SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Domagoj Perko

CouchDB

ZAVRŠNI RAD

Varaždin, 2017.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Domagoj Perko
Matični broj: 34436/04–I
Studij: Informacijski sustavi

CouchDB

ZAVRŠNI RAD

Mentor:
Doc.dr.sc. Markus Schatten

Varaždin, veljača 2017.

Sadržaj

1. Uvod

U ovom završnom radu obradit će se tema CouchDB. Tematski je rad podijeljen u 3 glavne cjeline; prvi dio vezan uz generalni rad s NoSQL bazama podataka, drugi s težištem na CouchDB, te dio vezan uz izradu manje pomoćne aplikacije koja bi koristila CouchDB za bazu podataka. Ova poglavlja zaokružena su uvodom i zaključkom te popisom korištene literature.

U toku školovanja većine današnjih informatičara najveći fokus stavlja se (s pravom) na relacijske baze podataka, odnosno sustave za upravljanje istima. Oni čine većinu funkcionalnih sustava, te tabličnom pohranom i prikazom podataka zadovoljavaju veliku većinu današnjih zahtjeva. Ipak, iz razloga upoznavanja nešto šire slike danas dostupnih sustava za upravljanje bazama podataka, odabran je CouchDB kao zanimljiv kandidat iz grupe NoSQL (Not only SQL) baza podataka. Ono što je pomoglo u odabiru CouchDB-a konkretno je naglasak autora na opuštenom pristupu, jer se na više mjesta na službenim stranicama i u dokumentaciji spominje riječ relax, odnosno opušteno. U radu će se istražiti valjanost ove tvrdnje, te provjeriti upotrebljivost ovakvog sustava u manjem radnom okruženju, kao alternativu postojećoj razmjeni dokumenata.

2. NoSQL baze podataka

U svijetu baza podataka dugo vremena dominira jedna vrsta – relacijske baze podataka. povezane tablice (relacije) su se pokazale iznimno robusnim i efikasnim načinom prikaza svih potrebnih podataka u raznolikim scenarijima upotrebe.

Ipak, kao jedan od glavnih nedostataka pojavljuje se manjak fleksibilnosti kod manje strukturiranih baza podataka ili baza podataka koje često mijenjaju te imaju „neprirodan“ opis podataka što otežava svrstavanje u savršeno strukturirane tablice.

Kod kreiranja modela većine relacijskih baza podataka, jedna od temeljnih pretpostavki je dobro poznavanje podataka koji se u tu bazu podataka namjeravaju pohraniti. Na osmišljavanje i izgradnju opisne strukture podataka otpada značajan dio početnih aktivnosti vezanih uz samu izradu. To ponekad uključuje i detaljno istraživanje samog seta podataka kako bi se isti što bolje opisali u prvom pokušaju te smanjili potrebu za izmjenama shematskog prikaza u kasnijim koracima razvoja.

Većina NoSQL (Not only SQL) baza podataka, odnosno sustava za upravljanje istima, ima veću fleksibilnost kod pojave ovakvih problema, jer se opis podataka nalazi među samim podatcima, te ga dobivamo po potrebi iz dokumenata ili podataka koje zaprimimo.

U ovom radu bit će prikazan sustav za upravljanje jednom od takvih baza podataka – CouchDB, a stranica nosql-database.org navodi više od 225 najpoznatija sustava podijeljenih u sljedeće generalne skupine:

- Wide Column Store / Column families
- Spremište dokumenata
- Ključ-vrijednost
- Grafovi
- Višemodelne

2.1. Ključ-vrijednost NoSQL baze podataka

Neki od najpoznatijih predstavnika ovih baza podataka su Riak i Amazonov Dynamo. Ključ-vrijednost baze podataka drže se pravila da shema ne postoji. Ključ može biti dodijeljena vrijednost ili automatski generirana, dok vrijednost može biti String, JSON, BLOB i slično. Tip ključ-vrijednost u osnovi koristi hash tablicu u kojoj postoji unikatni ključ i pokazivač na određeni podatak. Košara (eng. Bucket) je logička skupina ključeva, ali ona ne grupira podatke fizički. Mogu postojati identični ključevi u raznim „košarama“. Ovakve baze podataka se fokusiraju na dostupnost i particioniranje, ali su slabije kod konzistentnosti.

Primjer:

Ključ	Vrijednost
Hvar	{“Caffe Minolta”,“Plaža jug“ }
Brac	{“Restoran zlatni Rat”}
Krk	{“Zračna luka”,“BP Malinska“,“TP Valbiska“}

Ključ-vrijednost baze podataka omogućavaju korisnicima upisivanje i čitanje vrijednosti korištenjem ključeva na sljedeći način:

- Get(key), vraća vrijednost dodijeljenu ključu
- Put(key, value), dodjeljuje vrijednost ključu
- Multi-get(key1, key2, .., keyN), vraća listu vrijednosti dodijeljeni listi ključeva
- Delete(key), Briše unos za ključ

Ovakve baze podataka mogu biti zanimljive i korisne u određenim prigodama, ali dolaze s određenim problemima. Model podataka nema većine tradicionalnih mogućnosti drugih baza podataka kao npr. atomičnost transakcija, ili dosljednost kad se više transakcija odvija istovremeno. Naravno, kreatori aplikacija imaju mogućnost ove nedostatke pokriti mogućnostima samih aplikacija. Uz to, s povećanjem baze podataka i vrijednosti koje su u nju unošene dolazi do potrebe za uvođenjem novih načina generiranja ključeva kako bi oni ostali jedinstveni.

2.2. Spremište dokumenata

Najpoznatiji predstavnici spremišta dokumenata su CouchDB koji će biti detaljno obrađen u ovom radu te MongoDB. Spremišta dokumenata su na neki način nadgradnja baza podataka ključ-vrijednost. Razlika kod spremišta dokumenata je da spremljene vrijednosti, odnosno dokumenti, uključuju razinu strukturiranja samih podataka. Najčešće se koriste XML, JSON i BSON, iako ne postoji ograničenje na iste. Primjer:

{id:"Hvar",
{Lokacija_bankomata: "Caffe Minolta"}}
{id:"Brac",
{Opis:"Otok u sredisnjoj dalmaciji"}}
{id:Krk",
{Velicina:"veliki otok"}}

U primjeru je vidljiv niz vrijednosti spremljen u dokument, s nazivima raznih otoka na obali. Isto tako svi modeli podataka su različiti no to ne predstavlja problem jer ne postoji fiksna shema.

2.3. Spremište kolona

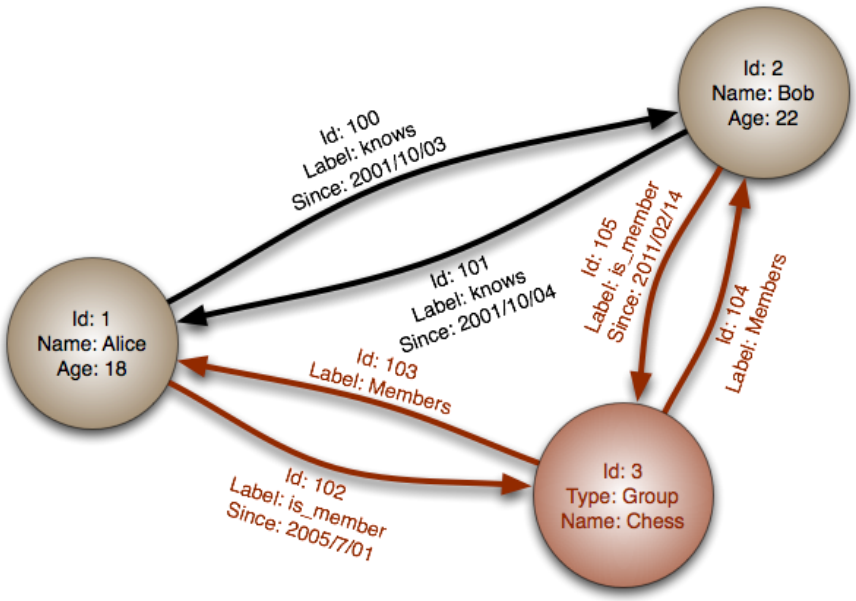
Sljedeći oblik NoSQL baza podataka su Spremišta kolona. Neki od poznatijih sustava koji koriste ovakve baze podataka su Googleov BigTable, Hbase i Cassandra. U ovakvim bazama podataka, podatci su spremljeni u ćelije grupirane u kolone s podatcima, za razliku od klasičnih baza podataka koje podatke spremaju u redove. Kolone se logički grupiraju u familije kolona koje mogu sadržavati beskonačan broj istih. Sve operacije nad bazom podataka rade se korištenjem kolona umjesto redaka što u teoriji omogućava bržu pretragu podataka jer su istoznačni podatci pohranjeni na obližnjim mjestima na disku.

Prema Girish Kumar (2014) Model podataka ovakvih baza podataka temelji se na:

- ColumnFamily – struktura koja može grupirati kolone i superkolone
- Key – trajno ime zapisa. Ključevi mogu imati raznolik broj kolona
- Keyspace – definira vanjsku razinu organizacije – najčešće naziv aplikacije
- Column – sadrži uređenu listu elemenata s ključem i vrijednosti

2.4. Baze grafova

Možda najapstraktnija varijanta NoSQL baza su baze grafova. U ovakvim bazama podataka ne postoji klasičan prikaz pomoću tablica i kolona nego se koristi grafički prikaz. Strukture grafa su prikazane s rubovima, čvorovima i svojstvima bez indeksa. Primjer prikaza opisan je na sljedećoj slici:



Slika 1. Graf (Izvor: Girish Kumar 2014)

2.5. Polu-strukturirani podatci i baze podataka

Kod polustrukturiranih podataka, informacije koje su obično spremljene u schemi, su sadržane u samim podatcima, što se najčešće zove i "samoopisni" način zapisa. U nekim tipovima polustrukturiranih baza podataka ne postoji odvojena shema, dok u nekima postoji u blažem obliku, te opisuje samo blage restrikcije nad podatcima.

Razlozi sve češćeg pojavljivanja teme polustrukturiranih baza podataka zbog pojave izvora podataka koje je teško opisati jednostavnom shemom. Primjer takvog izvora podataka je Web. Drugi razlog je to što je ponekad poželjno ostaviti format dovoljno fleksibilnim kod razmjene podatak između različitih baza podataka. Uz to, čak i kod rada s nekim strukturiranim podatcima, nekad ih je poželjno prikazati kao polustrukturirane za svrhe pregleda.

Schatten i Ivković (2012) prilikom predstavljanja upitnog jezika nad grafovima govore detaljnije o polu-strukturiranim podatcima koji čine osnovicu za veliku većinu današnjih NoSQL baza podataka. U radu je predstavljen Regular Path Expression – programski jezik za upravljanje polustrukturiranim podatcima. Detalji rada dostupni su na https://bib.irb.hr/datoteka/594133.SchattenIvkovic2012.pdf.

Manjak strukture kod podataka očituje se u teškoj implementaciji aplikacija i programskih dijelova, obzirom na manju pripremljenost podataka za obradu na strani same baze podataka. S druge strane, nepostojanje ograničenja dozvoljava olakšava posao na ulaznom dijelu podataka. Ukratko, radi se o jednostavnom unosu koji zahtjeva kompliciranje prilikom očitavanja, za razliku od striktno strukturiranih baza podataka gdje dodatna ograničenja prilikom unosa omogućuju preskakanje već odrađenih dijelova kod upita/čitanja.

2.6. Primjer razlike u zapisima SQL i NoSQL baze podataka

U tablici ispod prikazano je trenutno stanje (pojednostavljen prikaz sa samo nekoliko kolona) vezanih za popis bankomata s određenim vrijednostima. Budući da se broj kolona povećavao kroz vrijeme (kako se realizirala potreba za postojanjem dodatnih polja), neki od ranije upisanih redaka imaju znatnu količinu NULL vrijednosti pohranjenih među vrijednostima.

ATM_ID	Model	Lightbox	Postolje	Contactless	Site_ID
55101	TTW	Vertikalni	NULL	NULL	1001
55102	TTW	Horizontalni	NULL	NULL	1002
55103	FSW	Horizontalni	4cm	NULL	1003

Tablica 1. Bankomati

Sličan problem postoji i u tablicama vezanima za lokacije:

Site_ID	Adresa	Winterizacija 2012	Kontakt
55101	TTW	11.10.2012	NULL
55102	TTW	14.11.2012	Aco Acić
55103	FSW	NULL	Beco Becić

Tablica 2. Lokacije

Iz primjera je vidljivo da SQL tablice koriste identifikator u vidi ključa po kojemu se može jednoznačno odrediti redak. U ovim primjerima radi se o ATM_ID i Site_ID kolonama. U praksi se u tablici Bankomati koristi vanjski ključ Site_ID koji upućuje na lokacije (vidljiv u tablicama), te ostali vanjski ključevi koji nisu predmet primjera. Povezivanjem raznih tablica dobiva se moćan sustav za pohranu koji je u posljednjim desetljećima vladar na području baza podataka.

NoSQL baza podataka se fokusira na drukčiji pristup, a jedna od glavnih razlika je smanjen, odnosno gotovo nepostojeći fokus na shemu, odnosno pravila, rasporede i formate koji ograničavaju unos podataka u predefiniran kontekst. Ukoliko bismo za npr bankomat 55101 željeli dodati informaciju da se npr koristi specijalni branding, ili za određenu lokaciji dodati napomenu da je potrebna najava dolaska, bilo bi potrebno kreirati nove kolone, odnosno proširiti shemu iste – a to za posljedicu ima utjecaj na eventualne aplikacije koje su izrađene s pretpostavkom neke ranije postojeće sheme.

U NoSQL bazama podataka, a posebno onima fokusiranim na dokumente, dokumenti sami sadrže opis podataka skupa s podatcima, te su tako samostalni i mogu shematski razlikovati od dokumenta do dokumenta, iako bi se možda radilo o sličnim podatcima (npr. Dokument s detaljima o više lokacija).
Primjer dokumenta:

{id:”55101”, {Model: “TTW”, Lightbox: “Vertikalni“ }}
{id:”55102”, {Model: “TTW”, Lightbox: “Horizontalni“ }}
{id:”55103”, {Model: “TTW”, Lightbox: “Horizontalni“, Postolje: “4cm“}}

2.7.Prednosti i nedostatci NoSQL

Prilikom izrade ovog rada dobiven je uvid u glavne nedostatke NoSQL sustava. Radi se o relativno novom načinu izrade baza podataka, a i samog razmišljanja. Ovo za posljedicu ima manjak kvalitetnih izvora podataka za učenje jer su skoro sve varijante ovih baza podataka ograničene na maksimalno jednu do dvije knjige, ponešto dokumentacije, te velik broj neodgovorenih pitanja kod svake nove verzije sustava. NoSQL baze podataka su nedostatkom vremena razvoja još uvijek relativno nestabilne te se učestalo događaju nekompatibilnosti s raznim verzijama pomoćnog softvera.

Prilikom potrage za podrškom kod određenih pitanja vezano za kreaciju aplikacije na primjeru CouchDB-a utvrđeno je da službeni support pružaju svega dvije specijalizirane tvrtke, od kojih je jedna individualno djelo entuzijasta CouchDB-a. Kapaciteti podrške su samim time drastično manji od podrške kod SQL sustava koji imaju razgranatu mrežu tvrtki koje ih podržavaju.

Iako se NoSQL baze podataka hvale jednostavnošću pohrane podataka, one ipak zahtijevaju tehničko osoblje od same instalacije do administracije sustava. Uz sve to, primjetan je i nedovoljan broj programa za učenje te nedovoljan broj eksperata koji bi takve programe provodili.

Jenny Richards (2015) navodi sljedeću listu prednosti i nedostataka:

Nedostatci:

- Manja zrelost
- Slabija podrška
- Smanjena poslovna analitičnost (business intelligence)
- Potreba za administracijom
- Nedostatak ekspertize

Prednosti:

- Skalabilnost
- Primjena kod masivnih baza podataka
- Niža potreba za administracijom u odnosu na RBDMS
- Cijena (manja potreba za skupim hardverom)

Iako lista navodi veći broj nedostataka, treba imati na umu da RBDMS kao već zreli sustav, ima manje prostora za napredak, pa u budućnosti može doći do znatnog smanjenja nedostataka NoSQL sustava, dok je potencijal za prednosti još uvijek velik. Vrijeme će pokazati dali NoSQL sustavi mogu zauzeti veći udio u pohrani podataka te se maknuti s marginalnih vrijednosti koje trenutno ima.

3. CouchDB

Apache CouchDB je softver za upravljanje bazama podataka otvorenog koda. Arhitekturom se radi o NoSQL bazi podataka s fokusom na dokumente a razvijen je korištenjem Erlang jezika. Koristi JSON za pohranu podataka, JavaScript za upitni jezik i HTTP za API.

CouchDB je prvotno objavljen u 2005. godini, a dio Apache Software Foundationa postaje 2008. Autori CouchDB-a su Damien Katz, Jan Lehnardt, Noah Slater, Christopher Lenz, J. Chris Anderson, Paul Davis, Adam Kocoloski, Jason Davies, Benoît Chesneau, Filipe Manana i Robert Newson.

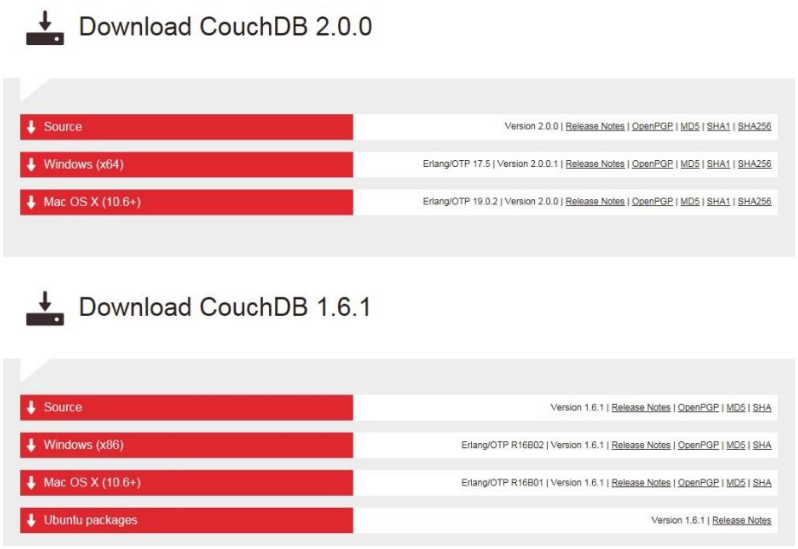
Za razliku od relacijskih baza podataka, CouchDB ne pohranjuje podatke i njihove međuodnose u tablice. Svaka CouchDB baza podataka je skup neovisnih dokumenata koji sadrže svoje podatke i opise istih. Aplikacije koje se oslanjaju na CouchDB mogu pristupiti višestrukim bazama podataka, te je jedna od najčešćih upotreba korištenje baze podataka na nekom prijenosnom sredstvu (laptop, mobitel) koja se na zahtjev ili kontinuirano sinkronizira s bazom podataka na serveru. U opisu podataka nalaze se podatci o revizijama što omogućava spajanje svih razlika koje su se dogodile dok su baze podataka bile odspojene.

3.1. Korisnici CouchDB-a:

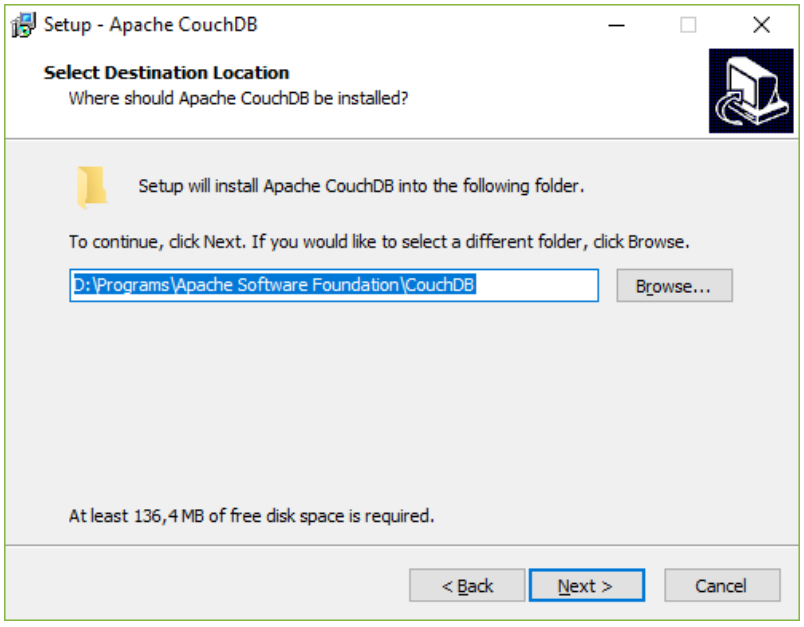
- Amadeus IT Group, za back-end sustave
- Credit Suisse, za internu upotrebu
- Meebo, danas ugašena platforma za socijalizaciju.
- Sophos, za neke back-end sustave.
- BBC, za platformu s dinamičkim sadržajem
- Canonical za sinkronizacijski servis "Ubuntu One", od 2009. do 2011.
- AirFi zaInflight Entertainment platformu.
- CANAL+ za međunarodnu platformu CANAL+ Overseas.

3.2. Instalacija CouchDB:

Stranica http://couchdb.apache.org sadrži poveznice za sve verzije sustava. U ovom radu korišten je sustav 1.6.1 iako je u međuvremenu dostupna i verzija 2.0. Za instalaciju je potrebno posjetiti navedenu stranicu te preuzeti instalacijske datoteke. Podržani operativni sustavi su Linux, Windows i MacOS iako verzija 2.0 za sad nema spremnu instalacijsku datoteku za Linux. Jednom preuzete datoteke su identične za razne operativne sustave, iako postoje manje razlike prilikom instalacije.



Slika 2. Dostupne verzije CouchDB



Slika 3. Instalacija paketa na windows platformi

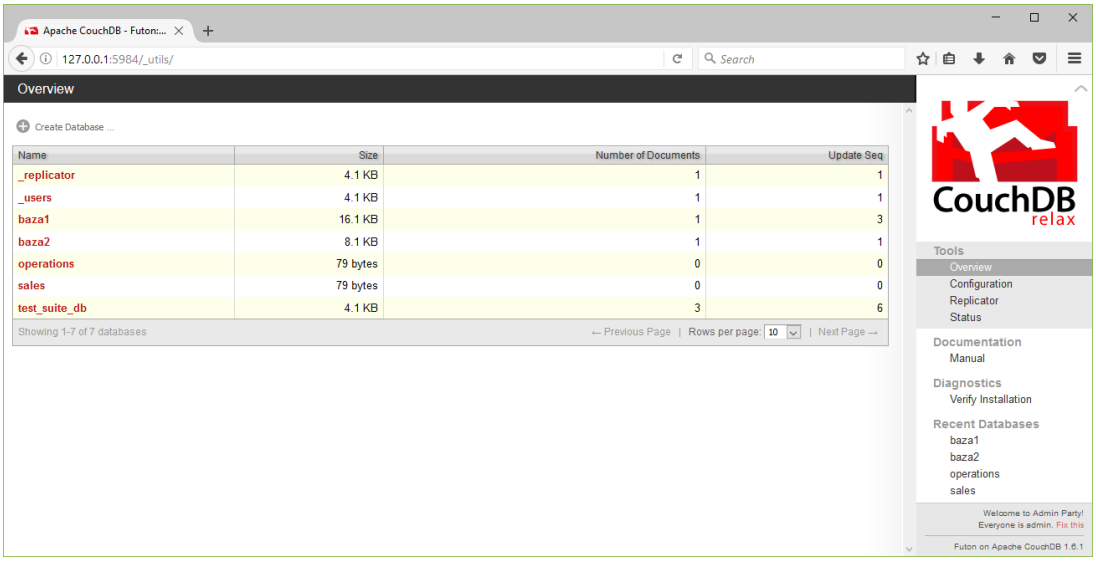
3.3. Grafičko sučelje CouchDB (Futon i Fauxton)

Prilikom instalacije CouchDB baze podataka, u paketu je uključeno grafičko sučelje kojemu se pristupa preko web preglednika. U verziji 1.6.1. radi se o Futon Sučelju koje je prikazano na slici ispod. U verziji 2.0 napravljen je redizajn sučelja te dodane nove mogućnosti što je objavljeno pod novim imenom – Fauxton.

Osnovni pogled čini početna stranica s pregledom svih postojećih baza podataka u instanci CouchDB. U glavnom prozoru moguće je navigirati po bazama podataka, dokumentima te raditi operacije nad njima. Pomoćni panel sa strane koristi se za navigaciju do glavnih dijelova sučelja.

Kao što je vidljivo iz slike, sastavnice pomoćnog panela su:

- Overview (pregled)
- Configuration (sučelje za konfiguraciju CouchDB instalacije)
- Replicator (sučelje za repliciranje između više baza podataka, bez obzira radi li se o bazama podataka u istoj instanci CouchDB-a ili više različitih.
- Status – lista aktivnih zadataka na serveru
- Verify Installation – link za verifikaciju instalacije – još uvijek dosta bugovit te zna prikazivati greške i kad ne postoje.
- Test Suite – paket testnih aktivnosti za provjeru funkcionalnosti instalacije.



Slika 4. Futon grafičko sučelje

3.4. Pristupanje podatcima preko HTTP-a

Aplikacije ostvaruju interakciju s CouchDB-om preko HTTP-a. U tablici ispod prikazani su neki osnovni primjeri interakcije korištenjem cURL, komandno-linijskog alata. Pretpostavka je da je CouchDB postavljen na localhost (127.0.0.1) , na portu 5984 (automatska pretpostavljena adresa kod instalacije).

Aktivnost	Komanda	Rezultat
Dohvaćanje informacija o serveru	curl http://127.0.0.1:5984/	{ "couchdb": "Welcome", "version":"1.6.0"}
Kreiranje baze podataka s nazivom aparati	curl -X PUT http://127.0.0.1:5984/aparati	{"ok": true}
Pokušaj kreiranja druge baze podataka s nazivom aparati	curl -X PUT http://127.0.0.1:5984/aparati	{ "error":"file_exists", "reason":"The database could not be created, the file already exists."}
Brisanje baze podataka aparati	curl -X DELETE http://127.0.0.1:5984/aparati	{"ok": true}

Tablica 3. Primjeri HTTP komandi u interakciji s CouchDB

CouchDB podržava sljedeće HTTP request metode:

- GET – zahtjeva specifičan podatak. U CouchDB-u to može biti dokument, konfiguracijski podatak ili npr neka statistička informacija
- HEAD – metoda koja se koristi za dobivanje HTTP headera za rezultat GET zahtjeva bez sadržaja odgovora.
- POST – postavljanje (upload) podataka. U CouchDB-u POST se koristi za postavljanje vrijednosti, dokumenata, ili pokretanje administracijskih komandi
- PUT – koristi se za postavljanje specifičnog resursa te je najčešće korišten kod kreiranja novih baza podataka ili dokumenata
- DELETE – briše određeni resurs
- COPY – metoda koja se koristi za kopiranje dokumenata i objekata

3.5. Prednosti CouchDB

CouchDB temelji svoj dizajn na prilagodljivosti i realnom prikazu stvarnih podataka. Kao dodatak tom principu dodane su moćne mogućnosti pretrage, filtriranja i mapiranja podataka/dokumenata. Iz samog dizajna proizlazi i velika skalabilnost, otpornost na pogreške i mogućnost replikacije što daje zaokružen dojam cijelom paketu.

Obzirom da srž računalnih, a i svih ostalih sustava u korijenu ima poboljšavanje ili olakšavanje života ljudi, kreatora i korisnika, tako i CouchDB svoju viziju postojanja temelji na olakšavanju aktivnosti vezano za određeni dio stvarnog života, ovaj put s fokusom na dokumente koji su sastavni dio svakodnevice. Mnoštvo informacija i podataka koje zaprimimo su u obliku jednostavnog dokumenta ili popisa vrijednosti koji nisu nužno prilagođeni spremanju u tablične formate.

CouchDB pretpostavlja da stvari ne moraju uvijek funkcionirati savršeno, te podrazumijeva da su barem neke od sljedećih tvrdnje netočne.

Prema Gosling J (1997) zablude kod mrežnog rada uključuju :

1. Mreža je tehnički pouzdana
2. Latencija je jednaka 0
3. Mrežna propusnost nije ograničena
4. Mreža je sigurna
5. Mreža nema topološke izmjene
6. Postoji samo jedan administrator
7. Trošak transporta je 0
8. Mreža je homogena

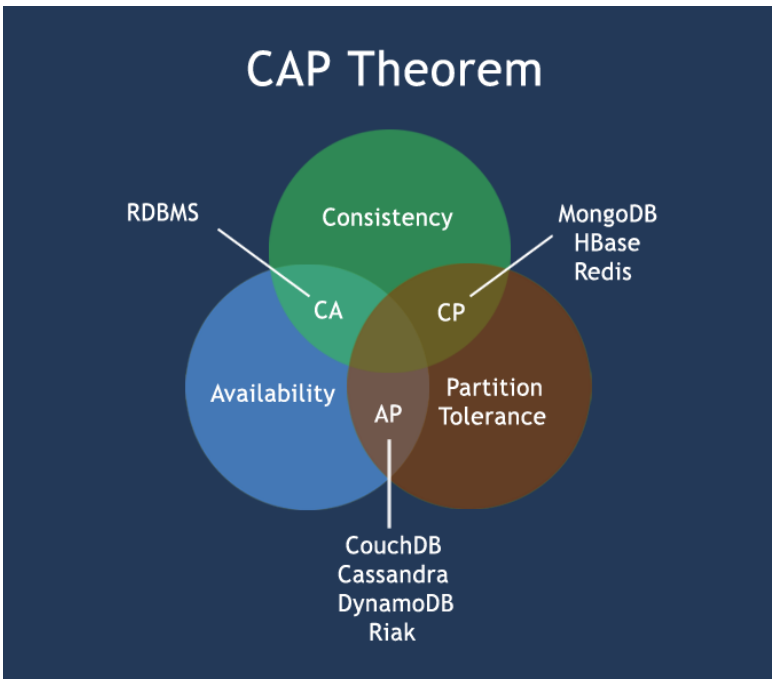
Postojeći alati često pokušavaju ignorirati činjenicu da neka ili čak sve navedene izjave nisu točne kod određenih sustava. S druge strane CouchDB ne skriva mrežu, nego pokušava riješiti eventualne nastale probleme, a u slučaju nemogućnosti zatraži aktivnost korisnika/administratora.

3.6. CAP teorem i krajnja dosljednost (eventualna konzistencija)

Sustav koji planski funkcionira na mreži s više čvorova zovemo distribuirani sustav. Jedan od glavnih problema koje takvi sustavi susreću odnose se na samo funkcioniranje mreže, odnosno postojanost linkova između čvorova. Samim time, posljedično su nastale brojne strategije vezane za dostupnost podataka na navedenim sustavima - s težištima na raznolikim vrijednostima.

CAP teorem prikazuje presjek različitih težišta važnosti kod raznih sustava. Kao glavne težišne točke navode se:

1. Dosljednost - svi klijenti vide identičan set podataka
2. Dostupnost - svi klijenti mogu pristupiti nekoj verziji podataka
3. Tolerancija na fragmentiranje - mogućnost raspodjele podataka na više poslužitelja



Slika 5. Cap Teorem (Izvor: Abram Simon (2014))

Kao što je vidljivo na slici, fokusom na 2 od 3 težišta smanjuje se izravna dodirna površina s trećom, međutim, to se donekle nadoknađuje naknadnim funkcionalnostima, što će u ovom radu biti prikazano na primjeru CouchDB i dosljednosti.

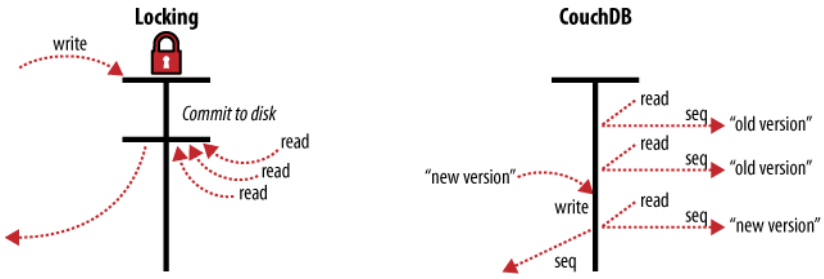
3.7. Lokalna dosljednost

Srž CouchDB čini moćan način spremanja podataka zasnovan na binarnom stablu. Binarno stablo je struktura podataka koja omogućava pretragu, dodavanja i brisanja u relativno brzom vremenu. Korištenje binarnog stabla nije ograničeno samo na dokumente nego se odnosi na sve pohranjene elemente.

Kako bi se najbolje pojasnio način na koji CouchDB upotrebljava pohranjene podatke, valja opisati postupak koji se zove MapReduce. Radi se o kombinaciji dvije riječi, Map i Reduce, koje same po sebi odgovaraju dodijeljenim istoimenim funkcijama. Ove funkcije se jedna za drugom vrše nad željenim dokumentom ili skupom dokumenata. Ove funkcije u konačnici donose parove ključ/vrijednost, što CouchDB onda može ponovno pospremiti u binarno stablo, sortirano po ključu. Mogućnost da se u konačnici rezultat pregleda prema ključu ili rasponu ključeva u ovakvom je načinu spremanja iznimno efikasna.

Tablica u klasičnoj relacijskoj bazi podataka je jedna samostalna struktura podataka. U slučaju da želimo modificirati tablicu radi ažuriranja potrebno je osigurati da nitko drugi nema pristup dijelu baze podataka koji se ažurira. Ovo se najčešće radi zaključavanjem po „first come“ principu – odnosno onemogućavanjem korisnika koji nisu prvi zauzeli tablicu/dokument da rade izmjene sve dok prvi korisnik nije gotov s istima. Tablica se zatim serijski prosljeđuje na rad drugom korisniku i tako dalje. Ovakav način rada u nekim situacijama zna dovesti i do situacije da se više resursa troši na odlučivanje oko toga tko ima pristup tablici i kojim redom nego na sam rad. U primjeru koji se koristi kroz ovaj rad često se događa da se tablice koje su spremljene ili lokalno kao odvojeni dokumenti, ili dijeljenje tablice na npr. Microsoft Sharepoint platformi – te je jedna od pretpostavki pokušaja ostvarenja boljih rezultata korištenjem CouchDB-a uzrokovana istim.

CouchDB koristi MVCC (Multi-Version Concurrency Control) kako bi se riješio problem pristupa bazi podataka. Ono što se u konačnici događa je da CouchDB ima na raspolaganju sve ili većinu resursa na raspolaganju bez obzira koliku opterećenost korisnicima i zahtjevima ima.



Slika 6. CouchDB u usporedbi s klasičnim zaključavanjem (Izvor: Anderson C, Lehnardt J, Slater N (2010))

CouchDB može validirati dokumente korištenjem ugrađenih JavaScript funkcija. Kod svake modifikacije dokumenata odradit će se željene validacijske funkcije te u konačnici odobriti ili odbiti ažur. Korištenjem ovakvih funkcija smanjuje se potreba za dodatnim provjerama na razini aplikacija. Ovaj dio će biti detaljnije opisan kasnije s funkcijama validacije.

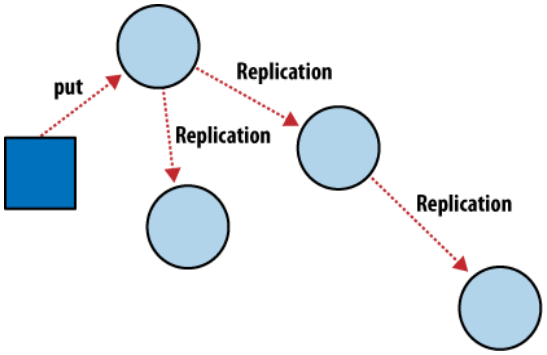
3.8. Distribuirana dosljednost

Ranije spomenuta dosljednost poprima sasvim nove dimenzije jednom kad se problem odnosi na cijelu mrežu čvorova. Dok većina baza podataka dosta uspješno rješava probleme na pojedinim čvorovima, kod održavanja dosljednosti na više servera postavljaju se dodatna pitanja na koja je potrebno pronaći rješenja. Ukoliko klijent ažurira dokument na jednom serveru potrebno je se pobrinuti da taj ažur bude proslijeđen i na ostale čvorove u mreži/klasteru.

Kod relacijskih baza podataka ovaj problem je kompleksan te postoji cijela teorija s raznim oblicima rješenja (npr. multi-master, master/slave, particioniranje, sharding itd.)

3.9.Postepena replikacija

Kod postepena (inkrementalne) replikacije dokumenti se sporadično/periodički izmjenjuju među različitim čvorovima. Ovaj process „kad-tad“ dovodi do dosljednosti između svih čvorova. Proces replikacije između dvije baze podataka može se odvijati neovisno o postojanju drugih čvorova što je prikazano na slici



Slika 7. – postepena replikacija (Izvor: Anderson C, Lehnardt J, Slater N (2010))

Obzirom na navedeno, moguće je koristiti razne metode sinkronizacije kao što su npr alati za sporadično zadavanje zadataka (eng. Schedulers), bez ograničenja i razmišljanja o ukupnom broju baza podataka i čvorišta. Svaka pojedina baza podataka može se koristiti samostalno, a promjene se mogu sinkronizirati kasnije propagiranjem u svim smjerovima. U slučajevima kad je se na više čvorova ažurira isti dokument CouchDB donosi i automatsku detekciju i rješavanje konflikata, što će detaljnije biti opisano u radu pod metodama rješavanja konflikta.

4. Razvoj uz CouchDB

4.1. Modeliranje dokumenata

Jedna od jednostavnijih podjela dokumenata su „konkretni“ dokumenti i „virtualni“ dokumenti. Konkretan dokument je nešto što bi odgovaralo dokumentu u stvarnom životu, ili nečemu kreiranom u tekst procesoru. Ovdje je najčešće prikaz dokumenta jednak njegovom sadržaju. Sve bitno se nalazi na jednom mjestu i s njega se, kroz razne metode, prikazuje krajnjem korisniku.

Virtualni dokumenti koriste „poglede“, odnosno „view“ metode kako bi skupili sadržaj iz jednog ili više izvora u jedan pregledan skup.

4.2. Pogledi

Pogledi su korisni na više načina. Bez njih, komplicirao bi se svaki upit koji se radi na bazu podataka, a u slučaju ponavljanja dijelova upita, postojanje virtualnog „view“ dokumenta, pojednostavit će izbjegavanje nepotrebnog tereta dodanog koda. Pomažu kod:

- Filtriranja dokumenata u bazi podataka
- Izvlačenja podataka iz više dokumenata i prezentiranja istih u određenom redoslijedu
- Kreiranje indeksa za lakši pronalazak dokumenata
- Smanjivanje „front-loada“ kalkulacija s pohranom gotovih među rješenja ili pomoćnih izračuna vezanih za dokumente.

Primjer pogleda koji omogućava filtriranje podataka nad nekoliko oglednih dokumenata koji sadrže lokacije i datume budućih instalacija bankomata ili POS (eng. point of sale) uređaja.:

{"_id":"Palmižana",
"_rev":"00000000001",
"ID":"APAR003",
"body":"instalacija na otoku, kafić Makarena",
"datum":"2015/04/30 09:00:00"}
...
{"_id":"Stradun",
"_rev":"00000000004",

"id":"APAR007",

"body":"dućan Kod Mladoga Duje"}

...

{"_id":"Zagreb1",

"_rev":"00000000002",

"id":"APAR009",

"body":"šetnica gornji grad, mjenjačnica",

"datum":"2015/04/25 09:30:00"}

Kako bi dobili listu instalacija koje slijede iz gore navedenih dokumenata, potrebno je kreirati funkciju mapiranja. U ovom primjeru koristi se JavaScript, koji je ujedno i preporučeni jezik zbog istovjetnosti sintakse s JSON dokumentima na koje se CouchDB oslanja.

```
function(doc) {
  if(doc.datum && doc.id) {
 emit(doc.datum, doc.id); }}
```

Rezultat ovakvog pogleda je:

Key	**Value**
“ *2015/04/25 09:30:00*”	*APAR009*
“ *2015/04/30 09:00:00*”	*APAR003*

Tablica 4. Rezultat funkcije mapiranja

Kod upita na pogled, CouchDB uzima kod pogleda i njime prolazi kroz svaki dokument u bazi podataka nad kojim je pogled definiran. Upit na pogled daje rezultat pogleda.

Sve map funkcije imaju jedan *doc* parametar koji se odnosi na pojedini dokument unutar baze podataka. U gornjem primjeru funkcija je provjeravala ima li dokument vrijednosti *datum* i *id* (u primjeru dva od tri dokumenta zadovoljavaju ovaj uvjet). Nakon toga zove se emit funkcija. Radi se o funkciji koja uvijek uzima dva argumenta, prvi kao ključ a drugi kao vrijednost, te u konačnici kreira unos u rezultatu pogleda. CouchDB uzima sve što se prosljeđuje u emit() funkciju te to slaže u listu. Svaki red liste uključuje ključ i vrijednost, te je sortirana po ključu, što nam daje dodatnu prednost kod naknadnog pretraživanja.

Kako bi se spriječilo veliko zauzeće resursa, CouchDB prilikom upita na pogled

prolazi kroz sve dokumente isključivo pri prvom pokretanju. Naknadno se provjeravaju samo ažurirani dokumenti. Rezultat pogleda sprema se u binarno stablo, koje se sprema odvojeno za poglede i za dokumente.

4.3. Funkcije Validacije

Kao prvu razinu zaštite od neispravnih ili neautoriziranih ažuriranja dokumenta, CouchDB koristi *validate_doc_update* funkciju. Ovakve funkcije, kao i map i reduce funkcije vrše se neovisno o trenutnom zahtjevu korisnika. Njima se može blokirati svako nastajanje/izmjena dokumenta, bilo da dolazi od krajnjeg korisnika ili kao replikacija iz drugog CouchDB čvora.

CouchDB šalje funkcije i dokumente u JavaScript interpreter – što nam omogućava korištenje funkcija pisanih u JavaScriptu za validaciju. Validate_doc_update funkcija se izvršava prilikom svakog stvaranja ili ažuriranja svakog dokumenta. Ukoliko ova funkcija vraća exception, ažur se odbija, a ukoliko se funkcija izvrši do kraja, ažuri se prihvaćaju.

Kao u mnogim sustavima, korištenje mehanizama validacije kao što su ove funkcije je opcionalno. One su tu u svrhu poboljšanja kvalitete korištenja, jer izbacivanjem prethodno očekivanih grešaka kod kreacije ili ažura dokumenata izbjegavamo one neočekivane u daljnjem radu koje nastaju kao posljedica krivog formata sadržaja ili strukture dokumenata. U slučaju postojanja više dokumenata dizajna s raznolikim validacijskim funkcijama, sve one se prolaze kod svake kreacije dokumenata, te se upis ili ažur u takvim situacijama dozvoljava isključivo ako sve funkcije budu zadovoljene.

Pisanje same validacijske funkcije zahtjeva sljedeće argumente:

```
function(noviDokument, stariDokument, korisnik) {}
```

Funkcija uzima 3 argumenta, novu i postojeću verziju dokumenta i objekt koji odgovara korisniku koji je napravio zahtjev. Funkcijska logika se jednostavno dodaje unutar tijela funkcije – a najčešće korištene metode prilikom istoga su „warning“ i „forbidden“.

Primjer validacijske funkcije koja ne dozvoljava nikakve radnje:

```
function(noviDokument, stariDokument, korisnik) {
throw({forbidden : 'nije dozvoljeno });}
```

Obzirom da CouchDB dozvoljava samo jednu validacijsku funkciju po dokumentu dizajna, najčešće se koristi validacija raznolikih tipova dokumenata i podataka grananjem funkcijske logike unutar iste funkcije.

Najosnovnija provjera (validacija) u korištenju je provjera dali se tražena polja nalaze u dokumentu. Ovime se olakšava kasnije pisanje pogleda, budući se ne mora provjeravati sve vrijednosti prije nego se dokumenti krenu koristiti.

4.4. Rješavanje konflikta

Do konflikta dolazi kad ažuri s dva ili više izvora trebaju izmijeniti određeni dokument. Radi se o prirodnoj pojavi u distribuiranim sustavima. Ovdje će biti okvirno prikazani neki od načina na koje CouchDB rješava takve situacije.

Kod replikacije dvije baze podataka koje imaju različita ažuriranja, CouchDB dokumentu dodaje poseban atribut „_conflicts“ te ga postavlja na vrijednost true. Nakon toga CouchDB odlučuje koja od verzija će biti pohranjena kao posljednja revizija. U ovakvim situacijama spremaju se obje (ili više) ažuriranja, međutim verzije koje ne budu odabrane spremaju se kao stare, odnosno prethodne verzije.

Bitno je istaknuti da CouchDB ne pokušava spojiti sve izmjene koje su rađene u svakoj varijanti dokumenata – to radi logika na aplikacijskoj razini, odnosno programer/kreator. Obzirom da se u dokumentima često radi o jednostavnim vrijednostima bez mogućnosti određivanja njihove važnosti i točnosti, nije lako odlučiti koja je ispravna bez poznavanja dubljeg konteksta zaprimanja informacije. Replikacija garantira da će svaki nezavisni čvor CouchDB-a detektirati konflikt i na temelju deterministčkog algoritma donijeti istu odluku o odabiru konačnog dokumenta za najažurniju verziju.

U konačnici, korisniku ili aplikacijskoj logici preostaje da između sačuvanih verzija odabere onu pravu, što nakon replikacije te odluke, dovodi do konzistentnosti u svim čvorovima/bazama podataka.

Primjer koraka rješavanja konflikta ažuriranjem u dva odvojena čvora bez stalne sinkronizacije:

1. Kreira se dokument u čvoru A
2. Pokretanjem replikacije A->B dokument (posljednja revizija) se kopira u čvor B
3. Ažuriranjem dokumenta u čvoru B on dobiva novu reviziju u čvoru B
4. Ažuriranjem dokumenta u čvoru A nastaje još jedna revizija u čvoru A
5. Ponovnim pokretanjem replikacije A->B CouchDB detektira postojanje dvije revizije te nastaje konflikt – s postojanjem dvije „posljednje verzije“
6. Odlukom o ispravnoj posljednjoj verziji dobiva se „pobjednička“ revizija dokumenta.

Konkretizacija koda za navedeni primjer obavljena u CouchDB-u izgleda ovako:
U navedenim primjerima koda prilikom oznaka revizija koristit će se „md5“ što skraćeno predstavlja md5-hash umjesto niza konkretnih znakova.

Prvo je potrebno kreirati dvije baze podataka (u ovom primjeru obje se nalaze na istoj instanci CouchDB-a)

```
HOST="http://127.0.0.1:5984"
> curl -X PUT $HOST/bazaA
{"ok":true}
> curl -X PUT $HOST/bazaB
{"ok":true}
```

Nakon toga kreira se novi dokument u bazi podataka A te pokrene replikacija na bazu podataka B:

```
> curl -X PUT $HOST/bazaA/primjer -d '{"brojac":1}'
{"ok":true,"id":"primjer","rev":"1-md501"}
> curl -X POST $HOST/_replicate -d
'{"source":"bazaA","target":"http://127.0.0.1:5984/bazaB'
{"ok":true,...,"docs_written":1,"doc_write_failures":0}]} -H "Content-Type:
application/json"
```

Ovdje se mogu napraviti ažuri dokumenta na obje baze podataka. Primjerice ažur vrijednosti brojač u bazi podataka B na 2 i bazi podataka A na 3 nakon čega se ponovi replikacija, s očekivanim konfliktom.

> curl -X PUT $HOST/bazaB/primjer -d '{"brojac":2,"_rev":"1-md5"}'

{"ok":true,"id":"primjer","rev":"2- md502"}

> curl -X PUT $HOST/bazaA/primjer -d '{"brojac":3,"_rev":"1- md5"}'

{"ok":true,"id":"primjer","rev":"2- md503"}

> curl -X POST $HOST/_replicate -d

'{"source":"bazaA","target":"http://127.0.0.1:5984/bazaB"}'

{"ok":true,..."docs_written":1,"doc_write_failures":0}]} -H "Content-Type:

application/json"

Za provjeru postojanja konflikta potrebno je pokrenuti jednostavan pogled koji pretražuje iste:

function(doc) {

if(doc._conflicts) {

emit(doc._conflicts, null); }}

Što za rezultat daje:

{"total_rows":1,"offset":0,"rows":[

{"id":"primjer","key":["2-md503"],"value":null}]}

Iz čega je vidljivo da je jedan od dva ažura u konfliktu, a konkretno se radi o ažuru s baze podataka A. U slučaju da se slažemo s odabirom CouchDB-a ovdje bi proces završio potvrdom, sa zadržanim dokumentom koji sadrži vrijednost brojača 2 kao finalnim.

> curl -X GET $HOST/bazaB/primjer

{"_id":"primjer","_rev":"2-md502","brojac":2}

Ukoliko se ne slažemo s navedenim odabirom, potrebno je propagirati izmjene te obrisati verziju koja nam ne odgovara:

> curl -X DELETE $HOST/bazaB/primjer?rev=2-md502

{"ok":true,"id":"foo","rev":"3-md503"}

> curl -X PUT $HOST/bazaB/primjer -d '{"count":3,"_rev":"2-md503"}'

{"ok":true,"id":"primjer","rev":"3-md504"}

Nakon ovoga u CouchDB-u je stvorena još jedna verziju dokumenta prema našim željama, koja mijenja konfliktnu. U konačnici dokument zaprima vrijednosti koje odgovaraju uvjetima:

> curl -X GET $HOST/bazaB/primjer

{"_id":"primjer","_rev":"3-3-md504","count":3}

I jedino što preostaje je replicirati željeni rezultat natrag u bazu podataka A:

> curl -X POST $HOST/_replicate -d

'{"target":"bazaA","source":"http://127.0.0.1:5984/bazaB"}' -H "Content-Type: application/json"

Nakon čega i baza podataka A ima najažurniju varijantu dokumenta

> curl -X GET $HOST/bazaA/primjer

{"_id":"primjer","_rev":"3-3-md504","count":3}

Ovim primjerom prikazano je kako rješavanje konflikta, iako jednostavno, ipak u konačnici ovisi o samom autoru aplikacije ili programske logike jer o dobroj definiranosti iste ovisi hoće li rješavanje dati željene rezultate.

5. Aplikacija

Kao primjer aplikacije odabran je unos i upiti nad dokumentima koji se koji se trenutno odvija između dva odjela u tvrtki iz financijskog sektora koja se između ostaloga bavi postavljanjem bankomata na odabrane lokacije. Primjer je pojednostavljen iz razloga što točan proces ne smije biti prikazan obzirom na osjetljivost podataka.

U pojednostavljenom primjeru posao vezan uz instalaciju bankomata ili sličnih uređaja najčešće se odvija suradnjom 2 odjela, odjela prodaje koji pronalazi i ugovara lokacije te odjela operative čija je zadaća na ugovorenim lokacijama postaviti, upogoniti i održavati uređaj. To se najčešće radi uz pomoć vanjskih suradnika koji dobivaju naloge od operative.

Budući da se sales odjel sastoji od samostalnih komercijalista koji podatke o novim lokacijama unose u tablice/baze podataka neovisno jedni o drugima, zna se dogoditi da tablice budu zaključane od strane jednog korisnika te za to vrijeme drugi djelatnici nemaju mogućnost unosa. Isto tako, kako bi se podatci unijeli u tablicu potrebno je da djelatnik koji ih unosi bude u tom trenutku povezan na internet kako bi uopće mogao pristupiti tablicama (Excel datoteke).

Obzirom da CouchDB kao svoje prednosti navodi mogućnost lake sinkronizacije podataka i van mrežni rad, bit će napravljen ogledni primjer male aplikacije koja bi omogućila takav unos podataka. Na strani korisnika bit će vidljiva HTML datoteka koja će uz podršku JavaScripta moći odraditi navedene radnje klikom miša.

Prilikom prikaza aplikacije koristit će se dvije baze podataka na istoj instanci CouchDB-a, ali sam način replikacije i upravljanja ostaje isti i u slučaju kad bi se koristila dva udaljena čvora. Prva baza podataka s nazivom „operativa“ po svojoj prilici odgovara trenutnim excel datotekama kojima svi djelatnici pristupaju dok su u uredu. Baza podataka sales predstavlja bazu podataka koju bi zaposlenici salesa mogli imati na vlastitim prijenosnim računalima, te na istoj mogli raditi van mrežno, uz sinkronizaciju s bazom podataka operations prema potrebi. Bit će omogućen unos poznatih podataka o selektiranoj lokaciji, te dostupan gumb za replikaciju. Baza podataka operations će sadržavati sve poznate podatke o lokacijama dobivene od sales time, te omogućiti operativnom timu da iste mijenja, dodaje, grupira i pregledava.

Kao osnova aplikacije koristi se CouchApp, alat za razvoj couchDB aplikacija, te dostupnom predlošku za izradu jednostavne aplikacije. Kao i mnoštvo drugih potrebnih alata, ovaj sustav ne dolazi upakiran u CouchDB instalaciju, nego se radi o odvojenom projektu, koji isto tako ima specifične zahtjeve prilikom instalacije. Jedna od opcija instalacije

uključuje potrebu za prethodno instaliranim Pythonon 2.7. ili niže.

Aplikaciju čine 2 HTML stranice, po jedna za Sales i jedna za Operations tim. U slučaju potrebe za dijeljenjem aplikacije na više korisnika, svaki od predstavnika sales tima bi mogao dobiti primjerak sales HTML stranice s pripadajućim kodom te to koristiti kod unosa podataka. Radi jednostavnosti oglednog primjera, na stranice sales i operations dodani su linkovi koji ih međusobno vežu, kako bi se jednostavnije prikazao rad u obje.

HTML stranice u oba primjera su vrlo jednostavne, obzirom da se većina logike nalazi u odvojenim JavaScript datotekama:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Operations Page</title>
    <link rel="stylesheet" href="style/main.css" type="text/css">
    <script src="vendor/couchapp/loader.js"></script>
    <script src="recordedit.js"></script>

  </head>
  <body>
    <div id="account"></div>

    <h1>Opeartions</h1>
    <div data-role="head">
      <select id="showcase">
        <option>Date</option>
        <option>location</option>
        <option>ATMID</option>
      </select>
    </div>
    <a href="http://127.0.0.1:5984/sales/_design/sales/index.html#" class="operationsView">Visit Operations</a>

    <div id="items"><div id="add"><a href="#" class="add">Add Contact</a></div>
      <div id="contacts"></div>
      <div id="contactform"></div>

  </body>
</html>
```

Slika 8. – HTML stranice Operations

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sales Page</title>
    <link rel="stylesheet" href="style/main.css" type="text/css">
    <script src="vendor/couchapp/loader.js"></script>
    <script src="recordedit.js"></script>

  </head>
  <body>
    <div id="account"></div>

    <h1>Sales</h1>
    <a href="http://localhost:5984/operations/_design/operations/index.html" class="operationsView">Visit Operations</a>

    <div id="items"><div id="add"><a href="#" class="add">Add Sales</a></div>
      <div id="contacts"></div>
      <div id="contactform"></div>

  </body>
</html>
```

Slika 9. – HTML stranice Sales

Java script datoteke (s naglaskom na recordedit.js) sadrže većinu potrebne logike za rad s dokumentima u bazi podataka. Kod korištenja bilo koje HTML datoteke poziva se pripadajuća JavaScript skripta. Na početku svake skripte nalaze se podatci za spajanje na samu bazu podataka:

index.html recordedit.js

```
// ucitaj bazu
db = $.couch.db("sales");
operationDb = $.couch.db("operations");

// pokazi informacije o bazi
function updatecontacts() {
    $("#contacts").empty();

    db.view("sales/byname", {
        success: function(data) {
            for (i in data.rows) {
                $("#contacts").append('<div id="' + data.rows[i].value._id + '" class="contactrow"><span>' +
                                      data.rows[i].value.ATMID +
                                      "</span><span>" +
                                      data.rows[i].value.location +
                                      "</span><span>" +
                                      data.rows[i].value.date_of_installation +
                                      "</span><span>" +
                                      '<a href="#" id="' + data.rows[i].value._id + '" class="edit">Edit Contact</a>' +
                                      "</span><span>" +
                                      '<a href="#" id="' + data.rows[i].value._id + '" class="replicateOne">Replicate Contact</a>' +
                                      "</span><span>" +
                                      '<a href="#" id="' + data.rows[i].value._id + '" class="remove">Remove Contact</a>' +
                                      "</span></div>"
                                     );
            }
        }
    });
}
```

Slika 11. – JavaScript stranice

Pokretanje aplikacije temeljene na CouchApp-u zahtjeva par dodatnih koraka. Pod uvjetom da su sve predispozicije za funkcioniranje instalirane (CouchDB, CouchApp), potrebno je gotovu aplikaciju „umetnuti“ u CouchDB što se čini sljedećim komandama:

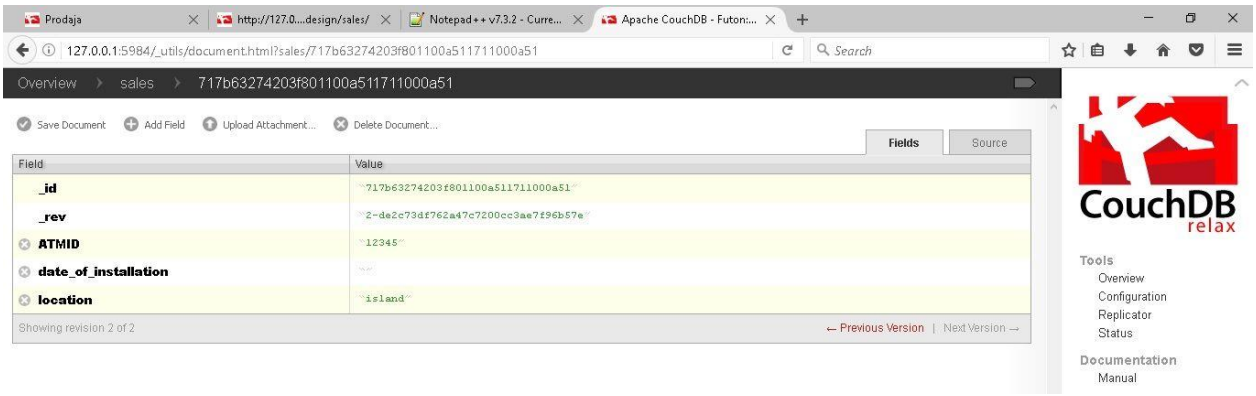
```
c:\Users\Domagoj\Desktop\operations\operations>couchapp push _design/sales http://localhost:5984/sales
2017-02-18 17:45:54 [INFO] Visit your CouchApp here:
http://localhost:5984/sales/_design/sales/index.html
```

Slika 12. – Umetanje CouchApp aplikacije Sales

U konačnici dobijemo stranicu Sales dostupnu unutar CouchDB-a:

Slika 13. – CouchApp aplikacija Sales

Stranica omogućava unos, brisanje i replikaciju podataka o bankomatima. Po dodavanju bankomata, isto je moguće provjeriti unutar same baze podataka, što je vidljivo na sljedećoj slici.



Slika 14. – Podatci u Futon sučelju

Stranica Operations je po svojoj strukturi skoro istovjetna sales stranici. Razlikuje se u tome što se operacije događaju na pripadajućoj Operations bazi podataka unutar CouchDB-a (izuzev replikacije koja podatke smješta u Sales bazu podataka).

Slika 15. – CouchApp aplikacija Operations

Pomoću navedenih aplikacija moguće je osnovno upravljanje podatcima o bankomatima unutar CouchDB baze podataka. Dodavanje novih mogućnosti se ne razlikuje puno od programiranja istih u bilo kojem jeziku ili bazi podataka. Za daljnje poboljšanje aplikacije bilo bi potrebno uljepšati izgled kroz CSS datoteku.

6. Zaključak

U ovom radu proučen je CouchDB, jedan od NoSQL alata/mehanizama za upravljanje bazama podataka. Pokušalo se odgovoriti na pitanje koliko jedna takva baza podataka/sustav može pomoći u svakidašnjem radu u slučaju jednostavnih dokumenata koji se ažuriraju od strane više djelatnika tvrtke, često i u isto vrijeme. Iako se autori CouchDB-a trude na više mjesta oglasiti CouchDB kao iznimno jednostavnim za rad i učenje, iskustvo iz istraživanja vezana za ovaj rad govore da su iznimna ograničenost u izvorima informacija, te problemi s radom raznih verzija baza podataka na različitim operativnim sustavima u suprotnosti s navedenom mantrom. CouchDB može poslužiti kao nadogradnja nekome tko je dobro upoznat s radom na različitim bazama podataka, ali ne predstavlja lak zadatak ukoliko bi bio proučavan samostalno kao neka vrst uvoda u baze podataka. Smatram da ovdje ima dosta prostora za napredak, npr. pojednostavljanjem grafičkog sučelja kako bi moglo konkurirati komandno-linijskom radu. Ovaj rad je rađen na verziji 1.6.1 CouchDB-a, a u međuvremenu je izašla i verzija 2.0 koja donosi napredak u vizualnom smislu, prelaskom s Futon na Fauxton sučelje koje ipak sadrži nešto više mogućnosti te je vidljiv pomak u tom području.

Što se tiče korištenja manje CouchDB aplikacije kao zamjene za postojeće alate u tvrtki, ispostavilo se da prednosti CouchDB-a nisu dovoljne kako bi pokrile nedostatke podrške i potrebu za učenjem novoga pristupa. Obzirom da se radi o redu veličine do desetak korisnika sustava, gubitak vremena za vrijeme „zaključanosti“ dokumenata ipak nije dovoljno velik kako bi njegovo izbivanje donijelo dovoljne uštede vremena. Kod distribuiranog rada olakšava se unos podataka dislociranim djelatnicima, ali u konačnici dovodi do potrebe za revizijom jer korisnici koji imaju konačnu odluku prilikom rješavanja konfliktnih dokumenata su najčešće korisnici s najmanjim dodirnim točkama sa samom bazom podataka, te preferiraju rad na ranije naučenim tehnologijama (MS Excel, Sharepoint).

U konačnici radi se o zanimljivom pristupu radu s bazama podataka, koji može poslužiti kao nadgradnja postojećim sustavima, ali je potrebna promjena načina razmišljanja kako bi se iskoristile prednosti iste. Ovakva promjena razmišljanja možda će biti jednostavnija nekim novim generacijama koje će biti upoznate s ovakvim sustavima od početka, za razliku od autora ovog rada koji godinama razmišlja o bazama podataka isključivo kao o skupovima tablica u nekom od sustava relacijskih baza podataka.

Literatura:

[1] Anderson C, Lehnardt J, Slater N (2010) *CouchDB The definite Guide* O'Reilly Media.

[2] Brown MC (2012) *Getting started with CouchDB* O'Reilly Media.

[3] Brown MC (2014), Building CouchApps, dostupno 12.1.2017. na: http://www.ibm.com/developerworks/opensource/tutorials/os-couchapp

[4] couchdb.org Docs, dostupno 12.1.2017. na: http://docs.couchdb.org/en/1.6.1/

[5] De Jong J (2011), A simple task list application in couchdb, dostupno 12.1.2017. na: http://www.speqmath.com/tutorials/couchdb_tasklist/index.htmlSd

[6] Girish Kumar (2014) Exploring the different types of NoSQL databases, dostupno 12.1.2017. na: https://www.3pillarglobal.com/insights/exploring-the-different-types-of-nosql-databases

[7] Gosling J (1997). The 8 fallacies of distributed computing, dostupno 10.11.2016. na https://www.cse.unsw.edu.au/~cs9243/16s1/papers/fallacies.pdf

[8] mondgodb.com, NoSQL explained, dostupno 12.1.2017. na: https://www.mongodb.com/nosql-explained

[9] nosql-database.org, dostupno 12.1.2017. na: http://nosql-database.org

[10] Rabuzin K (2015) *SQL – Napredne Teme*. Čakovec: Zrinski d.d.

[11] Richards J (2015), Advantages and disadvantages of NoSQL databases, dostupno 12.1.2017. na: http://www.hadoop360.com/blog/advantages-and-disadvantages-of-nosql-databases-what-you-should-k

[12] Simon Abram (2014) Cap-Theorem, dostupno 12.1.2017. na http://www.abramsimon.com/cap-theorem/

[13] Schatten, M. i Ivković, K. (2012). Regular Path Expression for Querying Semistructured Data - Implementation in Prolog. Fakultet Organizacije i Informatike u Varaždinu, Central European Conference on Information and Intelligent Systems. Varaždin, Hrvatska, 19.-21. Rujna 2012. godine, Varaždin: Fakultet Organizacije i Informatike u Varaždinu.

Prilog 1. – programski kod aplikacije

Programski kod datoteke index.html:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sales</title>
    <link rel="stylesheet" href="style/main.css" type="text/css">
    <script src="vendor/couchapp/loader.js"></script>
    <script src="recordedit.js"></script>
  </head>
  <body>
    <div id="account"></div>
    <h1>Sales</h1>
    <a href="http://localhost:5984/operations/_design/operations/index.html"
class="operationsView">Posjeti Operativu</a>
    <div id="items"><div id="add"><a href="#" class="add">Dodaj bankomat</a></div>
      <div id="contacts"></div>
      <div id="contactform"></div>
  </body>
</html>
```

Programski kod datoteke recordedit.js:

```
// ucitaj bazu
db = $.couch.db("sales");
operationDb = $.couch.db("operations");
// pokazi informacije o bazi
function azuriraj() {
    $("#contacts").empty();
```

```
    db.view("sales/byname", {
        success: function(data) {
            for (i in data.rows) {
                $("#contacts").append('<div id="' + data.rows[i].value._id + '"
class="contactrow"><span>' +
                                data.rows[i].value.ATMID +
                                "</span><span>" +
                                data.rows[i].value.location +
                                "</span><span>" +
                                data.rows[i].value.date_of_installation +
                                "</span><span>" +
                                '<a href="#" id="' + data.rows[i].value._id + '"
class="edit">Izmjeni</a>' +
                                "</span><span>" +
                                '<a href="#" id="' + data.rows[i].value._id + '"
class="replicateOne">Repliciraj </a>' +
                                "</span><span>" +
                                '<a href="#" id="' + data.rows[i].value._id + '"
class="remove">Izbrisi</a>' +
                                "</span></div>"
                               ); }}});}

function formazaunos(doctoedit) {
    var formhtml;
    formhtml =
        '<form name="update" id="update" action="">';

    if (doctoedit) {
        formhtml = formhtml +
            '<input name="docid" id="docid" type="hidden" value="' + doctoedit._id + '"/>';
    }
    formhtml = formhtml +
        '<table>';
```

```
    formhtml = formhtml +
        '<tr><td>ATMID</td>' +
        '<td><input name="ATMID" type="text" id="ATMID" value="' +
        (doctoedit ? doctoedit.ATMID : '') +
        '"/></td></tr>';
    formhtml = formhtml +
        '<tr><td>location</td>' +
        '<td><input name="location" type="text" id="location" value="' +
        (doctoedit ? doctoedit.location : '') +
        '"/></td></tr>';
    formhtml = formhtml + '<tr><td>Date of installation</td>' +
        '<td><input name="date_of_installation" type="text" id="date_of_installation" value="'
+
        (doctoedit ? doctoedit.date_of_installation : '') +
        '"/></td></tr>';

    formhtml = formhtml +
        '</table>' +
        '<input type="submit" name="submit" class="update" value="' +
        (doctoedit ? 'Update' : 'Add') + '"/>' +
        '</form>';
    $("#contactform").empty();
    $("#contactform").append(formhtml);
}

function kreirajdokument(doc,form) {
    if (!doc) {
        doc = new Object;
    }
    // console.log("**$: ", form.find("input#ATMID"));
    doc.ATMID                = form.find("input#ATMID").val();
    doc.location             = form.find("input#location").val();
    doc.date_of_installation    = form.find("input#date_of_installation").val();
```

```
    return(doc);
}

$(document).ready(function() {
    azuriraj();
    $("#contacts").click(function(event) {

    var target = $(event.target);
    if (target.is('a')) {
        id = target.attr("id");

        if (target.hasClass("edit")) {
            db.openDoc(id, { success: function(doc) {
                formazaunos(doc);
            }});}

        if (target.hasClass("remove")) {
            html = '<span class="confirm">Sigurno obrisati? ' +
                '<a href="#" id="' + id + '" class="actuallydel">Da</a>' +
                '<a href="#" id="' + id + '" class="canceldel">Ne</a></span>';
            target.parent().append(html);
        }

        if (target.hasClass("actuallydel")) {

            db.openDoc(id, {
                success: function(doc) {
                    db.removeDoc(doc, { success: function() {
                    target.parents("div.contactrow").remove();
                    }});}});   }

        if (target.hasClass("canceldel"))  {
            target.parents("span.confirm").remove();
        }
```

```
    if (target.hasClass("replicateOne")) {

        var doc                  = new Object();
        var form                 = target.parents("div.contactrow").children("span");

        doc.ATMID                = form[0].textContent;
        doc.location             = form[1].textContent;
        doc.date_of_installation = form[2].textContent;

        operationDb.saveDoc(doc, {
            success: function() {
              alert("success");
            },
            error: function(err) {
               alert(err);
            }});}}});

$("a.add").live('click', function(event) {
    formazaunos();
});

$("input.update").live('click', function(event) {
    var form = $(event.target).parents("form#update");

    var id = form.find("input#docid").val();
    if (id) {
        db.openDoc(id, {
            success: function(doc) {
                db.saveDoc(kreirajdokument(doc,form), {
                    success: function() {
                        $("form#update").remove();
                        azuriraj();
                    }); }, });  }
```

```
    else
    {
        db.saveDoc(kreirajdokument(null,form), {
            success: function() {
                $("form#update").remove();
                azuriraj();
            },});}
    return false;
});});
```