Универзитет у Београду
Математички факултет

Мастер рад

NoSQL базе података докумената - Apache CouchDB

Ментор:
др Гордана Павловић-Лажетић

Студент:
Сара Попадић

Београд, 2015.г.

Ментор: др Гордана Павловић-Лажетић
редовни професор

Чланови комисије: др Саша Малков
доцент

др Јелена Граовац
доцент

Датум одбране рада: _____

Django[1] је можда изграђен за Веб, али CouchDB је изграђен од Веба. Никада нисам видео софтвер који комплетно обухвата философију која се налази иза HTTP-а. У поређењу са CouchDB-ом Django је стара школа баш као што у поређењу са Django-ом ASP делује застарело.

Jacob Kaplan-Moss

Django may be built for the Web, but CouchDB is built of the Web. I've never seen software that so completely embraces the philosophies behind HTTP. CouchDB makes Django look old-school in the same way that Django makes ASP look outdated.

Jacob Kaplan-Moss, Django developer

[1] Django је бесплатни оквир отвореног кода за веб апликације, чији је примарни циљ да олакша прављење сложених веб сајтова који раде са базама података

Садржај

1. Увод

Интерактивне веб апликације су се драматично промениле у току последње деценије, као и потребе за управљањем подацима у тим апликацијама. Велики подаци (енгл. *Big Data*), велики број корисника (енгл. *Big Users*) и дистрибуирано израчунавање (израчунавање у облаку, енгл. *Cloud Computing*) чине три глобална тренда који воде ка прихватању нове, *NoSQL* технологије складиштења и управљања дистрибуираним подацима. *NoSQL* технологија је често успешна алтернатива релационим базама података, посебно када је потребно обезбедити адекватно димензионисање система, коришћење разноврсних типова података и високу ефикасност уз ниске трошкове одржавања конзистентности (друштвене мреже, документне базе података и сл.)

Посебно значајна врста *NoSQL* система су базе података докумената. Оне се баве управљањем и складиштењем докумената и њихових верзија на хиљадама сервера на Интернету, које користе веб апликације. Релационе базе су у таквим условима споре и скупе, док је *NoSQL* ефикаснији и јефтинији начин за управљање таквим документима.

У раду ће бити приказани *NoSQL* системи за управљање документним базама података, формати у којима се складиште подаци, упитни језици које користе, модели израчунавања и програмски интерфејси. Ова технологија ће бити илустрована развојем веб апликације коришћењем *CouchDB* базе података отвореног кода компаније *Apache*.

1.1. Основни појмови у вези са базама података

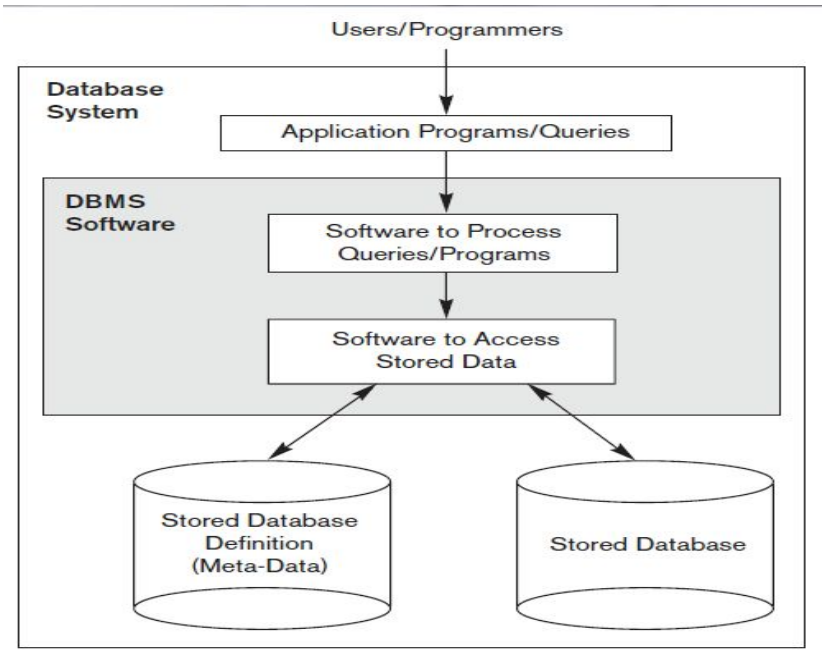
Базе података и системи за управљање базама података су у данашњем друштву јако битни, јер већина нас више пута дневно користи базе података. На пример, при подизању или уплаћивању новца у банци, при резервацији хотелске собе или авионске карте, при тражењу књиге у каталогу електронске библиотеке. Техника претраживања коју користе базе података се примењује при претраживању информација које се налазе на Вебу, коришћењем прегледача [13].

Ако бисмо желели некоме да опишемо шта је база података могли бисмо рећи да је **база података** организован и уређен скуп међусобно повезаних података. Базе података омогућавају једноставно „складиштење“ података исте врсте, њихово једноставно претраживање те једноставно руковање подацима. Најједноставније речено база података је скуп организованих података које се односе на одређену тему, а које се једноставно могу прегледати, претраживати, мењати, сортирати, упоређивати. Базе података су постојале и пре масовне употребе рачунара, најчешће у виду папирне

документације, а појавом рачунара су добиле електронски облик. Базу података, по правилу, користи велики број корисника [13].

Приступ и коришћење података из базе података омогућено је програмима који се називају **системи за управљање базама података** (скраћено СУБП - *DBMS*, *DataBase Management System*). СУБП је софтверски систем опште намене који олакшава процесе дефинисања, прављења, управљања и дељења (дистрибуирања) података базе између различитих корисника и апликација. Опште карактеристике и функције СУБП су [15]:

1. самоописујућа природа (основна разлика између фајл-система и СУБП-а);
2. раздвојеност програма од података;
3. конторла редундантности података;
4. омогућавање дефинисања погледа над подацима (различити корисници могу дефинисати различите погледе над базом);
5. обезбеђивање безбедног приступа подацима у вишекорисничким системима (контолом конкурентности се омогућава да више косисника истовремено приступа истим подацима);
6. дељење података;
7. стара се о извршавању трансакција;
8. опоравак базе података;
9. управљање подацима;
10. управљање паралелним извршавањем трансакција;
11. заштита података од неауторизованих или злонамерних приступа;
12. поуздано паралелно коришћење заједничких података од стране више овлашћених корисника;



Слика 1: Скица рада са базама података

Дефинисање базе података укључује навођење типова података, структуре података и ограничења (енгл. *constraints*) над подацима који се чувају у бази података. СУБП чува саму дефиницују базе података. СУБП може да чува и мета-податке. **Мета-подаци** су „подаци о подацима" тј. описују податке којI су у бази. **Подаци** су регистроване чињенице. Подаци могу бити: текстуални, нумерички, графички, звучни. Обрадом података добијамо **информације**. Информација је податак који нам помаже у доношењу одлука. Када неки податак за неког има одређену вредност, тада тој особи представља информацију.

Типови података према структури могу бити [6]:

- структурирани подаци (енгл. *structured data*)
    - постоји схема која дефинише формат података и подаци стриктно задовољавају формат описан схемом. У пракси, очекује се да таква схема буде разрађена до потребног степена детаљности. Предност овог типа података је што су њихов унос, чување, анализирање и упити над њима поједностављени. Најчешће се за управљење структурираним подацима користи програмски језик *SQL* (до прве стандардизације назив *SQL* био је скраћениза од енгл. *Structured Query Language*, првом стандардизацијом језика то је усвојено за име језика).
    - пример: подаци у релационој бази података - све торке имају исти формат.
- неструктурирани подаци (енгл. *unstructured data*)
    - (а) постоји схема која дефинише формат података
        - типови података су, међутим, често „прешироки", недовољно детаљни са становишта семантике података
    - (б) не постоји схема која дефинише формат података
        - не постоји никакав уграђени формат података
            - могуће је да постоји некакав екстерни формат који описује семантику података, али он не припада структури
    - примери:
        - текстуални или мултимедијални документи као што су фотографије, видео снимци, веб странице, *pdf* фајлови, *PowerPoint* презентације, електронска пошта, блогови... – подаци су неструктурирани са становишта система за њихово складиштење
        - *XTML* документ – тагови одређују форматирање али не и значење
- полуструктурирани подаци (енгл. *semistructured data*)
    - имају дефинисану структуру, али постоје и одступања

        - атрибути могу да постоје у свим ентитетима или само у неким ентитетима
    - често се приказују путем графова и стабала
    - могуће је дефинисати схему, која описује могуће елементе који учествују у структури тј. могу, али не морају сви да постоје

Дакле, полуструктурирани подаци имају делимично заједничку структуру и могу садржати поља која нису позната пре тренутка пројектовања документа. Исте врсте података се могу представити на различите начине. *XML* и *JSON* су најистакнутији језици за представљање полуструктурираних података. За рад са полуструктурираним подацима релационе базе података нису најбољи избор.

Модел базе података је начин репрезентовања података, њихових својстава и односа међу подацима. Најпознатији модели база података су [6][13]:

1. Хијерархијски модел
2. Мрежни модел
3. *ER* (скраћено од енгл. *Entity Relationship*) модел
4. Релациони модел
5. Објектно-оријентисани и објектно-релациони модел
6. Документни модел
7. *EAV* (скраћено од енгл. *Entity-Attribute-Value*) модел
8. Графовски модел

Осим наведених модела базе података, постоје и други модели који се ређе користе, неки који се више не користе и они који су у фази развоја. Неки од ређе коришћених модела су: асоцијативни модел и семантички модел. Модели који се ретко или уопште не користе су: мрежни и хијерархијски модел. *ER* модел се углавном користи само као помагало при креирању релационе базе података. Најчешће коришћени модели данас су: релациони, објектно-оријентисани и документни модел база података. Употреба граф модела база података тек почиње и добија на популарности. Његова употреба се огледа у друштвеним мрежама које свакодневно привлаче пажњу великог броја људи. Релациони модел је основа великог броја система за управљање базама података [6].

Правила интегритета базе података дефинишу дозвољена стања и дозвољене прелазе система из стања у стање. **Трансакција** је логичка јединица посла и може да се састоји од једне или већег броја радњи над базом података, при чему обезбеђује услов да подаци и односи међу њима после завршетка трансакције коректно одражавају реалност која се тим подацима моделира [1].

Основне особине трансакција су (***ACID*** својства):

- **атомичност** (енгл. *atomicity*) - све или ништа
- **конзистентност** (енгл. *consistency*) - очување унутрашње конзистентности базе података
- **изолованост** (енгл. *isolation*) - трансакција ради као да је једина
- **трајност начињених измена** (енгл. *durability*) - потврђене измене опстају чак и у случају накнадног квара

Недостатак релационих база података представља релативно висока цена читања због нередундантности (непонављања истог податка у једној бази података) и строге структуре података. Нередундантност има за последицу да је ажурирање података ефикасно, јер податак постоји само на једном месту у бази и само је тај податак потребно изменити. Идеално би било да релационе базе података користимо у ситуацијама када је потребно често ажурирати податке уз ретка читања података из базе. У раду са релационим базама података отежано је дистрибуирање пре свега због конзистентности података и скупа је промена структуре због повезаности структуре са употребом и оптимизацијама.

Данашње веб апликације сусрећу се са многим проблемима. Конкурентни корисници чији је број тешко предвидети и то што се дневна производња података мери терабајтима и петабајтима само су неки од проблема. Обрада података је отежана, јер корисници у сваком тренутку врше измену података. Такође, оптерећење је неравномерно са непредвидивим порастом, непрекидно додавање нових могућности и промена постојећих компоненти додатно отежавају рад са релационим базама података.

Нема јединствене дефиниције, али се може рећи да **нерелациона база података** је база података која не почива на релационом моделу података. Нека од својстава која карактеришу ово одступање карактеристична су за поједине типове нерелационих база података - лако дистрибуирање података, хоризонтална скалабилност, одсуство стабилне статичке схеме, редукција провере интегритета. Нерелационе базе податаке не користе упитни језик *SQL*, мада данас већина *NoSQL* база података користи неки језик који је сличан *SQL*-у.

Велики број *NoSQL* база података почива на моделу каталога, који се често назива модел кључ-вредност. Основна колекција у моделу је каталог. Елемент каталога може бити проста или сложена вредност. Структура сложене вредности зависи од имплементације, обично је неки вид торке. Вредностима се приступа по кључу, док базу података чини скуп каталога.

Иако концепти могу да подсећају на релационе базе података (каталог-табела, вредност-ред,...) заправо постоје значајне разлике. Структура вредности обично није строго дефинисана, обично је нетипизирана, не инсистира се на нередундантности, практично ретко има примарних кључева и не постоји референцијални интегритет.

Неформално говорећи**, дистрибуирана база података** је база података која се не налази у целости на једној физичкој локацији (на једном рачунару), већ је раздељена на више локација које су повезане комуникационим мрежом. Свака локација, која се зове и чвор комуникационе мреже, поседује свој сопствени, аутономни систем за управљање базама података, са сопственом контролом, управљачем трансакција и опоравка од пада, и осталим значајним функцијама, а има и свој централни процесор и улазно/излазне уређаје [1].

При прављењу дистрибуираних система (не само база података) као најважнији циљеви се истичу **три важна својства**, тзв. **„*CAP*” услови**:

- **конзистентност** (енгл. *consistency*)

    - Резултат операције (читања) никада не зависи од чвора на коме се извршава.
    - Разликује се од истог термина у контексту особина трансакција
- **расположивост** (енгл. *availability*)
    - Систем је увек расположив.
- **прихватање раздвојености** (енгл. *partition tolerance*)
    - Ниједан скуп проблема, осим потпуног отказивања, не сме да произведе неисправан одзив система.

Теорема „*CAP*”: Није могуће дефинисати систем који задовољава све „*CAP*” услове (конзистентност, расположивост и толеранцију раздвојености).

При пројектовању (или конфигурисању) дистрибуираног система неопходно је направити неки компромис:

- одбацивање толеранције раздвојености
- одбацивање расположивости
- одбацивање конзистентности
- заснивање система на другачијем скупу услова
- пројектовање заобилазних путева

Појмови из „*CAP*” теореме почивају на уобичајеним концептима рада са базама података, тј. постоји зависност са особинама трансакција – *ACID*. Може се дефинисати и другачији скуп услова (*BASE*), мање оштар, у ком случају се све одговарајуће особине могу ускладити. ***BASE*** својства трансакција су:

- У суштини доступан (енгл. *Basically Available*) - не гарантује се расположивост одговора, већ само система; ако не може да се добије одговор, добиће се обавештење о томе
- Мека стања (енгл. *Soft-state*) - стање система може да се мења чак и када није у току ниједна трансакција, на пример ради постизања конзистентности
- Одложена конзистентност (енгл. *Eventually consistent*) - жртвују се гарантована стална конзистентност и изолованост трансакција зарад расположивости; систем ће у неком тренутку постати конзистентан, али ће радити и давати одговоре (потенцијално различите) и до тада.

2. *NoSQL* системи за управљање базама података

Историјски гледано, термин „*NoSQL*” први је увео 1998. године *Carlo Strozzi*. Термин *NoSQL* и значење које је *Carlo* увео значајно се разликује од данашњег. Појам је увео као назив своје апликације, која је такво име добила јер није користила стандардан *SQL* интерфејс. Подаци се чувају као стандардне *UNIX ASCII* датотеке, над којима могу да се користе стандарне *UNIX* команде (*ls, cp, cat...*), едитори (*vi, emacs*…), као и системи за контролу верзија попут *RCS* и *CVS*. Ипак и као такав, овај

систем је и даље релациони и убраја се у системе релационих база података, развијао се више од 10 година и пројекат је под *GNU* лиценцом и спада у софтвер отвореног кода.

Данашњи термин *NoSQL*, изведен од „*Not only SQL*“ (у слободном преводу „не само SQL”) и описује целу класу база података, које немају карактеристике традиционалних релационих база података. У почетку су *NoSQL* базе података „бежале” од језика *SQL*. Било је покушаја да се зову и *NonRel* базе, али је *NoSQL* преовладао, са значењу „*no SQL*” („*не SQL*”). До промене значења термина „*NoSQL*” дошло је јер данас *NoSQL* базе података користе или *SQL* или неки језик који тежи да буде што сличнији *SQL*-у. *NoSQL* базе података су све популарније у последње време највише захваљујући великим компанијама попут *Google*-а и *Amazon*-а, које их користе за чување и управљање великим количинама података које се јављају у њиховим апликацијама. Ове компаније дају пример и наводе остале на употребу истих појмова и сличних решења. Без обзира на буквално значење, *NoSQL* се користи данас као општи термин за све базе података и складишта података које не прате популарне и добро утврђене принципе релационих база података. То значи да *NoSQL* није јединствени производ, чак ни сви системи база података не примењују исте технологије. *NoSQL* представља класу производа и концепата у вези са складиштењем података и њиховим управљањем.

Као и већина нових технологија, *NoSQL* је обавијен облаком страха, неизвесности и сумње. Свет програмера се поделио у 3 категорије када је у питању NoSQL технологија:

1. Они који га обожавају – користе га, побољшавају и прате новине у свету *NoSQL*-а
2. Они који га критикују – углавном се фокусирају на његове недостатке и труде се да покажу лоше стране *NoSQL*-а
3. Они који га игноришу – могу се поделити у две групе: једни чекају да ова технологија сазри, док други верују да је ова технологија само тренутни хир који ће ускоро нестати са тржишта захваљујући некој новој технологији.

NoSQL базе података могу се дефинисати као следећа генерација база података заснована на својствима [5]:

- нерелационе (енгл. *non-relational*)
- дистрибуиране (енгл. *distributed*)
- отвореног кода (енгл. *open-source*)
- хоризонтално скалабилне (енгл. *horizontally scalable*)

NoSQL базе података створене су на основу идеје о модерним веб базама података у циљу решавања проблема које релационе базе података не могу да реше. NoSQL базе података често имају и следеће особине:

- слободне схеме (енгл. *schema-free*) - схема која може да се мења
- једноставна подршка репликацији
- једноставни API

- BASE својстава трансакција (посебно одложена конзистентност)
- огромне количине података
- ефикаснија употреба дистрибуираних индекса и *RAM*-а.

Те особине омогућавају подршку за велики број једноставних операција читања и писања у јединици времена, и задовољавају потребе данашњих апликација. Последњих година, развијено је много различитих *NoSQL* решења чији је задатак био да задовоље потребе и захтеве компанија за скалабилношћу, одржавањем и разним могућностима над подацима. *NoSQL* је као појам врло широк и сваки систем база података који није релациони се може сврстати у *NoSQL* системе. Тренутно постоји више од 225 *NoSQL* система база података, који су према моделу података класификовани у следеће категорије [5]:

- складишта кључ/вредност (*Key-Value-Store*)
- складишта докумената (*Document Store*)
- колонска складишта/колонске фамилије (*Column Store / Column Families*)
- графовске базе података (*Graph databases*)
- вишемоделне базе података (*Multimodel Databases*)
- објектне базе података (*Object Database*)
- базе података на мрежи и у облаку (*Grid & Cloud Database Solutions*)
- XML базе података (*XML Databases*)
- вишедимензионе базе података (*Multidimensional Databases*)
- вишевредносне базе података (*Multivalue Databases*)
- порекло догађаја (*Event Sourcing*)
- мрежни модел (*Network Model*)
- сродне *NoSQL* базе података (*NoSQL related databases*)

Модел података је заправо главна одлика по којој се *NoSQL* системи за управљање базама података разликују од релационих система, али се *NoSQL* базе по моделу разликују и међусобом. Тренутно три најзаступљенија модела података код *NoSQL* база података су: документни модел, графовски модел и модел кључ-вредност.

API се користи за комуницирање са неким системом. Најчешће је потребно *API*-ју проследити одређене параметре, а *API* нам враћа резултат своје обраде. Код *NoSQL* база података не постоји стандард за дизајнирање интерфејса. Сваки систем је грађен помоћу различитих технологија и *NoSQL* системи се разликују на свим нивоима (концептуалном, имплементационом, употребном,...). Различитости међу *NoSQL* системима има за последицу немогућност стандардизације *NoSQL* система, што је велика мана *NoSQL*-а. Још једна мана *NoSQL*-а је што се скоро свака технологија односи само на једну базу података (често да бисмо почели да користимо неки од *NoSQL* система потребно је да учимо нову технологију).

Приликом израде апликација, одабир система за управљање базама података је можда чак и најважнија ставка. Једном кад се апликација изгради над неким системом за управљање базама података, скупо је и прилично изазовно мигрирати податке на други систем за управљање базама података. *NoSQL* системи су релативно нови и

тренутно постоји велики број система, но највероватније ће се временом издвојити неколико највећих система.

3. Базе података докумената

Традиционални начин пословања фирми, односно управљања документима у предузећима, захтева да се сваки документ „ручно" евидентира. У тај процес је, између осталог, укључено класификовање докумената те састављање потребних ознака, које једнозначно одређују сваки од докумената. С друге стране, постоје различити степени поверљивости, односно, за различите типове докумената права приступа имају различите групе људи. У данашње време овакав приступ пословању фирме чини се прилично застарео, јер се коришћењем информационокомуникационих технологија пословање може пуно лакше обављати и то помоћу система за управљање базама података докумената.

Системи за управљање базама података докумената увелико олакшавају евиденцију и приступ свим документима који пролазе кроз организације. *NoSQL* базе података докумената су најбољи избор за такве послове, оне омогућују похрањивање докумената најчешће у *JSON* формату. Предност таквих база података јесте у томе што оне немају унапред дефинисану схему, за разлику од *SQL* база података које имају схему и подаци који се похрањују у њих морају одговарати дефинисаној схеми. *NoSQL* базе података докумената омогућавају складиштење докумената који садрже полуструктуриране податке.

Документно оријентисане базе података су базе података намењене за чување, издавање и управљање полуструктурираним подацима. Документно оријентисане базе података представљају једну од главних категорија *NoSQL* база података и њихова популарност расте заједно са употребом *NoSQL* технологије. Главни концепт документно оријентисаних база података је нотација „Документа". Свака имплементација документно оријентисане базе података се разликује у детаљима. Подаци се најчешће чувају коришћењем отворених формата: *XML, JSON, BSON , YAML*. Постоје и базе података које чувају и документа у бинарним форматима (*PDF* или *MSOffice* документи) [6].

У документном моделу објекти се чувају као документи. Документи су независни, чиме се повећавају перформансе и смањују проблеми који настају као последица конкуретног приступа подацима. Документи чувају податке базе података и мета-податке.

Познате су документне базе са кључ/вредност структуром код којих вредност представља полуструктурирани документ чија је структура разумљива бази података. Код таквих база за приступ документима у бази података најчешће се користи вредност

кључа. Кључ је најчешће стринг, али може бити *URI* или неки облик путање. База је индексирана по вредности кључа како би приступ документима био олакшан.

Осим приступа документима коришћењем механизама кључа, већина документних база података нуди *API* или упитни језик за претраживање докумената. Циљ је пронаћи документе код којих одређено поље има наведену вредност. Неке документне базе података обезбеђују *HTTP* сервере који обрађују стандардне *HTTP* захтеве за подацима упућених методама *PUT, GET, POST* и *DELETE* (примери таквих база су *OrientDB, CouchDB* и *eXist*).

Једна од основних карактеристика документно оријентисаних база података је одложена конзистентност. Измене докумената нису видљиве у истом тренутку свим клијентима који им приступају, али свима постају видљиве у неком тренутку. Као последица овога повремено се јавља неконзистентност базе. Јако битна карактеристика документно оријентисаних база података је оптимистички приступ трансакцијама што значи да се синхронизација одлаже до тренутка потврђивања трансакције. Оптимистички приступ трансакцијама је погодан код конверзационих трансакција, које се користе код вишеслојних апликација или веб апликација. Овај модел података је погодно користити [9]:

1. када је потребно складиштити динамичке податке, у системима као што су *Content Management System* (скраћено *CMS*) и *Customer Relationship Management* (скраћено *CMR*) ентитети, које крајњи корисник може да мења по потреби.
2. код веб података (*Web Related Data*), као што су корисничке сесије, логови и други подаци који се одржавају за потребе веб апликација. Складишта докумената омогућавају да се подацима приступа као целини једним захтевом ка удаљеном серверу.
3. код динамичких ентитета, као што су кориснички прилагодљиви ентитети, ентитети са великим бројем изборних области, итд.
4. код постојаног модела погледа (енгл. *Persistent View Models*) – уместо поновног отварања и приказа модела од нуле на сваки захтев, може се приказ сачувати у свом коначном облику документа података. Ово доводи до смањења рачунања, удаљених позива и побољшања укупних перформанси.
5. код складиштења великих количина података.

Примери документних база података:

1. *MongoDB* [7] је оријентисана ка колекцијама. Подаци су груписани у скупове који се називају „колекцијама”. Свака колекција има јединствено име у бази и може да садржи неограничен број докумената. Колекције су аналогон табелама код РСУБП, једино немају дефинисану схему. Чува податке у *BSON* формату, који је структурирана колекција парова кључ/вредност, где је кључ стринг, док вредност може бити било који од многобројних типова података.
2. *RavenDB* [7] је документно оријентисана база података која као замену за упитни језик користи *.NET Linq*. *RavenDB* велику пажњу придаје обезбеђивању што бољих перформанси. *RavenDB* је флексибилна и

скалабилна *NoSQL* база података за рад са *.NET* и *Windows* платформама. Сладишти документе у JSON формату. Индекси се могу дефинисати помоћу кода написаног *Linq* синтаксом у оквиру *C#* кода.

3. *Terrastore* [7] је модерно складиште за документе. Омогућава унапређену скалабилност и еластичност без жртвовања конзистентности. То омогућава брза технологија кластеровања коју примењује компанија *Terracotta*. (*Terracotta* је компанија која израђује софтвер специјализован за повећање (побољшање) скалабилности, расположивости и перформанси за *real-time Big Data* апликације.)
4. *OrientDB* [7] је *NoSQL* систем отвореног кода за управљање базама података написан у Јави. Иако је база података оријентисана ка документима, односима се управља као у графовским базама података са директним везама међу слоговима. *OrientDB* подржава начине рада: без схеме (*schema-less*), комплетна схема (*schema-full*) и мешовита схема (*schema-mixed*). Овај систем је врло безбедан, а то дугује упитном језику *SQL*, који користи.
5. *ThruDB* [7] је скуп једноставних сервиса изграђених помоћу *Apache Thrift* оквира, што омогућава индексирање и сервисе складиштења докумената за израду и скалирање веб сајтова. Сврха овог система је да обезбеди веб програмерима флексибилне и брзе сервисе који се лако користе.
6. *Jackrabbit* [7]: *The Apache Jackrabbit* је складиште података чија је имплементација у складу са *Content Repository for Java Technology API* (*JCR*). То је хијерархијско складиште за структуриране и неструктуриране податке, над којима се могу вршити текстуална претраживања, трансакције, посматрања и тд.

Проблеми могу настати када се захтева трансакциона обрада (нема подршке за *ACID* трансакције) и када је неопходно користити велики број спајања.

Кључ-вредност	**CouchDB**	**MongoDB**	**RavenDB**
Написана у	Erlang	C++	C#
Платформе	Linux	Linux, Mac, Windows	Windows
Протокол	HTTP/REST	сопствени TCP/IP	HTTP/REST
Начин чувања података	диск	диск	меморија и диск
Репликација	peer-based	master-slave	plug-in
Партиционисање	ДА	ДА	ДА
Индекси	ДА	ДА	ДА
MapReduce	ДА	ДА	ДА

ACID трансакције	НЕ	НЕ	ДА

Табела 1: Поређење водећих документних база података

4. *Apache CouchDB* систем за управљање базама података

Apache CouchDB је систем за управљање базама података отвореног кода написан у програмском језику *Erlang*, користи *JavaScript* за *Map/Reduce* индексе, користи *RESTful HTTP API* за читање и ажурирање докумената базе података, омогућава мастер-мастер подешавање, што подржава аутоматско откривање колизија, док за контролу конкурентности користи *MVCC* (*Multi-Version Concurrency Control*). *CouchDB* је пројекат који води корпорација *Apache* [7]. *Damien Katz* је у априлу 2005. године започео рад на овом систему. *Katz* је 2012. године напустио пројекат, који је наставио да се развија [6]. Израз „*Couch*" у називу система је акроним од „*cluster of unreliable commodity hardware*".

Ако би требало једном речју описати *CouchDB*, рекли бисмо „**опуштање**" (енгл. *relax*). При покретању *CouchDB*-а, види се порука „*Apache CouchDB је покренут. Време је за опуштање.*" („*Apache CouchDB has started. Time to relax.*") [6]. Учење и разумевање *CouchDB*-а би требало да буде лако и природно сваком ко је упознат са Вебом. Идеја твораца овог система је била да искористе постојеће технологије како би корисницима што више олакшали рад и да би коришћење овог система постало задовољство (ужитак) за корисника. Цела архитектура CouchDB-а је толерантна на грешке, грешке се дешавају у контролисаном окружењу и отклањају се „грациозно". Корисници не морају да страхују од случајног понашања и грешака којима је тешко наћи узрок при коришћењу *CouchDB*-а у својим апликацијама. Ако нешто не ради како би требало, лако је уочити шта је проблем, иако су такве ситуације ретке. *CouchDB* добро подноси нагли скок оптерећења. Када се оптерећење сајта нагло повећа *CouchDB* ће опслужити много конкурентних захтева, а да при том неће доћи до пада система. То може захтевати мало више времена за обраду сваког захтева, али ће систем успети да одговори на све захтеве. Када „пик" оптерећења прође, *CouchDB* ће поново радити уобичајеном брзином.

***CouchDB* документ** је *JSON* објекат састављен од именованих поља. Вредности именованох поља могу бити бројеви, стрингови, датуми, чак уређене листе и асоцијативне мапе. Базу чини скуп докумената са јединственим *URI*-јима. Број именованих поља у документу није ограничен. Сваки документ има обавезна поља _id и _rev која означавају редом јединствени идентификатор документа и ознаку верзије

документа. Вредност поља _id може да генерише систем аутоматски (вредност типа *UUID*) или да корисник унесе назив (стринг), који ће јединствено одређивати документ. Идентификатор документа мора бити јединствен у бази у којој је документ смештен.

Премер документа, објава у оквиру блога: (поље *Tags* садржи листу)

```
{
  "Subject": "I like Plankton",
  "Author": "Rusty",
  "PostedDate": "5/23/2006",
  "Tags": ["plankton", "baseball", "decisions"],
  "Body": "I decided today that I don't like baseball. I like plankton."
}
```

Осим именованих поља, документ може да садржи мета-податке и додатке (енгл. *attachments*). Сви додаци се смештају у поље под именом _attachments. Додатке најчешће чине фајлови. Број додатака смештених у један документ није ограничен.

Документ може саджати путању ка неком другом документу (*URI* документа), али се не проверава исправност такве путање (верује се кориснику). Једно од ограничења *CouchDB*-а је то што један документ не може у себи садржати други документ („угњежденост докумената" не постоји).

Документи су индексирани B-стаблима на основу својих имена, DocID-ја, и секвенцијалног ID-ја. Свако ажурирање инстанце базе података проузрокује прављење новог секвенцијалног ID-ја. Секвенцијални ID-јеви се касније користе за инкременталну претрагу промена у бази. Индекси у B-стаблима се ажурирају симултано са ажурирањем докумената. Када је ажурирање завршено на диск се смештају подаци и индекси, а база је у конзистентном стању након ажурирања. Након ажурирања документи су смештени на диск, поља документа и мета-подаци су у баферу[2], документи су поређани један за другим редом, што утиче на ефикасност прављења погледа. Бафер се користи само приликом извршавања трансакција за тренутно смештање података.

Приступање подацима који се налазе у *CouchDB* бази података омогућавају **погледи**. Поглед је *JavaScript* функција која документе, који се налазе у бази, добија као аргументе. Погледи су дефинисани унутар **дизајн докумената**. Дизајн документ је специјални документ у којом су дефинисане појединости у вези са базом и апликацијом, ако постоји апликација која обрађује податке из базе. Погледи су динамички изграђени и не утичу на документ над којим су изграђени, тако да може постојати онолико различитих приказивања истих података колико је кориснику потребно. Погледи су динамичка репрезентација тренутног садржаја базе података, али у базама које имају хиљаде или милионе докумената прављење погледа при сваком приступу подацима одузима много времена и захтева много ресурса. Ради бржег приступа подацима CouchDB индексира погледе. Када корисник користи поглед,

[2] реч је о TCP баферу [16] , опширније о TCP баферу можете прочитати на следећој адреси: http://docs.citrix.com/content/dam/en-us/netscaler/9-3/downloads/en.netscaler.ns-tcpb-gen-wrapper-93-con.pdf

аутоматски се сви његови индекси ажурирају. Репликација података не утиче на погледе нити погледи представљају сметњу при репликацији. Погледи се при репликацији понашају као и било који други документ.

При ажурирању података *CouchDB* не закључава податке и има оптимистички приступ трансакцијама. Кад клијентска апликација жели да ажурира базу, најпре систем потражи документе базе над којима ће се вршити ажурирање, примени се жељено ажурирање и на крају се тако ажурирани документи сачувају у бази. Ако је неки клијент сачувао измене истог документа пре другог, други добија обавештење да је дошло до грешке. Овај проблем се може решити тако што се поновно отвари документ и још једном се покуша извршити жељено ажурирање. Дакле, ажурирање се врши по принципу „све или ништа", или је потпуно успела операција или је потпуно неуспешна.

Извршавање трансакција је серијско (једна за другом) осим у случају података типа бинарни blob када је конкурентно. Клијенти који читају не „закључавају" податке које читају и неограничен број клијената може истовремено читати исти документ. Клијенти који читају не морају да чекају да се неко писање заврши, коришћењем *MVCC* метода за операције читања сви клијенти виде конзистентан снимак стања. Корисници који читају виде исти снимак стања све време док читају из базе, без обзира на то да ли су током њиховог читања ти подаци мењани. *MVCC* метода обезбеђује да *CouchDB* може без проблема да ради конкурентно са великим бројем корисника који пишу и корисника који читају.

CouchDB је „база података за веб" јер складишти податке у облаку као *JSON* документа [7]. Ова база података је веома добро повезана с модерним интернет апликацијама и апликацијама за мобилне уређаје. *CouchDB* приступа документима и испитује индексе путем интернет претраживача, а поставља индексе, комбинује и трансформише документе уз помоћ погледа написаних у *JavaScript*-у. ***Futon*** је графовским кориснички интерфејс који омогућава креирање, ажурирање, брисање и прегледање докумената и погледа, омогућава приступ параметрима који дефинишу и описују базу, омогућава интерфејс за иницијализацију репликације. *Futon* је врло једноставан за коришћење и није потребно никакво предзнање да би се започело са његовим коришћењем.

Дистрибуирано скалирање је једна од могућности система база података која је јако пожељна при раду са дистрибуираним системима, *CouchDB* подржава и ту могућност и истичу је као јако значајну. *CouchDB* база података је толерантна на раздвојеност (партициоисање). *CouchDB* може да дистрибуира податке или апликације, користећи остале могућности ове базе података. Осим ових функција, кориснику су на располагању и остале могућности попут транспоновања докумената и извештаја о стварним, реалним променама у систему. *CouchDB* омогућава праћење измена у моменту догађања, а интерфејс је лак за употребу, гледано са аспекта администратора [7]. *CouchDB* је дистрибуирани систем за управљање базама података са непосредно повезаним чворовима (*P2PDDB*). Тачније, модел дистрибуирања *CouchDB*-а је федеративни *P2P* модел. Операције над подацима на локалним чворовима су потпуно

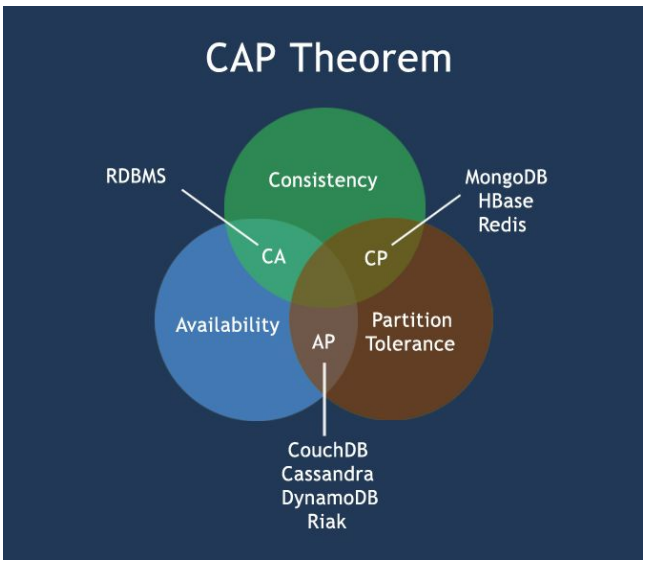
транспарентне, то значи да корисници могу да врше операције над базом као да је у питању централизована база података не водећи рачуна о томе где су смештени фрагменти и чворови. Али то не значи да можемо вршити операције над свим глобалним подацима са само једног чвора. За операције над подацима других чворова потребна је ауторизација.

Код *CouchDB*-а је схема базе слободна тј. не постоје унапред дефинисане структуре података као што су табеле. За разлику од *SQL* база података, које су дизајниране да складиште и извештавају о структурираним, међусобно повезаним подацима, *CouchDB* је дизајниран за складиштење и извештавање о великом броју полуструктурираних података. *CouchDB* у великој мери поједностављује развој документно оријентисаних апликација. Код *SQL* базе података, схема и складиштење постојећих података морају бити ажурирани у складу са развојем потреба. Са *CouchDB*, строга схема није потребна, тако да нове врсте документа са новом стуктуром могу безбедно да се додају уз старе. *CouchDB* је дизајниран тако да се лако рукује новим врстама докумената. Структура документа или података може се динамички променити ради прилагођавања порасту потреба, дакле, систем је скалабилан. Код *CouchDB* система база података лако и ефикасно се приступа подацима. *CouchDB* подржава партиционисање и одложену конзистентност. Доста пажње је посвећено скалабилности и бризи о подацима. *CouchDB* систем за управљање базама података је толерантан на грешке (енгл. *fault-tolerant*) и безбедност корисничких података ставља на прво место.

Решење *CouchDB* користи репликацију да пропагира промене апликација преко чворова који у њој учествују и тиме ставља нагласак на расположивост и прихватање раздвојености као начин за задовољавање ограничења која су установљена *CAP* теоремом (Слика 2). Када дође до раздвојености могуће је ажурирање података на сваком од чворова, али није могућа синхронизација података, тиме се не гарантује конзистентност базе. Најчешћи разлог настанка раздвојености је прекид у комуникацији (проблем са мрежом), тада сваки појединачни чвор ради (активан је), али чворови не могу да комуницирају међусобно. Понекад се мрежа чворова прекидом комуникације подели на „подмреже" унутар којих чворови могу неометано да комуницирају.

Откривање и решавање конфликтима, који настају при писању, је кључно питање за сваки дистрибуирани систем. *CouchDB* конфликте при писању третира као уобичајено стање, а не као изузетак. *CouchDB* дозвољава у оквиру једне базе података истовремено постојање докумената чије су верзије конфликтне. Када се чита документ за који постоје конфликтне верзије детерминистичким алгоритном чвор (са ког се чита) одређује која је верзија „победник" и чита податке те верзије документа. То што је алгоритам за одређивање „победника" детерминистички гарантује да ће на сваком чвору за „победника" бити изабрана иста верзија. Пример ситуације у којој долази до стварања конфликтних верзија документа: Алиса жели да податке из Бобове визит карте има на свом десктоп рачунару и на свом лаптоп рачунару. (Иницијално исти документ чува на обема локацијама, копирала је документ са десктоп рачинара на

лаптоп.) Алиса је затим на десктоп рачунару изменила адресу Бобове електронске поште, а на лаптоп рачунару број Бобовог мобилног телефона. При синхронизацији Алиса би очекивала да ће имати „најсвежије" податке о броју мобилног телефона и о адреси електронске поште на оба рачунара у оквиру једног документа. Али то се неће десити, већ ће на оба рачунара постојати обе верзије, а при читању једна ће бити „победник". Да би у оквиру једног документа постојали „најсвежији" подаци о броју мобилног телефона и о адреси електронске поште (на оба рачунара) у оквиру једног документа, потребно је направити нови документ у који се „учешљавају" подаци. Погледи када наиђу на документ који има конфликтне верзије, такође, користе поменути алгоритам за избор „победника".



Слика 2: CouchDB и CAP теорема

CouchDB је *NoSQL* база која има доста корисника који раде са великим количинама података. Ниједна база података не може бити замена за све остале, тако ни *CouchDB* није замена за све остале, али постоје ситуације у којима представља добар избор. Неки од корисника овог система су *Facebook*, који га користи за део својих апликација, *Credit Suisse Group AG* (швајцарска групација која се бави финасијама, улагањима; послују у 57 држава) и многи други.

5. Инсталација и почетак коришћења система *CouchDB* под оперативним системом *Ubuntu Linux*

За само неколико минута *CouchDB* се инсталира коришћењем команди командне линије. Прва команда је:

```
sudo apt-get update
```

Затим

```
sudo apt-get install couchdb
```

За следећу команду користи се *curl*, алат командне линије за примање и слање фајлова коришћењем синтаксе *URL*-а. Овај алат се зове исто као и један објектно-оријентисан програмски језик. Ако није инсталиран, неопходно је инсталирати га следећом командом:

```
sudo apt-get install curl
```

Следећи корак је:

```
curl localhost:5984
```

Требало би да се појави следећа порука:

```
{
    "couchdb": "Welcome",
    "uuid": "85fb71bf700c17267fef77535820e371",
    "version": "1.4.0",
    "vendor": {
        "version": "1.4.0",
        "name": "The Apache Software Foundation"
        }
}
```

Овим је завршено инсталирање.

Прављење нове базе података, која ће се звати „new_database”, могуће је помоћу методе *PUT*:

```
curl -X PUT localhost:5984/new_database
```

Требало би да се појави следећа порука:

```
{"ok":true}
```

Да бисмо из командне линије видели које базе постоје у систему, позивамо команду:

```
curl -X GET http://localhost:5984/_all_dbs
```

Као резултат претходног упита добићемо листу са именима база које постоје у систему. Ако желимо уз ову информацију да видимо како комуницирају клијент и сервер (локални, у овом случају), можемо позвати исту команду уз малу измену, користећи опцију -vX уместо -X:

```
curl -vX GET http://localhost:5984/_all_dbs
```

У конзоли након претходне команде види поруке које размењују клијент и сервер путем *HTTP* протокола и резултат. Порука је захтев који клијент шаље серверу или одговор који сервер шаље клијенту. На овај начин можемо да видимо шта се дешава „испод хаубе”. „Испод хаубе” све се извршава помоћу метода *HTTP* протокола *GET, PUT, POST* и *DELETE*.

Брисање базе „new_database” из система се може урадити помоћу команде:

```
curl -X DELETE http://localhost:5984/new_database
```

Ово је само један од многобројних начина на који се може комуницирати са базом података.

За визуелну репрезентацију базе података *CouchDB* нуди алат *Futon*. Помоћу *Futon*-а лако је направити или уклонити базу података, прегледати и мењати документе, састављати и покретати *MapReduce* погледе, вршити репликацију. Да бисмо покренули *Futon* довољно је у прегледачу у поље за унос *URI*-ја уписати:

http://127.0.0.1:5984/_utils/

и све је спремно за прављење нове базе, као и докумената у њој.

CouchDB се може инсталирати и на следећим оперативним системима: *Windows*, *Mac OS X* и *FreeBSD*.

6. Методологија

6.1. Програмски језик *Erlang*

Апликација *CouchDB* развијена је на програмском језику који се зове *Erlang*. То је конкурентни програмски језик развијен од стране *Ericsson*-а првобитно за њихове апликације у реалном времену. *Erlang* је име добио по математичару и инжењеру који се звао *Agner Krarup Erlang* (1878–1929). Програмски језик *Erlang* делује као најприроднији одговор на све конкурентне (и дистрибуиране) проблеме на које програмери наилазе, одличан је за писање апликације од којих се очекује висок степен расположивости. Програми писани у *Erlang*-у имају добре перформансе и могу се покретати на различитим платформама. *Erlang* се добро носи са грешкама које могу настати при извршавању програма. За *Erlang* „све је процес". Процеси су строго изоловани (не деле ресурсе; сваки процес има своју „приватну" меморију) и једино комуцирају разменом порука. Процеси се врло ефикасно стварају и бришу. Апстракција извођења процесима представља апстрактну границу која не дозвољава пропагацију грешака.

Још неке од занимљивих апликација развијених на овом језику су *Yaws*, веб сервер висиких перформанси, *Facebook Chat*, итд. Цена коју плаћају они који тек почињу да користе *Erlang* је како *Damien Katz* (творац *CouchDB*-а и *Erlang* ентузијаста) истиче:

- Незгодна синтакса, инспирисана *Prolog*-ом
- Необични условни изрази
- Тешке операције са стринговима
- Нема класа или простора имена (енгл. *namespace*)

6.2. Метод контроле конкурентности *MVCC*: контрола конкурентности са вишеструким верзијама података

При конкурентном раду може доћи до више различитих проблема, јер конкурентност подразумева дељење ресурса и често има за последицу истовремено

или блиско наизменично приступање. Проблеми који настају при конкурентном раду могу се поделити у четири групе: изгубљене промене, непотврђене измене, непоновљиво читање и фантомски редови.

Контрола конкурентности вишеструке верзије података (*Multi-Version Concurrency Control*, краће *MVCC*) је метод контроле конкурентности. Обично га користе системи за управљање базама података да обезбеде истовремени (конкурентни) приступ бази података и неки програмски језици. Ако неко чита из базе података у исто време кад неко други уписује нешто у базу, могуће је да ће читалац видети неконзистентне податке. Постоји неколико начина решавања овог проблема, који се решава контролом конкурентности. Најједноставнији начин је да се сачека да сви корисници заврше писање, а онда се омогући читање свим корисницима базе. Овај начин познатији је под називом „закључавање катанаца". Ово може бити веома споро, па *MVCC* има другачији приступ: сваки корисник повезан са базом података види стање базе података у одређеном тренутку у времену. Све промене које је извршио онај ко пише други корисници базе неће видети док промене не буду завршене (или, у смислу релационе базе: док трансакција не буде потврђена (*committed*)). Када се користи *MVCC* метода, ажурирањем се неће стари подаци заменити новим подацима, већ ће се означити стари подаци као застарели и додаће се новија верзија тих података на другом месту. Тако постоји сачувано више верзија базе, али се најновија користи. Свака верзија се означава идентификатором трансакције која је мењала податак. Ово омогућава онима који читају, читаоцима, да приступе подацима који су били тамо када су започели читање, па макар то било измењено или избрисано у међувремену. Такође омогућава бази података да избегне трошкове попуњавања рупа у меморији, али захтева (генерално) од система повремено кретање кроз верзије и брисање старих, застарелих података. *MVCC* открива мртве петље и решава проблеме приступа подацима.

MVCC је у последње време јако популарна метода контроле конкурентности, користе је *SQL* базе података, *Oracle* базе података, *MSSQL Server*, *MySQL, PostgresSQL* и поједине *NoSQL* базе података.

6.3. Модел програмирања *Map/Reduce*

Map/Reduce модел програмирања патентиран од стране *Google*-а представља једну од најпопуларнијих метода за процесирање података. То је модел паралелног програмирања који омогућава дистрибуирано обрађивање великих количина података смештених на кластерима рачунара. Ова идеја потиче из функционалног програмирања. У функционалном програмирању ако имамо листу (нпр. [1, 2, 3, 4]) и на њу применимо неку функцију креира се нова листа (нпр. [2, 4, 6, 8]), претходна листа остаје неизмењена. Постоји и концепт редукујуће функције, на пример, применимо

функцију збира на листу [2, 4, 6, 8] добијамо као резултат збир свих чланова листе, за дати пример резултат је 20. Ова идеја се користи и при обради великих количина података, с тим што је измењена да ради са колекцијама торки или паровима кључ/вредност. Функција се примењује на сваки пар кључ/вредност у колекцији и генерише нову колекцију, на коју се касније може применити редукујућа функција. *Map/Reduce* модел програмирања је инспирисан функцијама Map() и Reduce() функционалног програмирања. Функција Map() филтрира и сортира податке из базе, док функција Reduce() обавља збирну операцију. Основне предности *Map/Reduce* модела су робусност и скалабилност. Све предности овог модела могу се уочити само са вишенитном имплементацијом, јер једнонитна имлементација не може бити бржа од традиционалног модела програмирања.

6.3.1. Начин рада *Map/Reduce* модела

Начин рада *MapReduce* модела је такав да он обично раздвоји улазне информације у независне групе које се затим путем пресликавања обрађују на паралелан начин. Програмски оквир сортира информације добијене путем пресликавања које се затим претварају у улазне податке другог дела процеса, редуковања. Кроз програмски оквир се извршавају задаци распоређивања, подаци се прате, а неиспуњени задаци се поново извршавају. *Map/Reduce* програмски оквир обрађује искључиво уређене парове кључ/вредност, то јест, овај оквир посматра добијене информације као скуп парова кључ/вредност и производи други скуп парова кључ/вредност као добијене информације другачијег типа.

Пример *Map/Reduce* модела програмирања: „*WordCount*“ је једноставна апликација која пребројава колико се пута свака реч појавила у одређеном, датом скупу информација [8].

```
1       package org.myorg;
2
3       import java.io.IOException;
4       import java.util.*;
5       import org.apache.hadoop.fs.Path;
6       import org.apache.hadoop.conf.*;
7       import org.apache.hadoop.io.*;
8       import org.apache.hadoop.mapred.*;
9       import org.apache.hadoop.util.*;

10      public class WordCount {

        // имплементација интерфејса  Mapper
```

```
11        public static class Map extends MapReduceBase implements
 Mapper<LongWritable, Text, Text,IntWritable> {

12                private final static IntWritable one = new IntWritable(1);
13                private Text word = new Text();
14                public void map(LongWritable key, Text value,
                        OutputCollector<Text, IntWritable> output, Reporter reporter) throws
IOException {      //„хватање” изузетака
15                  String line = value.toString();
16                  StringTokenizer tokenizer = new StringTokenizer(line);
17                  while (tokenizer.hasMoreTokens()){
18                  word.set(tokenizer.nextToken());
19                  output.collect(word, one);
20                              }
21                  }
22            }

          // имплементација интерфејса Reducer
23        public static class Reduce extends MapReduceBase implements
              Reducer<Text, IntWritable, Text,IntWritable> {
24              public void reduce(Text key,Iterator<IntWritable> values,
              OutputCollector<Text, IntWritable>  output, Reporter reporter) throws
IOException {               // „хватање” изузетака
25                    int sum = 0;
26                    while (values.hasNext()) {
27                    sum += values.next().get();
28                          }
29                    output.collect(key, new IntWritable(sum));
30                      }
31        }

           // почетак програма
32        public static void main(String[] args) throws Exception {
33                JobConf conf = new JobConf(WordCount.class);
34                conf.setJobName("wordcount");
35                conf.setOutputKeyClass(Text.class);
36                conf.setOutputValueClass(IntWritable.class);
37                conf.setMapperClass(Map.class);
38                conf.setCombinerClass(Reduce.class);
```

```
39              conf.setReducerClass(Reduce.class);
40              conf.setInputFormat(TextInputFormat.class);
41              conf.setOutputFormat(TextOutputFormat.class);
42              FileInputFormat.setInputPaths(conf, new Path(args[0]));
43              FileOutputFormat.setOutputPath(conf, new Path(args[1]));
44              JobClient.runJob(conf);
45              }
46      }
```

Имплементација интерфејса „*Mapper*“ (редови 11-22), преко метода мапирања (редови 14-21), обрађује ред по ред, што је предвиђено наглашеним форматом „*TextInputFormat*“ (ред 40). Он затим раздваја линију у токене, додатно раздвојене бланко простором путем „*StringTokenizer*“ опције и емитује кључ/вредност пар < <word>, 1>.

Примери улазних података су смештени у два фајла. У првом фајлу је садржај: *Hello World! Bye World!* У другом фајлу је садржај: *Hello Hadoop! Goodbye Hadoop!*
За дати пример информација које емитује прво пресликавање:
<Hello, 1>
<World, 1>
<Bye, 1>
<World, 1>
Друго пресликавање емитује:
<Hello, 1>
<Hadoop, 1>
<Goodbye, 1>
<Hadoop, 1>

Апликација „*WordCount*“ такође предвиђа и наглашава опцију „*combiner*“ (ред 38). Стога, добијене информације сваког пресликавања пролазе кроз локалну верзију опције (што је исто као и опција умањења за саму конфигурацију задатка) за локализовање и скупљање након што се информације прво сортирају.
Добијене информације првог пресликавања:
<Bye, 1>
<Hello, 1>
<World, 2>
Добијене информације другог пресликавања:
<Goodbye, 1>
<Hadoop, 2>
<Hello, 1>

Коришћење опције „*Reducer*“ (редови 23-31), путем метода редуковања (редови 24-30) само сабира дате вредности које представљају број понављања у сваком реду (у овом примеру - речи).

Стога су добијене информације посла:
<Bye, 1>
<Goodbye, 1>
<Hadoop, 2>
<Hello, 2>
<World, 2>

Овакав метод наглашава посебне елементе посла, као што су путање убачених и добијених информација (пребачене путем линије команди), типова „key/value“, формата „input/output“ итд., што је све присутно у почетној конфигурацији. Онда се позива опција „*JobClient.runJob*“ (ред 44) да би се све то предало и да би се пратио прогрес рада.

Интерфејси „*Mapper*“ а затим и „*Reducer*“ су укључени у апликацију како би пружили методе пресликавања и редуковања.

6.3.2. *Hadoop Map/Reduce*

Програмски оквир *Hadoop* развијен је за паралелну обраду неструктурираних података. Користи програмски модел *Map/Reduce*, а за складиштење података користи *Hadoop Distributed File System* (скраћено *HDFS*). Програмски оквир *Hadoop* написан је на програмском језику Јава.

„*Hadoop Map/Reduce*“ је модел програмирања за олакшано описивање апликација које обрађују велике количине података (садржаје од неколико терабајта) на сигуран и поуздан начин, уз извесну толеранцију на грешке. Сервери за рачунање и складиштење података су исти. И „*Map Reduce*“ модел програмирања и систем за дистрибирање података „*Hadoop*“ раде на истом систему сервера.

Оваква конфигурација дозвољава програмском оквиру да ефикасно распоређује задатке на сервере где су подаци већ смештени, што резултује појачаном могућношћу примања и слања података. Модел програмирања се састоји из два налога, где главни налог распоређује задатке везане за компоненте самог посла на налоге који то извршавају. Ти налози се затим прате, а задаци који се не изврше се враћају на поновну обраду. Кроз апликације су одређене локације унетих и добијених информација путем имплементације пригодног интерфејса и/или апстрактних класа. Ти налози се функционално надгледају, задаци се распоређују, а статус израде и информације су доступне серверима задуженим за пружање резултата. Иако је програмски оквир „*Hadoop*“ имплементиран у програмском језику Јава, модел програмирања „*Map/Reduce*“ не мора да буде написан тим програмским језиком.

6.4. *JSON* формат

JSON (*JavaScript Object Notation*) је формат за размену података. Предност му је што је читљив људима, а такође је једноставан рачунарима за парсирање и генерисање.

JSON је заправо текстуални формат који не зависи ни од једног програмског језика, већ користи конвенције програмских језика сличних C-у.

JSON формат је добио већу примену већом употребом *AJAX* (*Asynchronous JavaScript and XML*) технологије код израде веб апликација. У тим апликацијама је потребно врло брзо и ефикасно приступити подацима асинхроно.

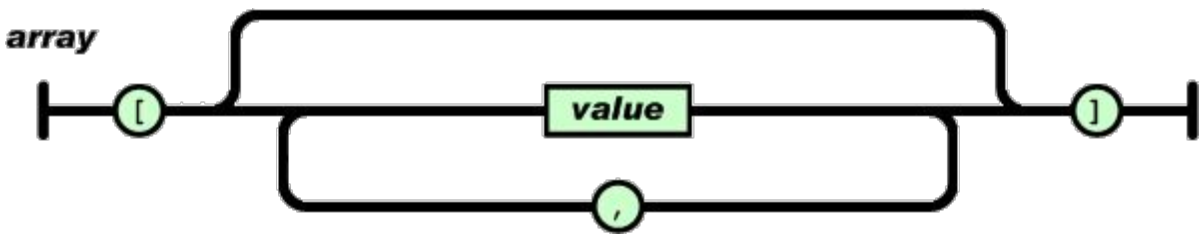
Повећањем популарности друштвених мрежа, много се веб страница ослања на садржај који је објављен на разним друштвеним мрежама, стога се на врло лаган начин кориштењем *jQuery* библиотеке за *JavaScript* програмски језик могу дохватати различити подаци с различитих доступних сервиса. Ти сервиси најчешће податке враћају управо у *JSON* формату. *JSON* јесте текстуални стандард намењен размени података у форми читљивој и разумљивој и за човека и за рачунар. *JSON* формат је изведен из *JavaScript* језика. Званична врста интернет медија за *JSON* је „application/json" (*MIME* тип/*MIME* подтип), а екстензија *JSON* фајлова је .json. *JSON* се често користи за серијализацију и пренос структурираних података путем мреже. Пре свега се користи за размену података између сервера и веб апликације, као замена за *XML*.

На пример, *MongoDB* документе складишти у формату који се назива *BSON*, и користи га за размену података са апликацијама на мрежи. *BSON* формат се заснива на *JSON*-у и настао као скраћеница од „*Binary JSON*".

6.4.1. Типови података у *JSON* формату

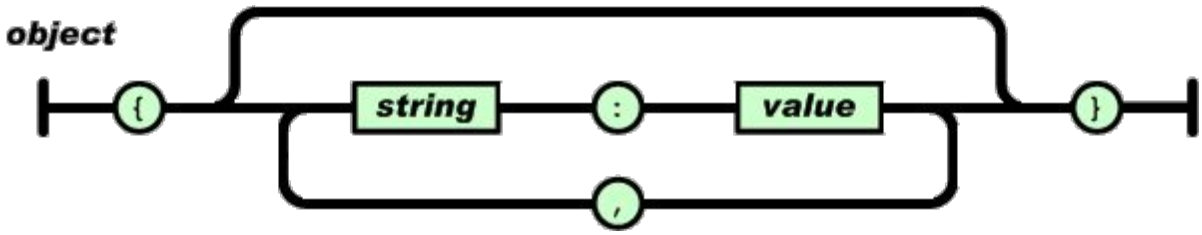
У *JSON* формату су допуштени следећи типови података:

- Број - *JSON* подржава и целе и децималне бројеве, с тиме да децимални морају бити одвојени тачком
- *String* - низ знакова
- *Boolean* - true или false вредности
- *null* - непостојећа вредност; кључна реч null
- Низ - сортирани низ чији елементи могу бити било који подржани тип података у *JSON* формату



Слика 3: Графички приказ низа у JSON формату

- Објекат - несортирани низ облика име/вредност парова. Витичастим заградама је оивичен објекат, а двотачка дели назив од вредности својства објекта



Слика 4: Графички приказ објекта у JSON формату

6.4.2. Пример података записаних у *JSON* формату

Пример *JSON* објекта

```
{
   "glossary": {
      "title": "example glossary",
       "GlossDiv": {
                  "title": "S",
                  "GlossList": {
                              "GlossEntry": {
                                          "ID": "SGML",
                                          "SortAs": "SGML",
                                          "GlossTerm": "Standard Generalized Markup Language",
                                          "Acronym": "SGML",
                                          "Abbrev": "ISO 8879:1986",
                                          "GlossDef": {
   "para": "A meta-markup language, used to create markup languages such as DocBook.",
    "GlossSeeAlso": ["GML", "XML"]
                                                      },
                                          "GlossSee": "markup"
                                                }
                                    }
                        }
                  }
}
```

Пример JSON објекта

У примеру је описан објекат glossary који се састоји од својстава title, који је типа стринг, и *GlossDiv*, који је објекат. *GlossDiv* чине својства описана помоћу још једног објекта GlossDef и пет својстава чије су вредности стринг типа.

7. *CouchDB* апликација за издавање возачких дозвола

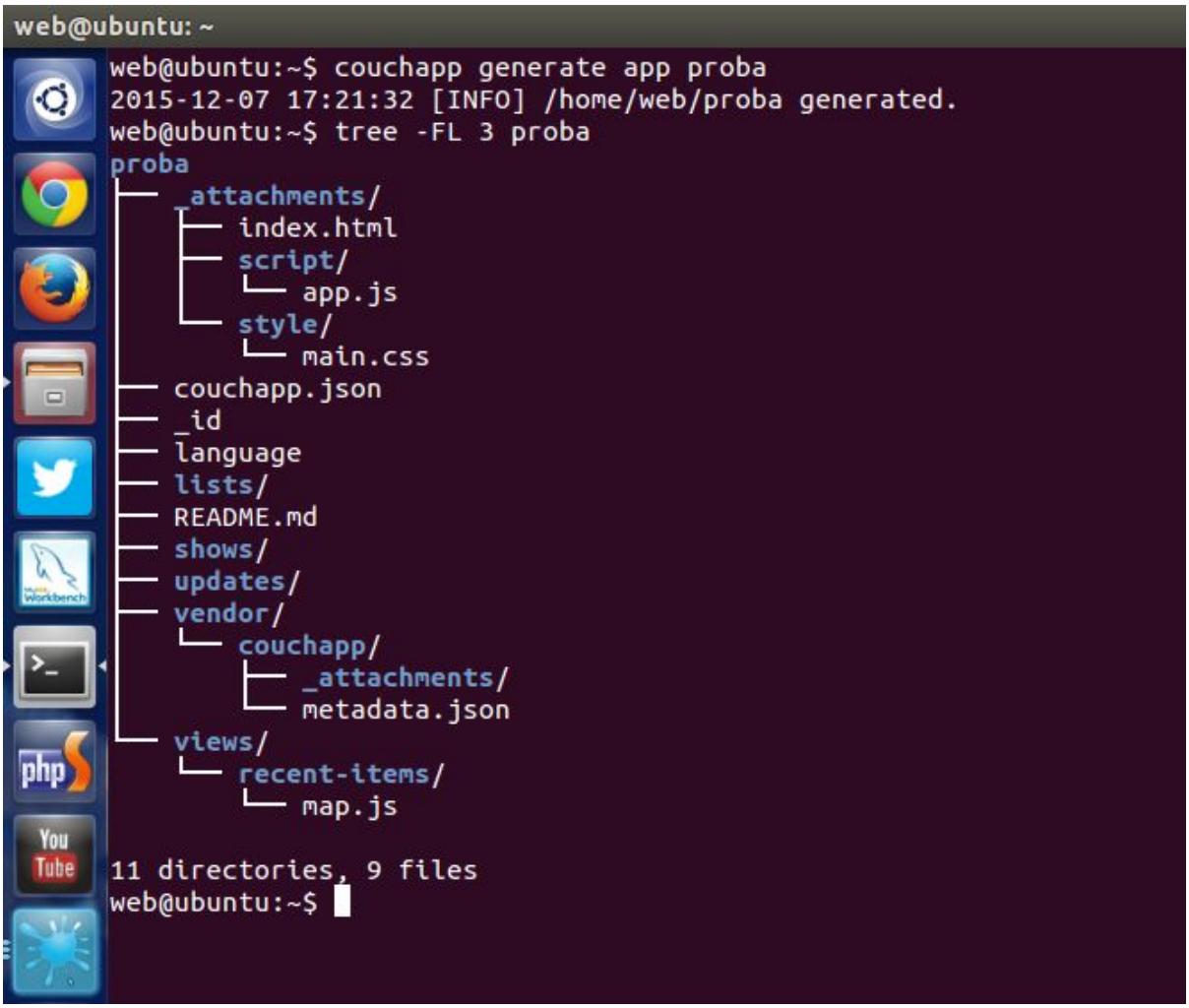
CouchApp је веб апликација написана помоћу *JavaScript*-а и *HTML*-а за рад са *CouchDB* базом података. Оваквој апликацији није потребан ни један други „алат" да би радила. Постоји истоимени алат за прављење *CouchApp* апликација и постављање апликације на сервер. Тако постављена апликација је повезна са *CouchDB* базом, која се креира на основу подешавања описаних унутар апликације. Помоћу couchapp алата једном командом се креира фолдер са подфордерима и фајловима који су обавезни за сваку *CouchApp* апликацију. Креирајмо апликацију која се зове „proba" командом:

```
couchapp generate app proba
```

Командом командне линије:

```
tree -f 3 proba
```

можемо видети све фолдере, подфолдере и фајлове креиране претходном командом. (Слика 5)



Слика 5: Садржај креираног фолдера

Најважнији елементи тако креиране апликације су фолдери _attachments, views, lists и shows. Фолдер views садржи погледе над базом података тј. *JavaScript* функције које су еквивалент упитима над базом. Фолдер lists садржи листе. Листе су *JavaScript* функције које информације добијене од погледа обликују за приказивање, најчешће у *HTML* формат их обликују. Фолдер shows садржи дефиниције *JavaScript* функција које приказују најчешће један документ. Фолдер _attachments саджи фајл index.html и *JavaScript* фајл са функцијама које дефинишу „понашање" апликације.

Да бисмо повезали апликацију са базом података потребно је да је поставимо на север на ком је инсталиран систем *CouchDB*. У овом случају поставићемо апликацију на локални сервер localhost. Подразумевано *CouchDB* при раду са локалним сервером користи порт број 5984, те постављамо апликацију на адресу localhost:5984. Можемо у конзоли отворити фолдер који представља нашу апликацију (за дати пример то чинимо командом: cd proba). Алат *couchapp* омогућава постављање апликације на сервер командом:

couchapp push . http://localhost:5984/proba

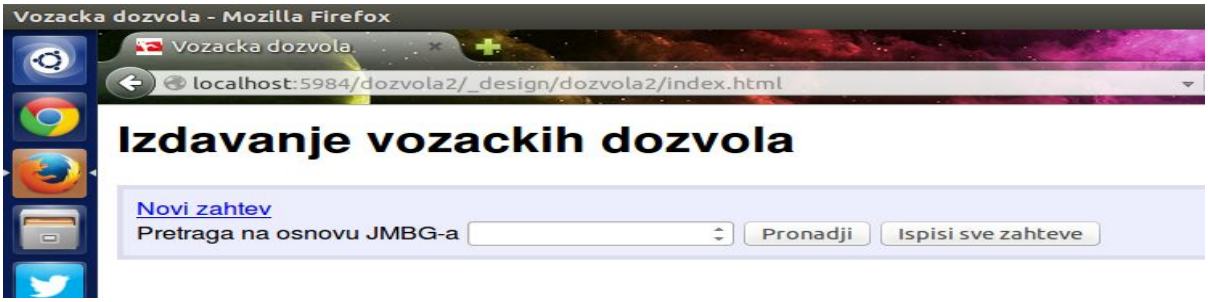
Ако је апликација успешно постављена добићемо обавештење са адресом почетне стране апликације:

[INFO] Visit your CouchApp here:
http://localhost:5984/proba/_design/proba/index.html

Претходном командом је креирана база података на основу параметара који су наведени у фајловима апликације. У фајлу _id наводи се име базе података коју ће апликација користити. База је иницијално празна. Дозвољен је „ручни" приступ бази на адреси localhost:5984/_utils ту се, помоћу *Futon* интерфејса, могу лако креирати, мењати и/или брисати документи одабране базе података.

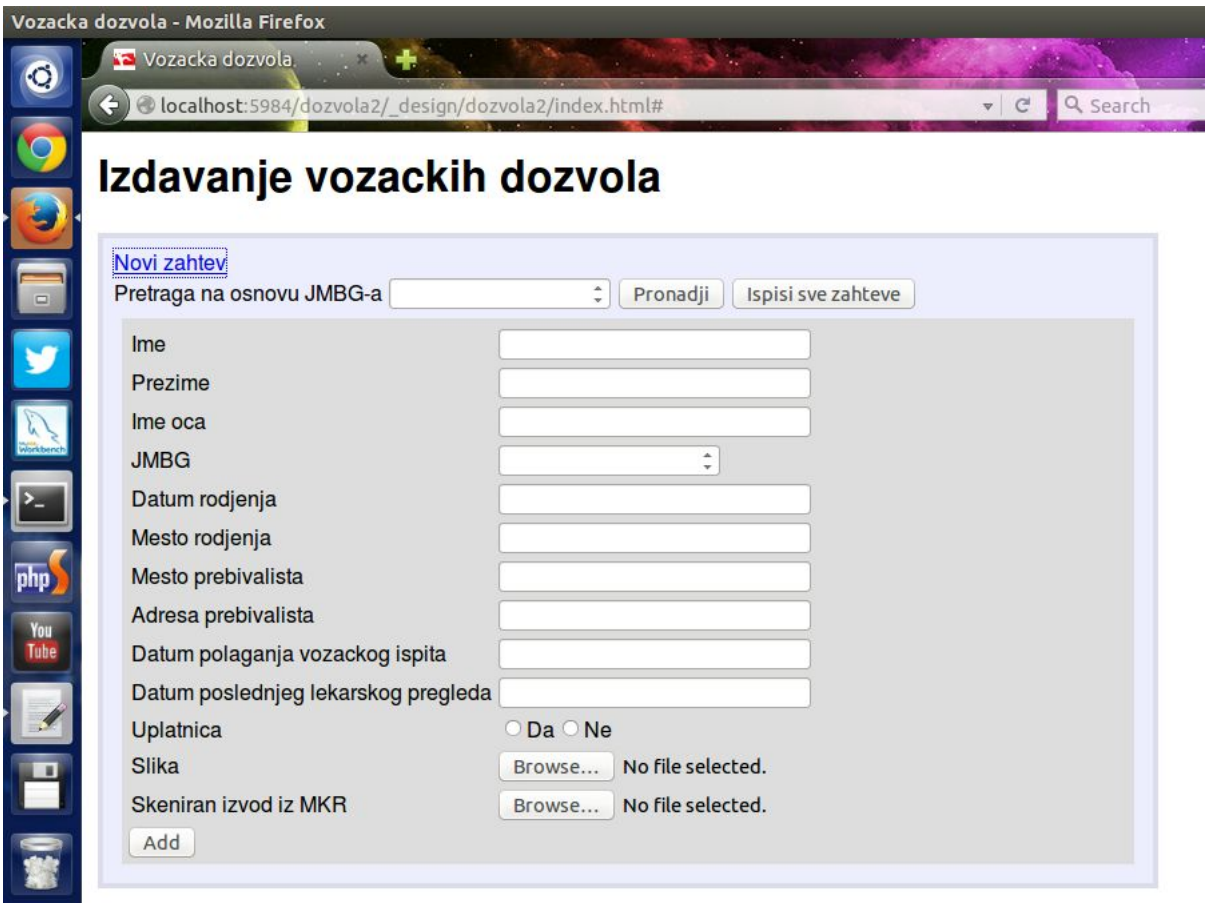
Део овог мастер рада је *CouchApp* апликација која представља образац за издавање возачких дозвола, у који се уносе основни подаци о подносиоцу захтева (име, презиме, ЈМБГ, имена родитеља, датум полагања возачког испита...), скенирани извод из матичне књиге рођених у *pdf* формату и слика подносилаца захтева у *jpg* формату је део овог мастер рада. У апликацији је искоришћена могућност *CouchDB*-а да чува податке различитих типова.

Апликација је тестирана на локалном серверу. При покретању апликације корисник видим понуђене опције за унос новог захетва, могућност да види све захтеве који су у бази и могућност претраживања по ЈМБГ-у подносиоца захтева (Слика 6).



Слика 6: Почетна страница апликације

За унос нових података корисник бира опцију „Novi zahtev” и добија образац који може да попуни, као на Слици 7. Сваки захтев за издавање возачке дозволе у бази података се чува као посебан документ. Поља „Slika” и „Skeniran izvod iz MKR” дозвољавају избор фајлова са рачунара.

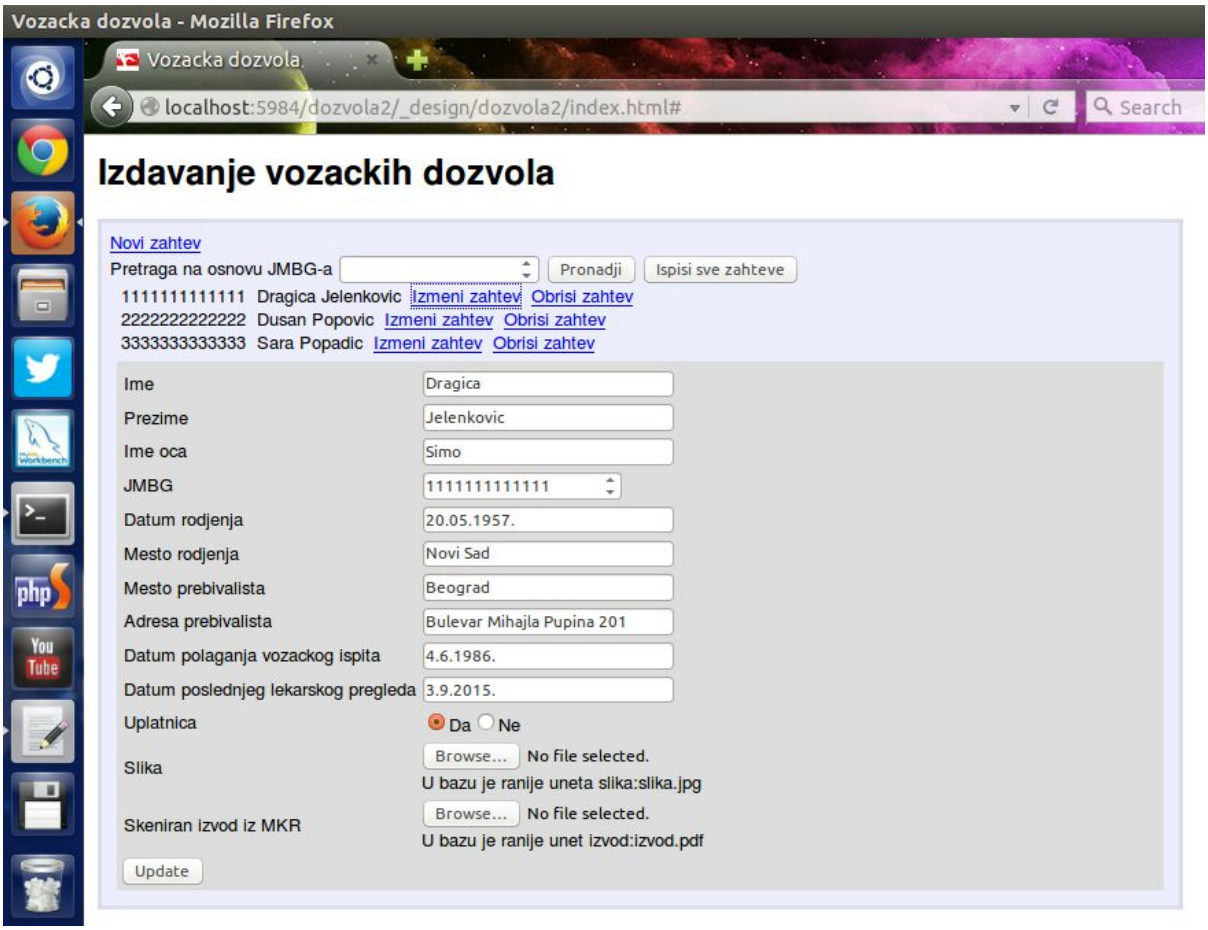


Слика 7: Уношење нових података у базу

Да би корисник видео све захтеве који постоје у бази може да изабере опцију „Ispisi sve zahteve”. Као резултат овог избора добија списак подносиоца захтева. Сваки захтев је представљен са три податка о подносиоцу захтева, а то су ЈМБГ, име и презиме уз које постоје опције „Izmeni zahtev” и „Izbrisi zahtev”, што је приказано на Слици 8.

Слика 8: Сви захтеви који се налазе у бази

Апликација омогућава измене постојећих захтева избором опције „Izmeni zahtev" поред јединственог матичног броја грађанина (ЈМБГ-а), имена и презимена подносиоца захтева. Корисник избором ове опције добија на увид све податке о подносиоцу захтева и може променити те податке (Слика 9).



Слика 9: Измене захтева

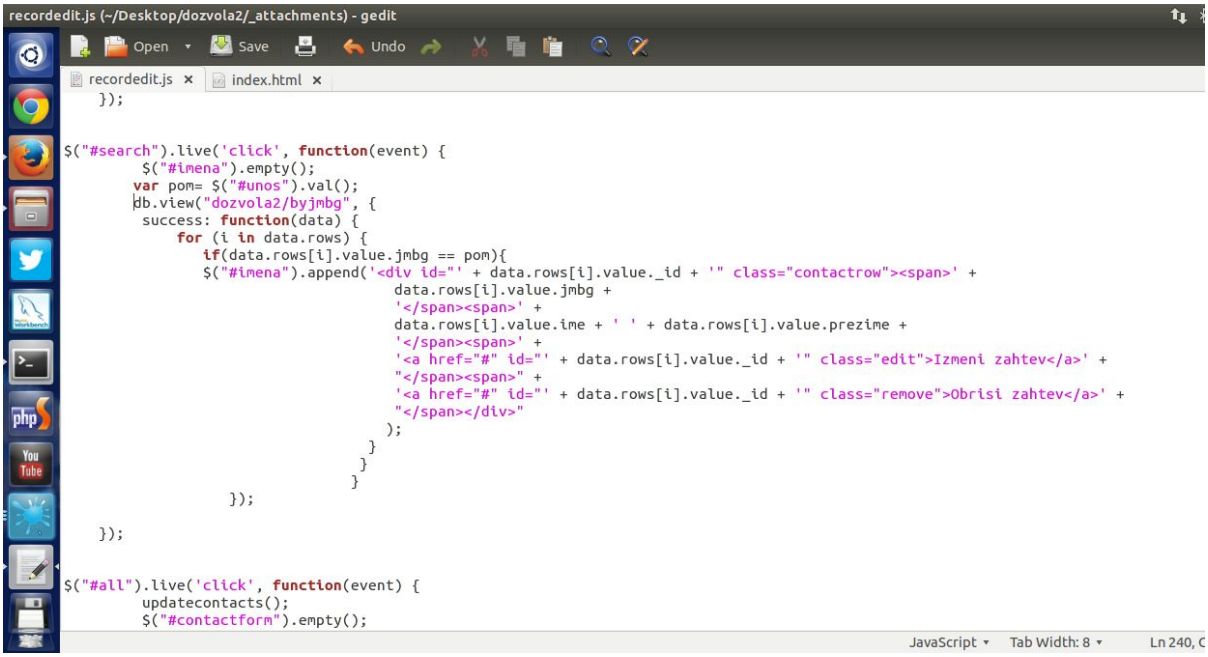
Корисник може по жељи обрисати неки од захтева избором, опције „Obrisi zahtev” која се односи на жељени захтев, назначене иза података о подносиоцу захтева. Да случано не би дошло до „случајног брисања” неког од захтева, поставља се питање кориснику да ли заиста жели да обрише захтев (Слика 10). Ако је захтев обрисан, страница се освежи и приказују се само постојећи захтеви у бази.

Слика 10: Брисање захтева

Корисник апликације може да унесе неки ЈМБГ и да види да ли у бази постоји подносилац захтева са унетим ЈМБГ-ом. То ће учинити тако што ће у поље за унос након лабеле „Pretraga na osnovu JMBG-a” унети жељени ЈМБГ и притиснути дугме „Pronadji”. Ако захтев са датим ЈМБГ-ом постоји, приказаће се само тај захтев на исти начин као на почетној страници апликације (ЈМБГ, име, презиме уз опције „Izmeni zahtev” и „Obrisi zahtev”), приказано на слици број 11. Ако захтев са датим ЈМБГ-ом не постоји, неће бити приказан ни један захтев.

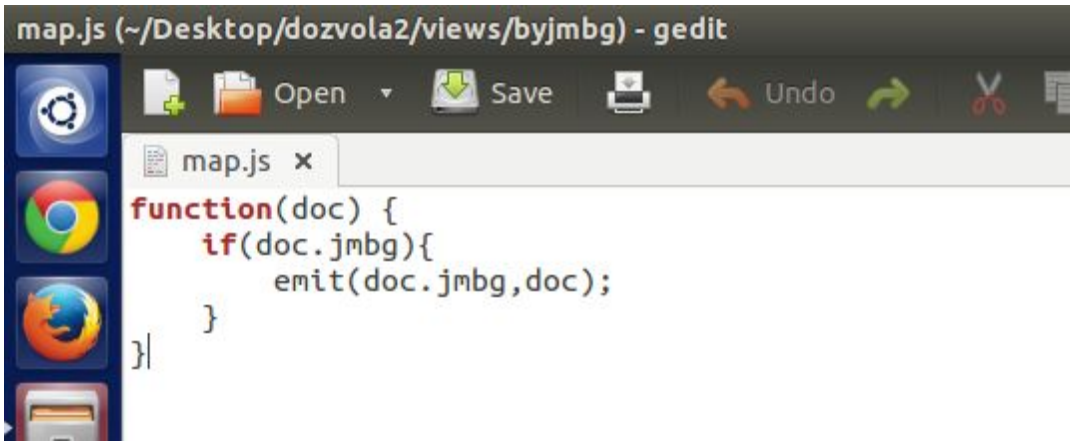
Слика 11: Претрага

Освежавање форме може се урадити кликом на дугме „Ispisi sve zahteve”. Претрага на основу ЈМБГ-а се ради коришћењем погледа byjmbg. Функција која дефинише понашање после клика на дугме „Pronadji” приказана је на Слици 12.



Слика 12: Део кода

Дефиниција погледа byjmbg приказана је на Слици 13.



Слика 13: Дефиниција погледа byjmbg

8. Закључак

Овај рад је показао основне карактеристике система и модела база података тзв. *NoSQL* база података и посебно основне карактеристике једне њихове врсте (документно оријентисаних система за управљање базама података) и једног

конкретног документно оријентисаног система *CouchDB*-а. *CouchDB* је врло једноставан за инсталацију и лако је започети са коришћењем овог система, што је добро јер код новог корисника не ствара отпор који би могао настати уколико би се на самом почетку сусрео са потешкоћама. За писање апликација на располагању је алат *CouchApp*, који убрзава прављење апликације и олакшава програмеру организовање самог пројекта. На програмеру је да измени одређене фајлове или дода нове и тиме постигне жељену функционалност аликације. Мана је то што не постоји довољно литературе и примера у вези са овим алатом и специфичном организацијом фолдера доступних на вебу, па неискуснији програмер може провести прилично времена покушавајући да разуме које измене су потребне да направи ради остваривања жељене функционалности. Сваки почетак је тежак. Међутим, већ следећа апликација ће бити написана за веома кратак временски рок. *CouchApp* апликације су првенствено замишљене као веб апликације, тако да нам алат *CouchApp* омогућава брзо и једноставно прављење веб апликација које користе *CouchDB* базу података. Грешке се јако једноставно и брзо отклањају, што је још једна од предности. Постављање апликације на сервер и истовремено креирање базе се врши једном командом у конзоли. Чињеница да се погледи (замена за упите) пишу на JavaScript језику олакшава учење, јер већина програмера која се бави веб програмирањем добро познаје овај језик. Још једна значајна предност *CouchDB*-а је што чува све претходне верзије података, па можемо имати увид како су се подаци мењали кроз време. Дакле, коришћењем технологија које нам нуди *Apache CouchDB* могуће је уз мање почетне напоре брзо правити веб апликације.

С обзиром на разноврсност потреба и захтева корисника као и разноврсност расположивих савремених технологија, новим тржиштима и специфичним случајевима употребе база података одговарају и специфична решења. *CoucheDB* као представник једне врсте *NoSQL* база података - документних база података је свакако једно од таквих решења за будућност.

Скраћенице коришћене у раду

SQL (енгл. *Structured Query Language*) - упитни језик релационих база података
СУБП - систем за управљање базама података
РСУБП - релациони систем за управљање базама података
JSON (енгл. *JavaScript Object Notation*) - формат за размену података
HTTP (енгл. *HyperText Transfer Protocol*) - мрежни протокол који представља главни и најчешћи метод преноса информација на вебу
HTML (енгл. *HyperText Markup Language)* - језик за означавање хипертекста
XML (енгл. *Extensible Markup Language*) - проширив језик за означавање
MVCC (енгл. *Multiversion concurrency control*) - метод за контролу конкуренткости који користе неки СУБП-ови да би омогућили конкурентни приступ бази података
API (енгл. *Application programming interface)* - Апликациони програмски интерфејс
EAV (енгл. *Entity–attribute–value model*) - модел података за опис ентитета
UUID (енгл. *Universally Unique Identifier*) - стандард за идентификаторе који се користи при прављењу софтвера. UUID је обична 128-битна вредност. Да би људима било читљив UUID запис може се представити у канонском облику, хексадекадни бројеви одвојени цртницама на стандардом одређеним местима.
URI (енгл. *Uniform Resource Locator*)
RCS (енгл. *Revision Control System*)
CVS (енгл. *Concurrent Versions System*)

Литература

1. Г. Павловић-Лажетић: *Увод у релационе базе података (друго издање)*, Математички факултет, Београд, 1999.
2. J. Chris Anderson,J. Lehnardt, N.Slater, *CouchDB: The Definitive Guide*, O'Reilly Media, 2010. http://buhoz.net/public/libros/db/CouchDB-TheDefinitiveGuide.pdf
3. Ozsu, Valduries, *Principles of Distributed Database Systems*, *3.ed,* Springer, 2011. [2.ed, Prentice Hall, 1999.]
4. Helal, Heddaya, Bhargava, *Replication Techniques in Distributed Systems*, Springer, 1996.
5. Chaudhri, Zicari, *Succeeding with Object Databases*, Wiley, 2010.
6. Date., *An Introduction to Database Systems*, *8.ed,* Addison-WesleyWiley, 2004.
7. Званични Apache CouchDB сајт http://couchdb.apache.org/ септембар 2015.
8. Hadoop-ов туторијал за MapReduce http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html септембар 2015.
9. Званични NoSQL сајт http://nosql-database.org/ септембар 2015.
10. http://www.christof-strauch.de/nosqldbs.pdf септембар 2015.
11. https://en.wikipedia.org/wiki/MapReduce септембар 2015.
12. http://science.webhostinggeeks.com/10-programskih-jezika септембар 2015.
13. Elmasri R, Navathe S B, *Fundamentals of Database Systems (шесто издање),* AddisonWesley, SAD, 2011. http://www.cse.hcmut.edu.vn/~ttqnguyet/CSDL/EbookDB.pdf
14. Званични сајт Erlang-a http://www.erlang.org/
15. http://opentextbc.ca/dbdesign/chapter/chapter-3-characteristics-and-benefits-of-a-database/
16. https://wiki.apache.org/couchdb/Performance
17. http://stackoverflow.com/questions/12346326/nosql-cap-theorem-availability-and-partition-tolerance
18. http://wiki.apache.org/couchdb/Technical%20Overview