
* Парцијални индекс
* Индекс за претрагу текста

# NoSQL у дигитализацији књига

Као што је већ наведено у уводу рада, проблем којим ћемо се бавити је смештање дигитализованих библиотекчких објеката у *NoSQL* базе података. Оне представљају релативно нов концепт, а рад настоји да прикаже њихову практичну примену на овом конкретном случају.

Након прегледа карактеристика различитих решења одабир је пао на *MongoDB,* базу података која је усмерена на складиштење докумената, због типа података које ове базе смештају, њиховог начина на који их смештају и извршавања упита.

У овом делу рада поставиће се пројектни захтеви које је неопходно извршити и даће се конкретан пример решења.

## Кориснички захтеви за апликацију

* Апликација ради као *web* сервис
* Почетни прозор нуди две опције: унос новог објекта, претрага постојећих објеката у бази
* Опција „Унос новог објекта“ отвара нови прозор где се од корисника тражи да унесе жељени објекат (PDF фајл), као и да унесе његове метаподатке
* Метаподаци обухватају: наслов, име и презиме аутора, годину издања, језик на коме је дело издано, штампарију, место у коме је дело издано, и колекцију којој објекат припада
* Опција „Претрага постојећих објеката“ отвара нови прозор где се нуде две опције за извршавање упита: по називу дела и по имену и презимену аутора
* Након извршења упита, добија се приказ резултата у *online* PDF форми

## Увод у практичну израду пројекта

Алати и технологије коришћени за израду пројекта:

* NetBeans IDE 8.1
* GlassFish Server 4.1.1
* MongoDB 3.2
* mongo-java-driver-3.3.0.jar

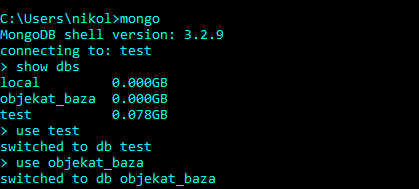
### Постављање и коришћење MongoDB-a

*MongoDB* је *open-source* пројекат који је доступан за све веће оперативне системе (*Windows, Linux, OSX, Solaris*), чија инсталација на систем је једноставна, за разлику од, на пример, *Oracle* бесплатне верзије базе података.

Након праћења једноставних упутстава за инсталацију, и постављања путање у систему (*System variable*), неопходно је креирати два фолдера: један за смештање података и други за *log*.

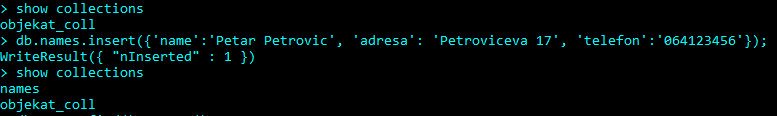
*MongoDB* у основи ради преко *Command Prompt*-а, и за то је неопходно отворити два прозора. У првом прозору покрећемо команду “mongod” како бисмо покренули сервер базе података, док се у другом повезујемо на тај исти сервер командом “mongo”. Уколико се база користи искључиво преко *Command Prompt*-а, током рада се користи само други прозор. Пошто је упитни језик заснован на *JavaScript*-у, углавном познатим командама се врло једноставно приступа различитим базама података, њиховим колекцијама, као и извршавање упита над њиховим записима.

Примери комуникације са базом:



*Слика 4.1 Повезивање са базом*

На претходној слици се види поступак од приступања серверу до избора жељене базе и промене на коришћење друге базе.



*Слика 4.2 Рад са колекцијама у бази*

Након што изаберемо базу, можемо приступити раду са њеним колекцијама и записима. На претходној слици се види да база у тренутку њеног покретања у себи садржи једну већ креирану колекцију под називом “*objekat\_coll*”. Креирање нове колекције захтева само унос новог записа у њу, без њеног претходног креирања. Након што се упит изврши, база ће сама додати нову колекцију.   
На претходној слици се јасно види практичан пример оваквог сценарија где се након уноса података о одређеној особи, у колекцију “*names*”, иста та колекција појавила у бази а да нисмо претходно задали никакав специфичан упит.

### GridFS

Одмах на почетку израде пројекта суочени смо са следећим проблемом: *MongoDB* коришћењем *BSON* –а ограничава величину записа који може да се унесе на величину од читавих 16 MB. Пошто пројекат захтева да се у базу смештају PDF фајлови који представљају електронске копије књига, чија величина у највећем броју случајева премашује дозвољених 16 MB, неопходно је користити уграђену спецификацију за рад са великим фајловима под називом *GridFS*.

Он представља алатку која штеди и време и ресурсе. Принцип рада је следећи: приликом креирања нове колекције, која ће садржавати у себи жељени фајл, *GridFS* ће приликом смештања направити две подколекције, *fs.chunks* и *fs.files* (где *fs* представља назив колекције дефинисан од стране корисника).

Подколекција *fs.files* у себи садржи само метаподатке фајла, смештене у *BSON*. Подаци који се налазе у овом запису обухватају осим назива дела и дужину, број записа на које је фајл „разбијен“, тачно време уписивања у базу...

Подколекција *fs.chunks* (пр: комади)у себи садржи делове фајла и сваки запис износи тачно 255 KB, осим последњег који је мањи (зависи од тачне величине фајла који се складишти). *GridFS* раздваја целокупан фајл на мање записе (255 KB), и додаје им заједнички *id* број који их повезује међусобно и са записом метаподатака.

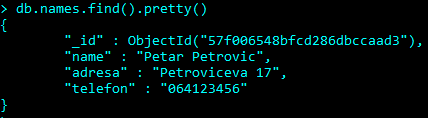
*GridFS* помоћу драјвера враћа кориснику жељени фајл тако што ће на основу *id* броја пронаћи све његове делове и затим их спојити на клијентској страни. Обављање ове процедуре ручно, без помоћи овог система би био изразито компликован и скуп по питању ресурса. То што се фајл тако разуђен смешта у базу значи да је за сада немогуће извршавати атомска ажурирања самог његовог садржаја (и даље не постоји уграђен механизам) тако да је у том случају неопходно правити његову копију, и водити рачуна да се из базе извлачи најновија верзија фајла.

Међутим предност овог начина складиштења је та што је, када је то пожељно, омогућен приступ само жељеним деловима фајла, уместо ишчитавања читавог фајла из базе. Такође, у овом погледу, могуће је чак и дефинисати величину „комада“ тако да задовољава потребе апликације. [7] [8]

### Модел података

Као што је већ објашњено, *MongoDB* за смештање података у своју базу користи *BSON* формат, који је дериват *JSON* формата, и који поседује додатна поља која су врло корисна.

На следећој слици се види практичан пример овог поступка конверзије.



*Слика 4.3 Упит за претрагу базе*

Јасно види да се у бази налази *BSON* формат податка пошто поседује поље “\_*id* ”, иако током ручног уноса записа (слика број), у виду *JSON* формата, то поље нисмо декларисали. Ово додатно поље у себи садржи хексадецимални стринг од 24 карактера, односно величине 12 бајта. [10]  
Ових 12 бајта су распоређени на следећи начин:

* 4 бајта представљају време креирања записа
* 3 бајта представљају идентификатор машине на којој је запис направљен
* 2 бајта представљају *id* процеса
* 3 бајта представљају бројач који почиње насумичном вредношћу

Као што је у уводном делу описано, коришћењем релационих база добили бисмо неколико међусобно повезаних табела за један унешен фајл, док помоћу *NoSQL* –а те табеле сводимо на један документ.

У нашем случају, моделовање целокупног документа није потребно јер то на себе преузима *GridFS.* Приликом креирања записа назив се повлачи аутоматски са унешеног PDF-а, а корисник уноси само метаподатке који се записују у виду поддокумента. Остала поља су препуштена аутоматизму.

## Практична израда

Основа израде апликационог дела пројекта је била MVC архитектура, међутим појавом потребе за коришћењем *GridFS* –а ова архитектура је донекле нарушена.  
Структурални садржај апликације је следећи: JSP странице за комуникацију са корисницима, DBBroker класа која је задужена искључиво за рад са базом, сервлети који представљају спону између JSP страна и DBBrokera.

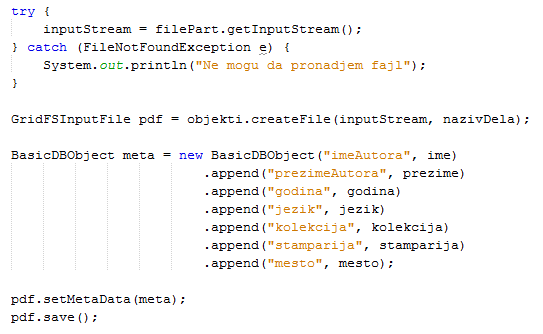
### Унос фајла у базу

Након што корисник на клијентској страни апликације одабере жељени фајл и унесе неопходне метаподатке, сервлет ће примити захтев и проследити га брокеру који остварује конекцију са базом и смешта жељени фајл у њу.



*Слика 4.4 Повезивање са базом и колекцијом*

Преко монго клијента приступамо жељеној бази, а затим креирамо *GridFS* помоћу кога бирамо колекцију унутар базе којој желимо да приступимо. Уколико колекција не постоји, база ће је аутоматски креирати.

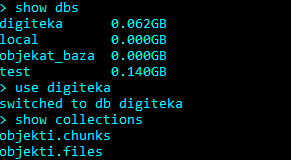


*Слика 4.5 Уношење фајла у базу*

Затим се фајл прослеђен од стране корисника прихвата и помоћу њега се креира *GridFSInputFile* који омогућује раније описано смештање у базу. С обзиром на то да у овој врсти записа метаподаци представљају поддокумент, креира се нови објекат у који се смештају неопходна поља и њихове вредности. Помоћу метода *.setMetaData* додаје се објекат са метаподацима у већ креиран фајл, и сада је спреман за уписивање.

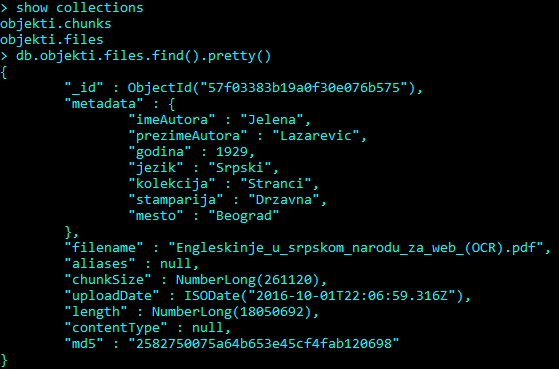
Ради провере исправности уписа, помоћу *CommandPrompt*-а приступамо бази где се претрагом колекција види да су уместо једне, креиране две различите колекције:

“objekti.chunks” и “objekti.files”.



*Слика 4.6 Преглед колекција у бази*

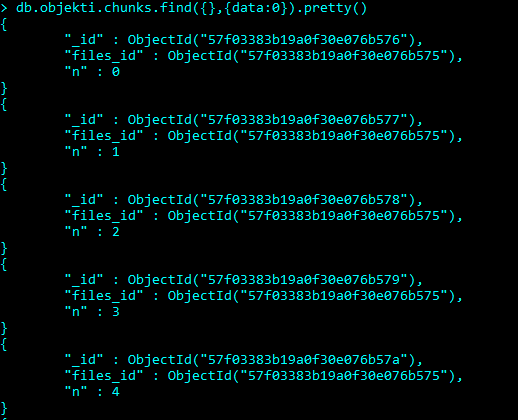
Колекција “objekti.chunks” у себи садржи „раскомадан“ фајл који је корисник унео, док у колекцији “objekti.files” можемо пронаћи метаподатке.



*Слика 4.7 Преглед новог записа у колекцији*

На овој слици видимо формат у коме је унешени фајл смештен у базу. Аутоматски креирано поље “*\_id”* које представља јединствени идентификатор (одлика *BSON* –а), поддокумент “*metadata*” који у себи садржи податке унешене од стране корисника, “*filename*” који је *GridFS* извукао из документа, “*chunkSize*” који представља величину појединачног “комада”, “*uploadDate*” где се бележи тачно време уноса записа, “*length*” исписује дужину фајла у бајтовима, “*md5*” представља хеш вредност фајла. [9]

Затим ћемо претражити колекцију “objekti.chunks” како би се стекао увид шта се у њој налази. Пошто се обичном претрагом добија резултат нечитљив људском оку, у упит се убацује захтев да се одеђени подаци не приказују. На овај начин се добијају само за нас релевантне информације.

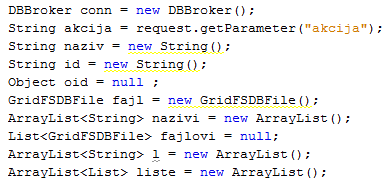


*Слика 4.8 Преглед колекције “objekti.chunks”*

Овде можемо приметити следећа поља: “*\_id* “ који се генерише за сваки запис засебно, “*files\_id* ” помоћу којег се комади везују за главни запис који се налази у колекцији “objekat.files” (приметно је да се број понавља за записе истог фајла), и на крају поље “*n*” које представља редни број записа (почиње од 0).

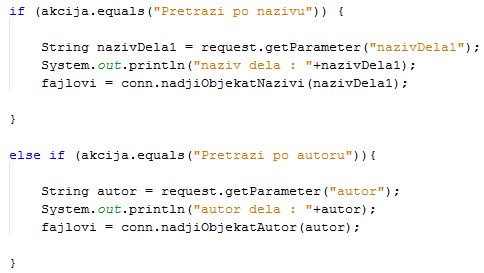
### Претрага базе и приказивање резултата

Након што је поступак уношења у базу завршен, корисник може приступити претрази колекције. Клијентски део апликације пружа две могућности: претрага преко назива фајла и претрага преко имена и презимена аутора.



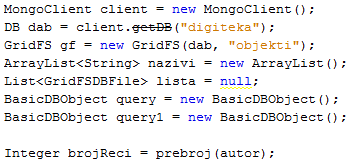
*Слика 4.9 Инстанцирање објеката и примање позива са странице*

Неопходно је инстанцирати алате који се користе, као и прихватити вредност захтева који је корисник послао: претрага по аутору или по називу.



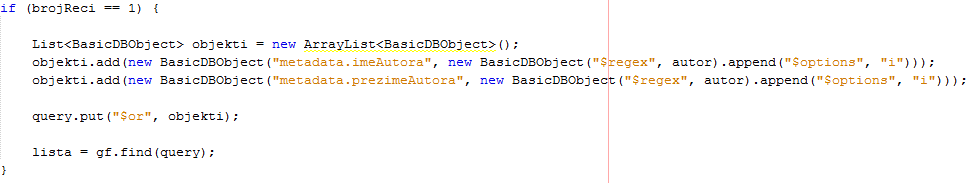
*Слика 4.10 Различите врсте претраге*

На основу вредности избора, сервлет позива неопходан метод у *DBBrokeru* како би испунио захтев. Пошто се методи извршавају на сличан начин, у раду ће бити приказан само метод претраге по аутору, јер је за нијансу компликованији.



*Слика 4.11 Повезивање са базом и одређивање броја речи*

Након инстанцирања неопходних објеката и приступања бази, неопходно је израчунати број речи у стрингу који је прослеђен (метода ”*prebroj*“ је ручно испрограмирана, није нативна за систем). Број речи је неопходно израчунати како би систем знао коју функцију да позове. Прво имамо пример када се прослеђује само једна реч.



*4.12 Извршавање упита са једним стрингом*

Када је претпостављени услов испуњен креира се нова листа, у коју се убацују два објекта. У овим објектима се налазе упити које ће систем извршити над базом.

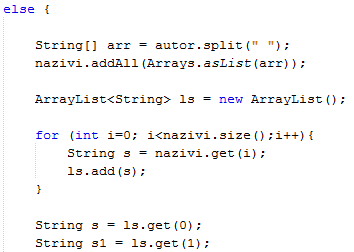
Први аргумент у објекту представља поље унутар документа који се прегледа у бази (“*metadata.imeAutora*“ означава поље где се налази име аутора у поддокументу *metadata*. Други аргумент представља нови објекат који се креира како би се унели специфични захтеви у упит. Ознака ”*$regex*” је у овом формату специфична за *MongoDB*, и пружа нам могућности да приликом претраге систем упореди да ли циљано поље у себи садржи послат запис, уместо класичног упоређивања где је неопходно послати тачну вредност.

Затим се додаје и помоћна ознака “*$options*” где се помоћу “*i*” даје захтев да се занемари упоређивање малих и великих слова.

Потом се ова листа додаје у претходно креиран објекат уз параметар “*$or*” који налаже да је неопходно да се послата вредност налази само у једном од два задата поља како би упит био успешан.

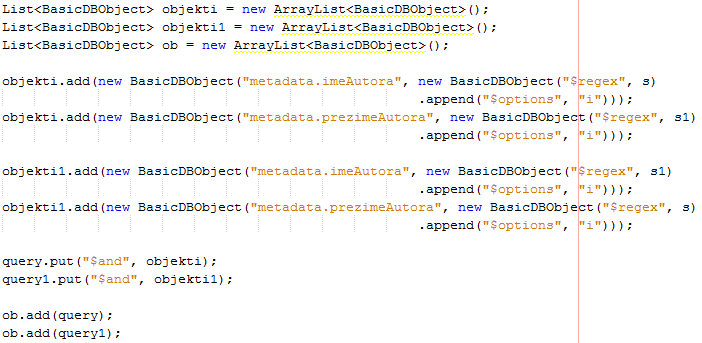
У листу резултата потом уписујемо вредност извршавања упита помоћу претходно креираног *GridFS*-а.

Уколико послата вредност у себи садржи више од једне речи, претходни део кода се прескаче, и прелаззи се на следећи део где се прво стринг раздваја на саставне делове и потом смешта у низ појединачних вредности.



*Слика 4.13 Разлагање сложеног стринга*

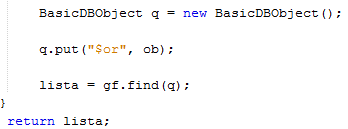
Затим се помоћу *for* петље ове вредности извлаче из овог низа и убацују у листу како би им се могло приступити. У овом случају се претпоставља да постоје две вредности у листи, једна за име и друга за презиме, стога креирамо два нова стринга која ћемо проследити у упиту.



*Слика 4.14 Креирање комплексног упита*

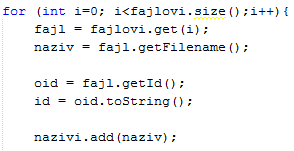
Сада је неопходно креирати неколико нових листа како би се сви ови захтеви могли сместити у упит. У једној листи се претпоставља да први стринг представља име аутора а други његово презиме, док се у другој претпоставља обрнута ситуација. Објекти са упитима се креирају идентично као у првом случају када смо имали једну реч, само што сада имамо више вредности, па је мало компликованије поставити решење.

Свака листа се смешта у засебан објекат, пошто сада имамо два упита унутар једног, додаје им се функција “*$and*” и коначно се додају у главну листу како би се упит извршио.



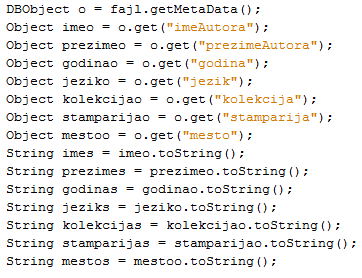
*Слика 4.15 Извршавање упита*

Креира се нови објекат у који се смешта претходно дефинисана листа уз поновно коришћење функције “*$or*” како би се задовољио услов да је само неопходно да је само један део упита исправан, и помоћу *GridFS*-а опет извршавамо упит и резултате смештамо у листу.  
Коначна листа са резултатима упита се прослеђује назад сервлету како би се испунио корисников захтев.



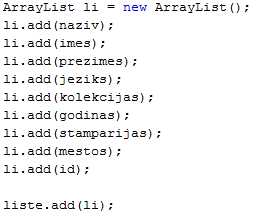
*Слика 4.16 Извлачење појединачног фајла из листе резултата*

Након што се листа пошаље у сервлет (одакле је позвана метода претраге), извлачимо засебне “*filename*” записе и додајемо их у листу наслова. Такође се извлачи и “*\_id*” поље, за касније потребе.



*Слика 4.17 Издвајање метаподатака из фајла*

Потом се креира објекат у који се смештају метаподаци засебног фајла, који се затим методама “*get*” уписују у нове објекте који се затим пребацују у стрингове.



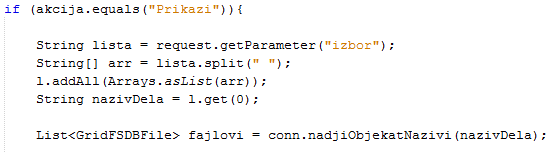
*Слика 4.18 Додавање метаподатака у листу*

Затим се креира нова листа са извученим метаподацима, и уписује се у укупну листу различитих листа метаподатака за различите фајлове.

Ове листе се потом шаљу назад на клијентску страну где се на JSP страници оне опет распакују како би кориснику приказале жељене податке који обухватају назив фајла и његове метаподатке.  
Сада се кориснику нуди нови избор, приказивање фајла на веб страници или ажурирање његових метаподака.

### Приказивање фајла

Након што корисник одабере жељени фајл и позове функцију да се он прикаже, JSP страна шаље сервлету вредност избора, и прослеђује листу параметара изабраног објекта.



*Слика 4.19 Позивање метода за претрагу базе*

Ова листа се опет раздваја на саставне делове, одакле се извлачи назив фајла.   
Након тога се креира листа која у себи садржи *GridFSDBFile* и у њу се смешта резултат метода *nadjiObjekatNazivi* чији је параметар претраге претходно извучен назив жељеног фајла.

Метод *nadjiObjekatNazivi* нећемо описивати у овом раду, јер је веома сличан методу *nadjiObjekatAutor* који је објашњен у претходном поглављу.



*Слика 4.20 Приказ PDF фајла на веб страници*

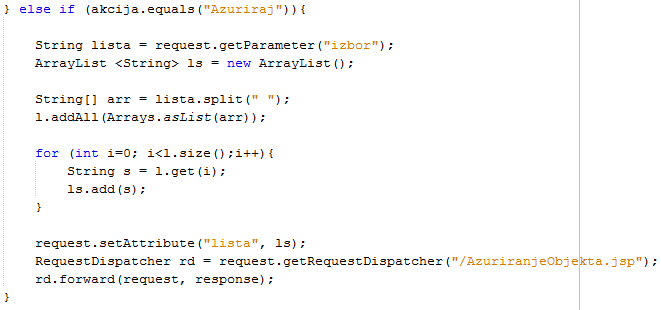
Након што добијемо листу резултата, помоћу *for* петље извлачимо појединачне резултате (у овом случају имамо само један резултат) како би их уписали у *GridFSDBFile* који садржи податке из колекција *objekat.files* и *objekat.chunks.*

Како би се фајл могао учитати директно на веб страници, неопходно је креирати *Java* фајл који је могуће привремено сачувати на корисничком рачунару.

Након чувања фајла на дефинисаном месту на рачунару, припрема се веб страница како би била оптимизована за приказ PDF формата и фајл се са локалног рачунара шаље на JSP. Након што се фајл учита на страници, систем ће се очистити, затворити методе за слање фајла и обрисати сам фајл са корисничког рачунара пошто је он неопходан само ради његовог учитавања на JSP страну и има функцију *temp* фајла.

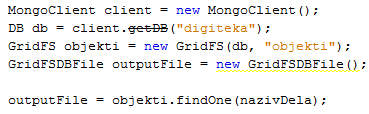
### Ажурирање фајла

Уколико се корисник одлучи на опцију ажурирања, након одабира жељеног фајла и кликтања на жељено дугме, низ метаподатака се опет прослеђује сервлету на обраду.



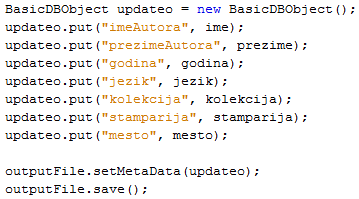
*Слика 4.21 Прослеђивање метаподатака на JSP страницу*

Сервлет ће примити захтев и низ ће трансформисати у листу како би је послао JSP страни задуженој за ажурирање датотеке. Листа се распакује на клијентској страни, и кориснику се пружа могућност измене жељених поља. Након измене и позива функције за ажурирање, сервлет прима послате вредности и позива метод за ажурирање са примљеним вредностима као параметрима.



*Слика 4.22 Повезивање са базом и упит за претрагу*

Креира се нова конекција ка бази, помоћу *GridFS*-а се бира колекција, и креира се нови *GridFSDBFile* који ће нам омогућити ажурирање објекта.  
Како би се објекат ажурирао неопходно је прво пронаћи га у бази и сместити у креирани *GridFSDBFile.*

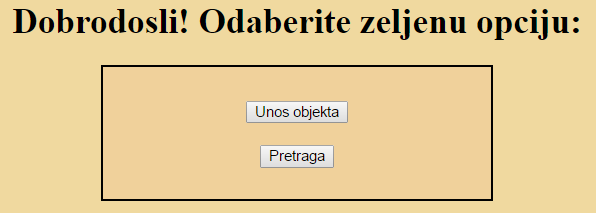


*Слика 4.23 Унос метаподатака и ажурирање фајла*

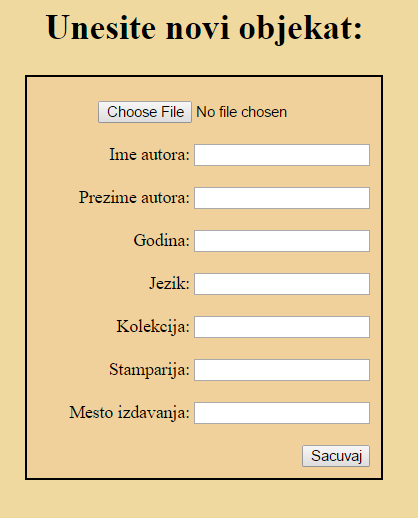
Затим се у нови објекат убацују прослеђени метаподаци, где први параметар представља поље унутар поддокумента *metadata* док други параметар представља нову вредност.

Овакав објекат се прво смешта у *GridFSDBFile* методом *setMetaData*, и затим се једноставним позивом објекат ажурира у бази.

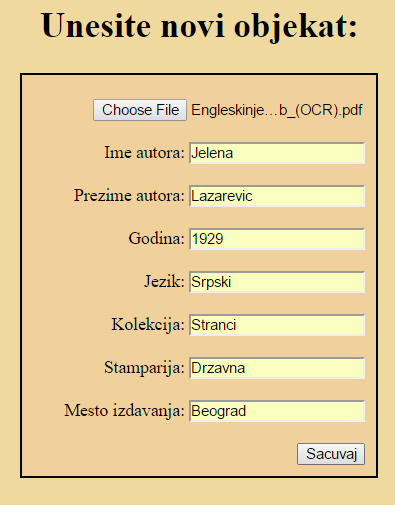
## Корисничко окружење



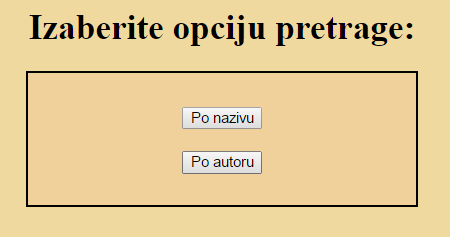
*Слика 4.24 Почетна страница апликације*



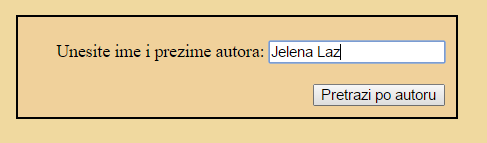
*Слика 4.25 Прозор за унос новог фајла*



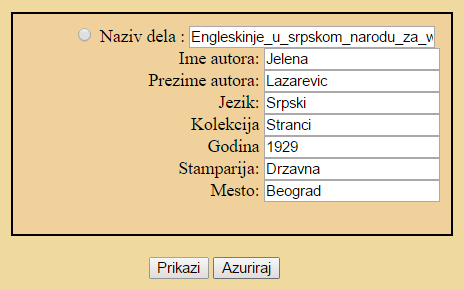
*Слика 4.26 Убачен фајл са унешеним метаподацима*



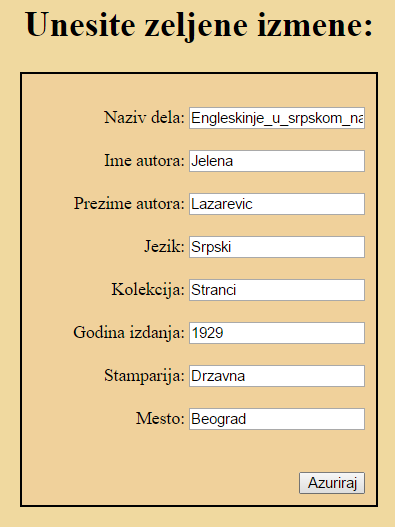
*Слика 4.27 Понуђене опције претраге*



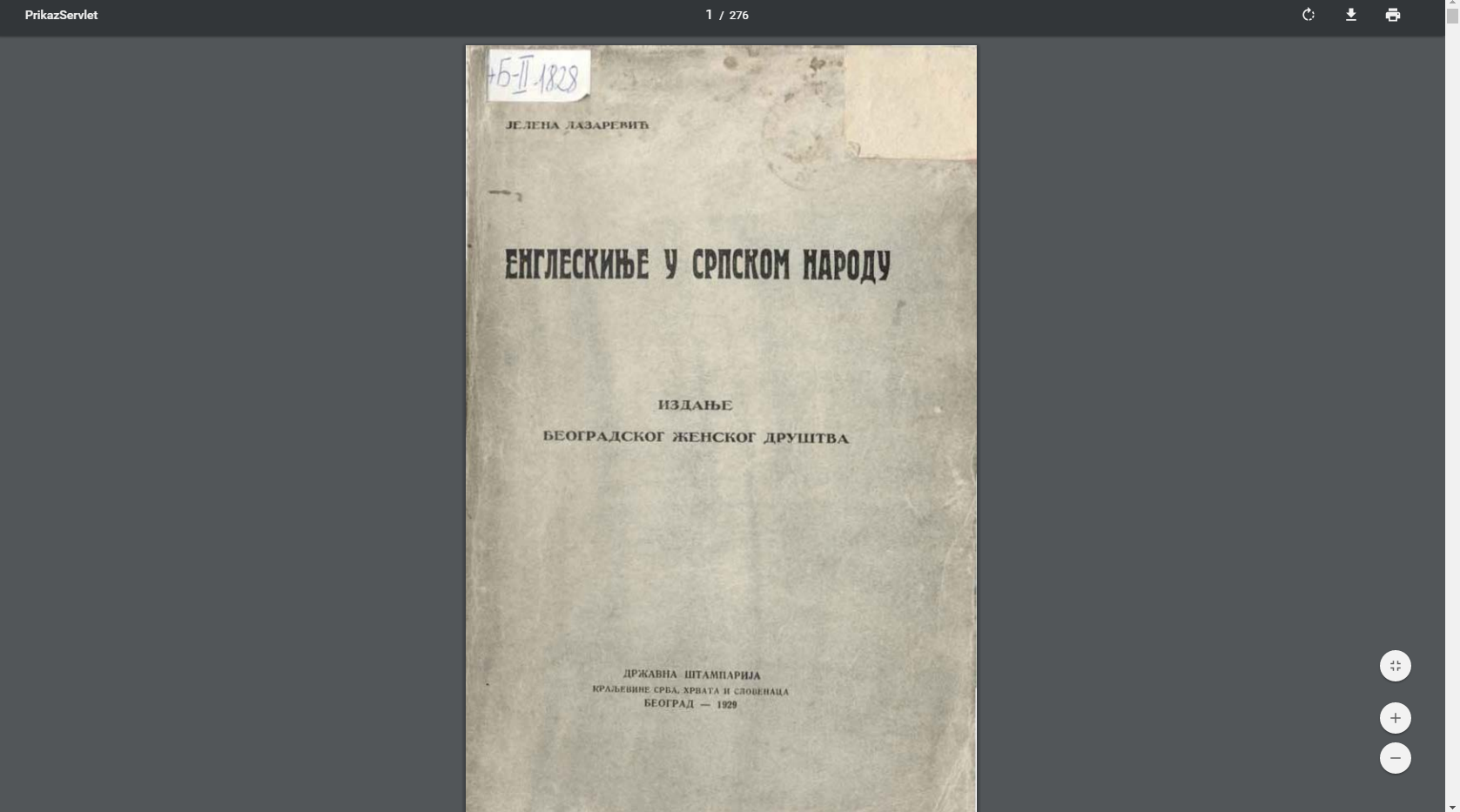
*Слика 4.28 Претрага по аутору (некомплетирано презиме)*



*Слика 4.29 Приказ резултата претраге*



*Слика 4.30 Унос измена*



*Слика 4.31 Приказ PDF фајла на веб страници*

# Закључак

Коришћење база података представља основу било каквог пословног система, у коју год привредну грану да спада, и та пракса траје већ неколико деценија. Са појавом рачунара појавили су се и документи у дигиталном облику, и као такве негде их је било потребно сместити. Простија времена, у технолошком смислу, су пружала могућност креирања технолошки конзервативнијих система који су креирани са унапред зацртаном наменом. Релационе базе података су тај посао обављале беспрекорно дуги низ година, међутим убрзаним напретком технологије долази и до њене разноврсности. Та разноврсност технолошких услуга које су доступне свима, од пословних људи до обичних грађана који услуге користе зарад сопственог задовољства, неминовно доводи и до константног повећања обима докумената које је неопходно негде ускладиштити. Такође, услед свега тога појавиле су се нове врсте података, на које креатори релационих база података нису рачунали. Неструктуирани, полуструктуирани, структуирани подаци који обухватају све могуће видове података, од текстуалних докумената, података о корисницима, видео и аудио фајлова, раличитих графова...Заиста велика разноврсност. Константно увећавање њиховог броја није ни најмање занемариво и то је почело да представља проблем. Релационе базе података су креиране са фиксним пољима и табелама и њихов капацитет је некада проблематично повећати.  
Тада су се појавиле *NoSQL* базе података као пандан релационим. Њихови капацитети су лако прошириви, лако се инсталирају и способне су да приме најразноврсније податке. Углавном су неструктуиране, што ће рећи да се врло лако прилагођавају пословним потребама било које врсте. Не постоји класична шема базе која у себи садржи унапред дефинисане редове, колоне и табеле у које је тешко унети измене уколико је база у употреби. Наравно, како би се одржао неки ред и поредак шема се мора осмислити и поставити, али она ни најмање није толико фиксна као што је то случај у релационим базама. Тренутно постоји неколико врста ових база за различите намене.

Овај рад је имао за циљ да покуша да имплементира једну такву базу у систем за похрањивање и ишчитавање PDF фајлова. У питању је дигитална библиотека која је остварена као веб сервис, и која садржи функције уноса новог фајла, претраге базе и ишчитавања фајла на веб страници. Од неколико врста *NoSQL* база изабрана је *MongoDB*, а разлог је то што је оријентисана према складиштењу докумената и у себи садржи методе за смештање великих фајлова. Врло је популарна из још једног веома битног разлога: компатибилна је за рад са свим популарнијим програмским језицима. Њен упитни језик је заснован на *JavaScriptu*, али драјвере који су неопходни за рад са другим језицима константно унапређује и тим који стоји иза ове базе података, као и корисници који желе да олакшају посао и себи и другима.

За потребе овог рада база се показала као веома добар избор. Манипулација са њеним шемама, колекцијама и подацима је једноставна, поседује уграђен систем за складиштење великих фајлова који је нама био неопходан, а уписивање, читање и ажурирање се брзо извршава. Документи који су складиштени су PDF формата и база је одвојено складиштила метаподатке и делове самог фајла, који су били повезани индексом. Иако је сваки фајл био „разбијен“ на више мањих делова, добро индексирање је омогућавало веома брзо ишчитавање самих фајлова из базе.

Основна предност у односу на релационе базе је та што дозвољава чување свих кориснику битних података у једном документу, уместо у неколико табела где је неопходно дефинисати повезивање, спољне кључеве и различите тригере и процедуре како би база радила без проблема.

Поставља се питање да ли је исплативо коришћење *NoSQL* база у домену овог проблема. Одговор је врло једноставан, и он гласи: да.   
Поред тога што су сви битни подаци смештени у један једини документ, њихови капацитети се врло лако проширују. С обзиром на то да се број дигиталних копија књига сваког дана све више умножава, карактеристика ових база да се једноставно и јефтино проширује њихов капацитет је добродошла.

У Србији свака већа библиотека поседује свој онлајн дигиталну библиотеку која је посвећена очувању и заштити старе и ретке књиге. Та врста књига углавном има листове различитих жућкастих нијанси и самим тим се „тежина“ фајлова значајно увећава јер једна књига може да заузима од неколико десетина до неколико стотина мегабајта, и врло је јасно да ће капацитети складиштења у будућности представљати проблем.  
Осим тога културна грађа коју је неопходно заштитити нису само књиге, већ у њу спадају и фотографије, разгледнице, мапе, аудио и видео записи...Ова разноврсност података не представља никакав проблем за *NoSQL* базе и то је такође једна од битнијих карактеристика које треба имати у виду приликом процене.

*NoSQL* базе података су релативно нов концепт, није ни пуну деценију стар и константно наставља да се развија. Промене и побољшања која ће их унапредити ће бити све бројнија и све чешћа, а тако ће се повећати и њихова погодност за најразличитије потребе складиштења података. Уколико тренутно нису конкурентне у неким сферама послова врло ускоро ће то постати, а до тада ће релационе базе података и даље остати поуздан избор за многе кориснике.  
Да ли ће *NoSQL* базе у потпуности заменити релационе? Неће, бар не у скорије време. Релационе базе су и даље неприкосновене у многим пословним системима који не генеришу неструктуиране податке, док су модерне *NoSQL* базе све више у примени у пословним процесима где су управо такви подаци доминантни. Оне и нису створене са идејом да комплетно замене релационе базе, већ да буду прилагођене потребама корисника модернијег и хаотичнијег света технологија и да као такве пруже могућност избора.

# Литература

[1] MongoDB Architecture guide, New York, 2016.  
 Преузето са веб странице <https://www.mongodb.com/white-papers>, 20.9.2016.

[2] Top 5 considerations when evaluating NoSQL Databases, New York, 2016.  
 Преузето са веб странице <https://www.mongodb.com/white-papers>, 20.9.2016.

[3] <https://dzone.com/articles/better-explaining-cap-theorem>, преузето 20.9.2016.

[4] <https://www.tutorialspoint.com/map_reduce/map_reduce_algorithm.htm>, преузето 21.9.2016.

[5] <http://json.org/>, преузето 27.9.2016.

[6] <http://bsonspec.org/>, преузето 27.9.2016.

[7] <https://www.compose.com/articles/gridfs-and-mongodb-pros-and-cons/>, преузето 27.9.2016.

[8] <https://docs.mongodb.com/manual/core/gridfs/>, преузето 28.9.2016.

[9] <http://api.mongodb.com/python/current/api/gridfs/grid_file.html>, преузето 28.9.2016.

[10] <https://docs.mongodb.com/manual/reference/method/ObjectId/>, преузето 28.9.2016.