UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Janez Štupar

Podatkovni strežnik Apache CouchDB in CouchApp aplikacije

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00046/2010
Datum: 05.11.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **JANEZ ŠTUPAR**

Naslov: **PODATKOVNI STREŽNIK APACHE COUCHDB IN COUCHAPP APLIKACIJE**

APACHE COUCHDB DATABASE SERVER AND COUCHAPP APPLICATIONS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Proučite podatkovno bazo in strežnik Apache CouchDB. Opišite arhitekturo strežnika ter predstavite osnovne gradnike podatkovne baze in njihovo namembnost. Na primeru aplikacije za ustvarjanje in objavo spletnih dnevnikov prikažite delovanje samopostrežnih aplikacij, nato pa realizirajte samopostrežno aplikacijo za zbiranje, hrambo in analizo dnevniških zapisov v formatu Syslog.

Mentor:

prof. dr. Viljan Mahnič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Janez Štupar,

z vpisno številko 63020301,

sem avtor diplomskega dela z naslovom:

Podatkovni strežnik CouchDB in CouchApp aplikacije

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  prof. dr. Viljan Mahnič
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 18.03.2011 Podpis avtorja:

Zahvala

Rad bi se zahvalil vsem, ki verjamete vame. Vaša opora mi pomaga v težkih trenutkih.

Še večja zahvala pa gre tistim, ki vame dvomite. Vaša pozornost me osredotoča na zastavljene cilje.

V bolečini se rojevajo najlepši biseri.

materi Kristini

Kazalo

Seznam uporabljenih kratic in simbolov

ACID Atomicity, Consistency, Isolation, Durability - nedeljivost, konsistenca, izolacija in trpežnost

AJAX Asynchronous JavaScript And XML - stil pisanja spletnih aplikacij z asinhronimi zahtevki

API Application Programming Interface - vmesnik za programiranje aplikacij

CouchApps CouchDB self hosted applications - aplikacije strežene neposredno iz strežnika Apache CouchDB

CSS Cascading Style Sheets - prekrivni slogi

DOM Document Object Model - objektni model dokumenta

ECMA European Computer Manufacturers Association - združenje evropskih proizvajalcev računalnikov

Erlang OTP Erlang Open Telecom Platform - odprta telekomunikacijska platforma Erlang

HTML Hyper Text Markup Language - označevalni jezik za nadbesedila

HTTP Hyper Text Transfer Protocol - protokol za prenos nadbesedil

JSON JavaScript Object Notation - zapis objektov v JavaScript formatu

NIST National Institute of Standards and Technology - nacionalni inštitut za tehnologijo in standarde

RFC Request For Comment - prošnja za pripombe/komentarje

REST Representational State Transfer - predstavitveni prenos stanja

SQL Structured Query Language - jezik za pisanje poizvedb

UML Universal Markup Lanugage - univerzalni označevalni jezik

URI Unified Resource Identifier - enotni označevalnik vira

URL Unified Resource Locator - enolični krajevnik vira

UUID Universaly Unique Identifier - univerzalno unikatni identifikator

WWW World Wide Web - svetovni splet

Povzetek

V diplomskem delu je predstavljena podatkovna baza in strežnik Apache CouchDB, njegova arhitektura, izvedbene značilnosti in primer izvedbe samostrežene spletne aplikacije.

Podatkovna baza Apache CouchDB spada v družino dokumentno orientiranih podatkovnih baz, pri čemer struktura podatkov ni omejena s shemo, poleg tega pa za dostop in upravljanje s podatki ne uporablja jezika SQL. Avtorji so želeli ponuditi podatkovno bazo, katere prvenstveni cilji so zanesljivost hrambe podatkov oz. odpornost na napake (ang. Fault Tolerance), vgrajeni mehanizmi za replikacijo podatkov ter na spletnih standardih temelječa arhitektura. Bazo tovrstnega tipa velja uporabiti v scenarijih, kot so: tradicionalne spletne aplikacije, aplikacije za hrambo metapodatkov, aplikacije za zajem podatkov, hitra izdelava prototipov in porazdeljene aplikacije, pri katerih so nekatera vozlišča občasno nedosegljiva. Diplomsko delo vsebuje popis gradnikov: strežnika podatkovne baze, podatkovne baze, njihovo vlogo in uporabne lastnosti. Omenjene so različne vrste aplikacij, ki za hrambo podatkov uporabljajo CouchDB. Poleg tega na primeru aplikacije za ustvarjanje in objavo spletnih dnevnikov (ang. web log, skrajšano blog) natančno preučimo delovanje posebne kategorije samostreženih (ang. self hosted) aplikacij in se na ta način seznanimo s pregledovanjem, dodajanjem, spreminjanjem ter varovanjem podatkov v samostreženih aplikacijah.

Za praktični del diplomskega dela je izdelana samostrežena aplikacija za zbiranje, hrambo in analizo dnevniških zapisov v formatu Syslog.

Ključne besede:

Apache CouchDB, podatkovne baze, CouchApps, spletne aplikacije, BoxSpring, logiranje, syslog

Abstract

The thesis subject is presentation of Apache CouchDB database and server, it's architecture and implementation details. The thesis also contains an example of CouchDB based self hosted web application.

Apache CouchDB database is a member of schema less, document oriented database family. CouchDB doesn't use SQL language for database operation and querying. It's authors' primary goal was to create a system that will exhibit characteristics of fault tolerance, built-in replication mechanism and architecture built around web standards. Typical usage scenarios for database of this type are web applications, meta-data storage, data collection applications, fast application prototyping and distributed applications with sporadic absence of connection between nodes. This thesis contains overview of database server and database functions, it also discusses their role and usage in the system. Thesis also approaches the topic of possible architectural styles for systems that incorporate CouchDB server. To illustrate the special self-hosted application paradigm, an example of weblog web application implementation is thoroughly analysed and presented.

We also developed a self hosted web application for collection, storage and analysis of Syslog formatted records.

Key words:

Apache CouchDB, database, CouchApps, web applications, BoxSpring, logging, syslog

Poglavje 1

Uvod

Danes se pri vsakodnevnih opravilih neprestano srečujemo s podatkovnimi bazami. Podatkovne baze različnih tipov informacijskim sistemom dajejo zmožnost pomnjenja, hranjenja in obdelave najrazličnejših vrst podatkov. Brez podatkovnih baz si ne bi bilo možno zamisliti najrazličnejših storitev, katerih uporabo si sodobni ljudje želimo ter na katere se vsakodnevno zanašamo. Od njih pričakujemo, da bodo hitre, zanesljive, poceni ter predvsem vedno razpoložljive. Brez podatkovnih baz bi se morali še vedno zanašati na papirnate arhive, ki so neprimerno počasnejši in pri katerih nam povsem fizikalne omejitve preprečujejo zbiranje, hranjenje in obdelavo velikih količin podatkov.

V zadnjih tridesetih letih na tržišču močno prevladujejo sistemi za upravljanje podatkovnih baz, temelječi na relacijski teoriji E.F. Codd-a [3]. Sicer so že pred tem obstajale podatkovne baze, a so bile z vidika sodobnega uporabnika močno omejene v svojih zmogljivostih. Omejitve podatkovnih baz so močno odvisne od tehnologij, na katerih tečejo sistemi za upravljanje podatkovnih baz. Tako so bile funkcije navigacijskih podatkovnih baz [10] močno odvisne od zmogljivosti in lastnosti medijev, npr. magnetnih trkov in bobnov, na katerih so se shranjevali podatki. S pojavitvijo trdih diskov na trgu so se pojavile tudi težnje po drugačnih podatkovnih bazah, ki bi odpravile največje pomanjkljivosti navigacijskih baz, to sta predvsem nezmožnost naključnih dostopov in odsotnost iskalnih funkcij. Koncept relacijske podatkovne baze temelji ravno na teh tehnoloških prednostih trdega diska kot medija za shranjevanje podatkov. Poleg omenjenih so se pojavljale tudi drugačne podatkovne baze, a je bila njihova uporaba omejena na specializirane niše [35]. Relacijske podatkovne baze se uporabljajo v specializiranih strežnih sistemih kot del centraliziranih aplikacij za obdelavo in hrambo podatkov. Ker je bila strežniška strojna oprema v preteklosti izjemno draga, in ker so stranke pričakovale vi-

soko zanesljivost podatkovnih baz, sta bili področji hitrosti in zanesljivosti za proizvajalce programske opreme za upravljanje podatkovnih baz bistvenega pomena. Povečanje strežnih kapacitet dosegamo tako, da strežnik za hrambo podatkov nadgradimo z močnejšim in seveda dražjim. Tipičen predstavnik aplikacijske arhitekture tega obdobja je arhitektura strežnik-odjemalec (ang. client-server), pri kateri uporabniki dostopajo do centralnega strežnika, na katerem so shranjeni podatki, te prenašajo k sebi, jih obdelujejo in zapisujejo nazaj na strežnik.

V zadnjih petnajstih letih pa je prišlo do velikega razmaha svetovnega spleta, pri čemer so se ponudniki spletnih storitev začeli srečevati z novimi tehnološkimi izzivi, ki izhajajo iz sledečih okoliščin:

- Storitve so postale globalne.
- Količina podatkov, ki jih je potrebno pomniti in obdelovati, se je zelo povečala.
- Do storitev dostopajo velike količine odjemalcev.
- Nad podatki pogosto želimo vršiti zapletene obdelave, velikokrat v mehkem realnem času.

V iskanju odgovora na navedene izzive je industrija začela iskati in ustvarjati drugačna orodja za shranjevanje podatkov. Pogoste značilnosti teh orodij so:

- Ozka specializacija posamičnih rešitev za reševanje najbolj perečih problemov, npr. zmanjševanje odzivnih časov oddaljenih uporabnikov, velika količina zahtev za branje, velika količina zahtev za pisanje, pogosto spreminjajoča se struktura podatkov, nedefinirana struktura podatkov, itd.
- Učinkoviti replikacijski mehanizmi katerih narava zahteva žrtvovanje določene stopnje konsistence podatkov v distribuiranih scenarijih [7]. Te rešitve se načeloma zanašajo na mehko obliko konsistence, imenovano začasna neskladnost (ang. eventual consistency) [37].
- Shramba podatkov je izvedena v obliki asociativnih seznamov.
- Podatkovne baze ne omogočajo "ad-hoc" poizvedb in posledično ne omogočajo podpore jezikom za strukturirano poizvedovanje (SQL). Namesto tega je potrebno vnaprej pripraviti programe, na podlagi katerih podatkovna baza zgradi ustrezne indekse.

V okviru te diplomske naloge sem se odločil predstaviti enega tipičnih predstavnikov novega vala podatkovnih baz, imenovanega Apache CouchDB [18]. Drugi pogosto omenjani oz. uporabljani predstavniki so: Hadoop, Cassandra, Cloudera, MongoDB, Terrastore, Redis, Tokyo Tyrant, MemcacheDB,... [31].

1.1 Cilji diplomske naloge

V okviru tega diplomskega dela nameravam predstaviti podatkovno bazo Apache CouchDB in njeno arhitekturo. Poleg tega se želim seznaniti in popisati delovanje posebne kategorije aplikacij, ki za delovanje potrebujejo zgolj spletni brskalnik in podatkovno bazo CouchDB, tako imenovanih CouchApps. V praktičnem delu je cilj izdelati in predstaviti konkreten primer samostrežene (ang. selfhosted) aplikacije, razvite na CouchDB podatkovni bazi.

Poglavje 2

Uporabljene tehnologije

Pozornost diplomskega dela velja predvsem podatkovni bazi, strežniku in aplikacijam, zgrajenim okrog Apache CouchDB. Vsi omenjeni so del širšega ekosistema, katerega podrobno poznavanje za branje tega dela ni potrebno. Je pa za boljše razumevanje priporočljivo, da bralec vsaj na konceptualnem nivoju razume, čemu služijo posamične omenjene tehnologije in koncepti. S tem namenom je spisan sledeči popis ogrodij in tehnologij.

ACID Je zbirka lastnosti, ki zagotavljajo, da bodo transakcije v podatkovnih bazah izvedene zanesljivo.

AJAX Asynchronous JavaScript And XML [25] je oznaka za arhitekturni koncept, pri uporabi katerega aplikacija v spletnem brskalniku napram storitvam izvaja asinhrone poizvedbe z namenom izvajanja oddaljenih programov, pridobivanja podatkov oz. spreminjanja podatkov v sklopu storitev. Za kodiranje podatkov se uporablja bodisi format XML, bodisi format JSON.b

CouchApps CouchApp je posebna kategorija spletnih aplikacij, katerih značilost je, da za svoje delovanje ne uporabljajo aplikacijskega strežnika, temveč se izvajajo zgolj v sklopu spletnega brskalnika, za hrambo podatkov in obdelave pa uporabljajo neposredno strežnik Apache CouchDB.

Evently Evently je jQuery JavaScript vtičnik (ang. plug-in), ki se uporablja za krmiljenje in proženje JavaScript programov v sklopu aplikacij CouchApp.

HTML Hyper Text Markup Lanugage [39] je označevalni jezik za izdelavo spletnih strani in predstavlja osnovo spletnega dokumenta. S pomočjo HTML ustvarimo strukturo in semantično ureditev dokumenta.

HTTP HyperText Transfer Protocol oz. protokol za prenos nadbesedila je glavna metoda za prenos informacij na spletu. Razvoj HTTP je koordiniral WWW konzorcij in delovne skupine za medmrežni inženiring. Rezultat je bila publikacija serije RFCjev, predvsem RFC 2616, ki definira HTTP/1.1, torej različico v pogosti uporabi dandanes [4]. Glavni gradniki protokola HTTP so URI nazivi oz. URL naslovi in HTTP metode. Protokol HTTP in protokoli TCP/IP omogočajo obstoj spleta in njegovo nadaljno rast in razvoj.

JavaScript JavaScript [6] je izvedba standarda za programske jezike ECMAScript. Uporablja se predvsem v spletnih brskalnikih za potrebe izvajanja programov, povezanih s spletnimi stranmi. Sicer pa se pogosto uporablja kot skriptni jezik za krmiljenje in programiranje kompleksnejših programskih rešitev.

JSON JavaScript Object Notation [26] je format za izmenjevanje podatkov. Temelji na programskem jeziku Javascript.

JQuery JQuery je JavaScript ogrodje za spreminjanje DOM dokumenta. Uporablja se za poenostavljeno in med spletnimi brskalniki neodvisno upravljanje DOM dokumentov.

Mustache Mustache je ogrodje za izdelavo HTML dokumentov na podlagi obrazcev.

OAuth OAuth je avtorizacijski protokol, ki uporabnikom informacijskega sistema omogoča, da z njegovo uporabo tretjim osebam, ki jih protokol imenuje odjemalci, kontrolirano omogoči dostop do omejenega nabora informacij o uporabniku v okviru informacijskega sistema. Protokol OAuth je definiran v RFC 5849 [8].

Python Python [34] je interpretiran, splošno namenski, visokonivojski programski jezik.

REST Representational State Transfer [5] oz. predstavitveni prenos stanja je programska arhitektura za porazdeljene sisteme. Temelji na predpostavki sistema, v katerem imamo odjemalce in strežnike. Odjemalec bodisi miruje oz. ne porablja strežniških virov, bodisi je v procesu spreminjanja stanja in ima posledično napram strežniku odprtih enega ali več zahtevkov. Stanje aplikacije je predstavljeno na podlagi povezav do strežniških virov. Odjemalec lahko povezave uporabi, kadar želi preiti v

novo stanje. Najbolj razširjen primer REST arhitekture je spletni protokol HTTP.

Syslog Syslog [32] je s strani organizacije NIST definiran format za zapisovanje dnevniških zapisov. Vsako Syslog sporočilo oz. zapis je sestavljeno iz treh komponent: prioriteta, glava in vsebina. Pri tem se glava deli na časovni žig in naslov oz. naziv izvora. Vsebina se deli na značko, ki je praviloma oznaka procesa, iz katerega izvira sporočilo, in na telo sporočila.

Ubuntu Linux Ubuntu Linux je distribucija odprtokodnega operacijskega sistema Linux.

UML UML [33] je standardiziran večnamenski modelirni jezik za softverski inženiring.

Poglavje 3

Apache CouchDB

3.1 Splošno o Apache CouchDB

Apache CouchDB [18] je odprtokodni bazni strežnik, za razvojem katerega stoji fundacija Apache [16]. Konceptualno in arhitekturno se močno zgleduje po platformi IBM Lotus Domino [29]. Vodilni razvijalec Damien Katz je z razvojem odprtokodnega sistema želel uporabiti najboljše ideje iz IBM Lotus Domina in jih združiti oz. jih zgraditi okrog tehnologij, na katerih temelji sodobna spletna infrastruktura.

3.1.1 Tehnološke značilnosti

Apache CouchDB je dokumentno orientirana podatkovna baza [36]. Njene glavne tehnološke značilnosti so:

- hramba nestrukturiranih (brez uporabe podatkovne sheme) dokumentov v formatu JSON [26],
- upoštevanje principov ACID,
- vgrajeni replikacijski mehanizmi,
- uporaba vzorca preslikaj in omeji (ang. map and reduce),
- zgrajena je na Erlang OTP [23],
- indeksiranje nestrukturiranih dokumentov v preglede (ang. view),
- za dostop do podatkov uporablja vmesnik API, sledeč principom REST (Representational State Transfer) [5].

Uporaba platforme Erlang OTP omogoča uporabniku baze Apache CouchDB namestitev in uporabo na večino sodobnih oz. popularnih operacijskih sistemov, edina omejitev je podpora Erlang OTP platformi. CouchDB je med drugim podprt na sledečih operacijskih sistemih:

- Android,
- Apple OSX,
- BSD Unix,
- Linux,
- Solaris,
- Palm WebOS,
- Microsoft Windows.

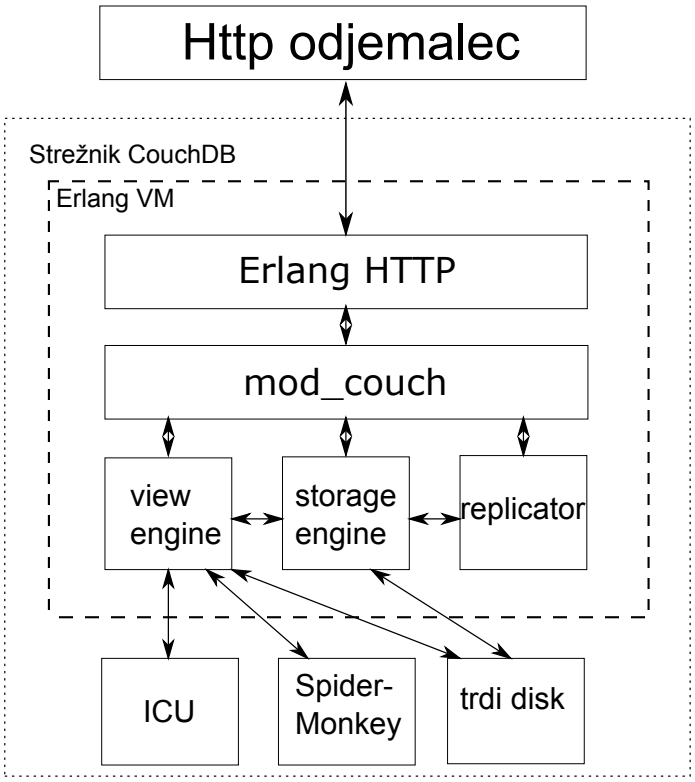
3.2 Arhitektura Apache CouchDB

To poglavje se deli na dva dela - prvi del obravnava izvedbene oz. sistemske značilnosti strežnika CouchDB. Drugi del pa obravnava logične/programske enote, s katerimi ima opravka razvijalec rešitve na CouchDB podatkovni bazi.

3.2.1 Sistemska arhitektura strežnika CocuhDB

Na sliki 3.1 vidimo arhitekturo strežnika podatkovne baze Apache CouchDB.
Razlaga pojmov s slike 3.1:

- HTTP odjemalec: katerakoli naprava s funkcijo pošiljanja zahtev HTTP. Praviloma so to aplikacijski strežniki oz. drugi CouchDB strežniki za potrebe replikacije, lahko pa je to tudi spletni brskalnik neposredno.
- Erlang VM: Celotna programska logika strežnika CouchDB je izvedena v obliki programov za navidezni strežnik Erlang VM (ang . Erlang Virtual Machine):
  - Erlang HTTP [24] je spletni strežnik, uporabljen za komunikacijo z zunanjim svetom. Ena od značilnosti CouchDB je, da protokol HTTP uporablja tako za upravljalske funkcije kot za prenos podatkov.



Slika 3.1: Arhitektura strežnika Apache CouchDB.

  - Mod_couch je strežna komponenta, ki krmili ustvarjanje, naslavljanje, spreminjanje in vzdrževanje podatkovne baze.
  - View engine, je komponenta ki skrbi za kreiranje in vzdrževanje seznamov dokumentov.
  - Storage engine je komponenta, ki skrbi za zapisovanje podatkov na medij za hrambo podatkov, npr. trdi disk, ter je predvsem zadolžena za konsistenco shranjenih podatkov ne glede na količino zahtev.
  - Replicator je komponenta, ki skrbi za sinhronizacijo dokumentov med različnimi replikami iste baze.

- Soodvisne zunanje komponente, ki morajo biti nameščene na operacijskem sistemu, znotraj katerega želimo izvajati CouchDB :
  - ICU (International Components for Unicode) skrbi za ustrezno obrav-

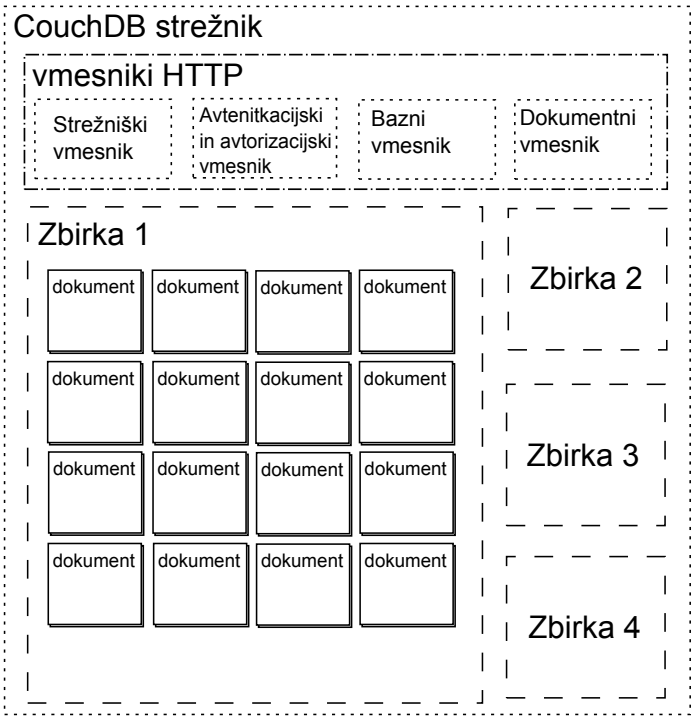
navo, pretvorbo in formatiranje tekstov, datumov in valut v raznih nacionalnih formatih.

– SpiderMonkey [30] je JavaScript pogon, razvit s strani fundacije Mozilla . Najpogosteje je uporabljen za izvajanje JavaScript kode spletnega brskalnika FireFox. Njegova funkcija v CouchDB je izvajanje uporabniških funkcij, s pomočjo katerih view engine izvaja operacije nad dokumenti.

3.2.2 Arhitektura podatkovne baze CouchDB

Ogledali si bomo osnovne gradnike podatkovne baze CouchDB in njihovo namembnost.

Slika 3.2 prikazuje logično organizacijo CouchDB podatkovnih baz in njihovo strukturo.



Slika 3.2: Logična struktura CouchDB podatkovne baze.

Vmesniki HTTP

Podatkovni strežnik Apache CouchDB sprejema vsa navodila, potrebna za delovanje, preko protola HTTP. Vmesnik se deli na štiri področja odgovornosti:

Strežniški vmesnik je namenjen nadzorovanju in upravljanju delovanja strežnika:

- Replikacijski vmesnik: omogoča sinnhronizacijo baz med različnimi strežniki. Vgrajena podpora zanesljivim sinhronizacijskim mehanizmom je ena od ključnih konstrukcijskih odločitev, posledica katerih je marsikateri kompromis na drugih področjih.
- Operativne statistike: uporabniku vračajo informacijo o stanju strežnika, pregled aktivnih opravil.
- Administrativne funkcije: omogočajo ponovni zagon strežnika, pregled log datotek, ipd.

Avtentikacijski in avtorizacijski vmesnik je namenjen upravljanju identitete odjemalcev in ugotavljanju pristopnih pravic uporabnikov. Funkcije tega vmesnika so:

- Avtentikacija: je možna bodisi po OAuth protokolu, avtentikacija s piškotki in tretja možnost, osnovna HTPP avtentikacija po RFC 2617.
- Ureja dostop uporabnikov do posameznih baz.
- Omogoča dodeljevanje vlog uporabnikom, CouchDB pozna tri večje skupine vlog:
  - strežniški administrator: ima pravice do vseh virov na strežniku,
  - bazni administrator: ima pravice do vseh virov znotraj ene baze,
  - bazni čitatelj: ima pravice do branja dokumentov znotraj ene baze; lahko bere in piše podatkovne dokumente, nima pa pravice za pisanje aplikacijskih dokumentov.
- Avtorizacijski vmesnik omogoča izvajanje posebnih funkcij za validacijo osveženih dokumentov (ang. Document Update Validation functions), ki se izvajajo ob vsakem shranjevanju podatkovnega dokumenta - na ta način lahko precej bolj razvejimo pristopne pravice do podatkov v bazi.

Bazni vmesnik vsebuje funkcije, ki se nanašajo na posamične baze podatkov. Njegovo področje odgovornosti zajema:

- Vmesnik za kreiranje in brisanje podatkovnih baz.
- Omogoča adminstrativne funkcije na nivoju podatkovne baze: kompaktiranje, pregled stanja, stanje pravic uporabnikov na bazi.
- Pregled nad vsebino podatkovne baze in spremembah nad dokumenti.

Najnižje v hierarhiji vmesnikov se nahaja dokumentni vmesnik, ki je namenjen uporabi podatkov v podatkovni zbirki in omogoča:

- Dodajanje, spreminjanje, kopiranje in brisanje dokumentov.
- Dodajanje, brisanje in pridobivanje priponk z dokumentov.
- Dodajanje, brisanje, spreminjanje in kompaktiranje pregledov.
- Na voljo so posebni vmesniki za aplikacijske dokumente.

Gradniki podatkovne baze CouchDB

V prejšnjem poglavju smo si v grobem ogledali, katere funkcije API omogoča vmesnik HTTP. V tem poglavju pa si bomo ogledali, nad čem se dejansko izvajajo omenjene funkcije API. Pri tem se bomo omejili na obravnavo podatkov.

V podatkovni bazi CouchDB podatke vedno obravnavamo v kontekstu sledečih logičnih kategorij:

- Podatkovne baze: vsaka podatkovna baza predstavlja eno fizično datoteko na mediju za shranjevanje podatkov.
- Vsaka baza je sestavljena iz množice dokumentov. Dokument je osnovna enota za shranjevanje podatkov - in predstavlja en zapis (ang. record). Dokumenti se delijo na:
  - aplikacijske dokumente (ang. design documents) in
  - podatkovne dokumente.

Aplikacijski dokumenti se od podatkovnih razlikujejo po tem, da se v aplikacijskih dokumentih nahajajo aplikacije in njihovi gradniki. Poleg tega imajo aplikacijski dokumenti svoj naslovni prostor, aplikacijske dokumente naslavljamo z vzorcem `_design/`. Podatkovne dokumente naslavljamo z unikatnim ključem, ki se ne sme začeti z vzorcem `_design/`.

Splošno o dokumentih

Dokumenti so atomarne enote za hrambo podatkov v podatkovni bazi CouchDB. Vsebina dokumenta je zapisana v formatu JSON. Primer dokumenta vidimo na sliki 3.3. Vsak dokument v bazi CouchDB mora imeti izpolnjeni dve polji:

- `_id`: unikatni ključ vsakega dokumenta.
- `_rev`: vsebuje informacijo o reviziji dokumenta, sestavljeno je iz sekvečne številke revizije dokumenta in ETAG časovnega žiga. Za shranjevanje dokumenta v podatkovno bazo mora naša kopija imeti isto vrednost polja `_rev` kot dokument na strežniku, sicer bo strežnik zaznal konflikt in ne bo dovolil spremembe dokumenta. Takšen mehanizem se uporablja za zagotavljanje konsistence podatkov.

```
{
  "_id": "bf30cd74ec3f7657ed5396d4d73e8822",
  "_rev": "1-f9ef6540732c7215a9f34450b3c5e487",
  "tipdokumenta": "oseba",
  "ime": "Janez",
  "priimek": "Stupar",
  "datumrojstva": "04.03.1983"
}
```

Slika 3.3: Primer dokumenta v bazi Apache CouchDB.

Poleg tega podatkovna baza CouchDB pozna še sledeča posebna rezervirana polja:

- `_attachments`: polje vsebuje meta-podatke o priponkah oz. datotekah na dokumentih.
- `_deleted`: polje vsebuje informacijo o izbrisu dokumenta. To pomeni, da po naslednjem ciklu kompaktiranja dokument ne bo več na voljo.
- `_revisions` in `_rev_info`: hranita informacijo o prejšnjih verzijah dokumenta.
- `_conflicts` in `_deleted_conflicts`: hranita informacijo o morebitnih konfliktih in njihovem razreševanju.

Aplikacijski dokumenti

V bazi Apache CouchDB aplikacijske dokumente označujemo na dva načina. Aplikacijski dokument prepoznamo po tem, da ima v polju `_id` vrednost, ki se začne z labelo `_design/` oz. `_local/`. V slednjem primeru dokument lahko imenujemo tudi lokalni dokument. Lokalni dokumenti so namenjeni uporabi v matični bazi in se ne replicirajo.

Aplikacijski dokumenti hranijo in krmilijo funkcije, na podlagi katerih podatkovni strežnik Apache CouchDB pripravi odjemalcem ustrezno predstavitev podatkov.

Seznam polj, ki so rezervirana za uporabo v aplikacijskih dokumentih:

- `shows`: vsebuje funkcije za prikazovanje dokumentov (ang. show functions). Funkcije služijo prikazovanju posamičnih dokumentov, zapisanih v JSON formatu, v drugih formatih npr. HTML ali XML.

- `lists`: v polju `lists` se nahajajo funkcije za prikazovanje pregledov (ang. list functions). Od funkcij za prikazovanje dokumentov se razlikujejo v tem, da se izvajajo nad seznami dokumentov, točneje nad pregledi (ang. views).

- `validate_doc_update`: vsebuje funkcije, s pomočjo katerih se lahko strežnik odloči, ali je sprememba dokumenta, ki jo zahteva odjemalec, dovoljena. Vsak aplikacijski dokument lahko vsebuje eno funkcijo `validate_doc_update`. Za uspešno shranjevanje dokumenta mora le-ta prestati teste validacijskih funkcij iz vseh aplikacijskih dokumentov v bazi.

- `update`: upravljalec posodobitev (ang. update handler) vsebuje funkcije, ki jih strežnik izvaja nad dokumenti. Funkcije služijo strežniškim obdelavam v kontrastu z običajnim zaporedjem pridobi, spremeni, vrni. Upravljalca posodobitev sproži odjemalec s HTTP zahtevkom POST ali PUT.

- `views`: polje vsebuje funkcije, s pomočjo katerih CouchDB strežnik zgradi preglede[1] (ang. views). Pregledi so indeksi dokumentov in so zgrajeni na podlagi dvostopenjske operacije, imenovane preslikaj in omeji (ang. map and reduce). Operacija preslikave se izvede nad vsemi dokumenti v bazi, pri čemer funkcija preslikave v primeru, da posamični dokument ustreza njenim pogojem, zgenerira pare `<ključ, vrednost>`. Ti zgenerirani pari, sortirani po ključu, predstavljajo pregled, lahko pa

so uporabljeni kot vhod v omejevalno fazo. Tedaj omejavalna funkcija (ang. reduce) združi preslikave v končne rezultate pregledov. Število rezultatov omejevalne funkcije mora biti manjše od števila vhodnih ključev in vrednost.

[1]V slovenski strokovni literaturi se sicer za angleški izraz view uporablja prevod pogled. Z uporabo izraza pregled sem želel izpostaviti razlike v izvedbi in delovanju med pogledi in pregledi. V relacijskih podatkovnih zbirkah pogledi ne vsebujejo kopij podatkov iz tabel in omogočajo združevanje (ang. join) podatkov iz različnih tabel. Pregled pa vsebuje kopije podatkov iz izvornih dokumentov in vsak vnos oz. vrstica v pregledu lahko izvira zgolj iz enega dokumenta.

Poglavje 4

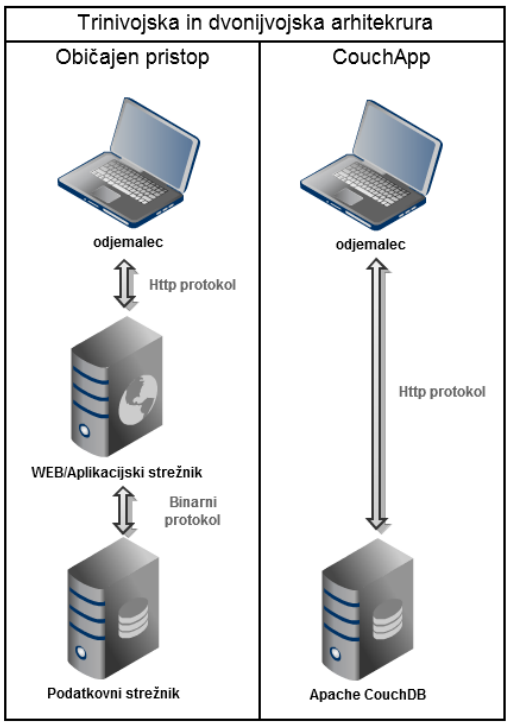
CouchApps

Arhitekturne in konceptualne posebnosti, kot je uporaba HTTP vmesnika API in vgrajeni mehanizmi za predstavitev podatkov, omogočajo podatkovnemu stežniku CouchDB, da deluje kot samostojen aplikacijski strežnik. Najpogostejši pristop pri gradnji aplikacij, ki sledijo vzorcu strežnik - odjemalec je, da se sistem gradi v treh nivojih, (slika 4.1). Takšen pristop omogoča tudi CouchDB. Tukaj se temu primeru uporabe ne bomo posvečali nadrobneje, bomo pa si ogledali primer uporabe, kjer je sistem strukturiran v dva nivoja (slika 4.1). Takšne aplikacije se v žargonu CouchDB imenujejo CouchApps.

Vlogo odjemalca v sistemu lahko opravlja spletni brskalnik, aplikacijski strežnik ali drugi CouchDB strežniki. Pri tem velja opozoriti na možnost namestitve CouchDB strežnika lokalno na odjemalcu, tako da je odjemalec sam sebi strežnik. Po potrebi pa podatke z drugimi strežniki sinhroniziramo z uporabo mehanizma replikacije.

4.1 Zgradba aplikacije CouchApp

CouchApp aplikacija je zgrajena iz vsaj enega aplikacijskega dokumenta in mnogih podatkovnih dokumentov. S pomočjo aplikacijskih dokumentov strežnik CouchDB interpretira shranjene podatke in jih odjemalcu pošilja v zahtevani obliki. Temu namemu služijo prikazi (ang. show functions) in seznami (ang. list functions), ki se izvajajo nad pregledi (ang. views). Celotna logika aplikacije je shranjena v aplikacijskih dokumentih. Običajno gre za spletno aplikacijo, temelječo na HTML ogrodju, oblikovanem s CSS dodatki, ter JavaScript programu, lahko pa je tudi bogata spletna aplikacija (ang. rich internet application), npr. Adobe Flash aplikacija, ki jo praviloma v lastnem okolju izvaja odjemalec. Izvajanje poslovne logike na strežniku omogočajo upravljalci poso-



Slika 4.1: Primerjava trinivojske in dvonivojske arhitekture s podatkovnim strežnikom CouchDB.

dobitev (ang. update handlers). V CouchDB je vgrajen varnostni mehanizem, imenovan kontrola dostopa (ang. access control) oz. odobravanje posodabljanja dokumentov (ang. document update validation). Za uporabnikom prijazno naslavljanje aplikacijskih virov pa imamo na voljo URL prepisovalnike (ang. rewrites).

Podatkovni dokumenti so delno strukturirani JSON objekti. Delno strukutirani podatki so podatki, ki imajo podatkovno shemo podano implicitno kot del podatkov samih in ponavadi nimajo ločene oz. formalne sheme, napram kateri bi lahko preverjali pravilnost formata zapisa podatkov [2]. To pomeni, da je obremenitev, povezana s konsistenco podatkov določenega tipa, v celoti prepuščena aplikacijski logiki in ponavadi razdrobljena po večih gradnikih aplikacije. Po eni strani to dopušča prožnost podatkovnih struktur, po drugi strani pa pomeni tudi večjo obremenitev razvijalcev, saj je za zagotavljanje kvalitete podatkov potrebna večja pazljivost in dodaten trud.

4.2 Delovanje CouchApp aplikacije

Kot smo že omenili, CouchApp aplikacije za delovanje ne potrebujejo aplikacijskega strežnika, temveč odjemalec dostopa do podatkov/virov neposredno na CouchDB strežniku. Ponazoritev delovanja CouchApp aplikacije bomo izvedli na enostavni CouchApp aplikaciji, imenovani Sofa [13].

Sofa je odprtokodna spletna aplikacija za ustvarjanje blogov, izvedena na platformi CouchDB. Tehnično je zgrajena iz HTML ogrodja, oblikovanega s CSS predlogami. Poslovna logika je izvedena z JavaScript aplikacijo, temelječo na aplikacijskem ogrodju JQuery. Za ponazoritev delovanja aplikacije bomo opisali sledečo sekvenco dogodkov:

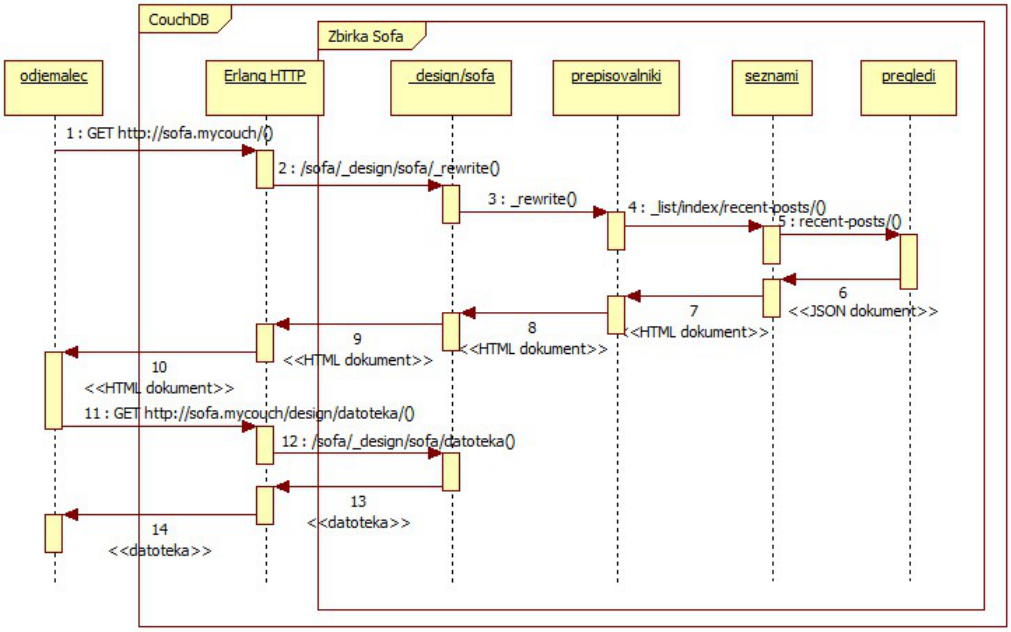
1. Dostop do aplikacije in opis dogodkov, ki se zgodijo ob zagonu aplikacije.
2. Pregled posamičnega vnosa.
3. Prijava uporabnika v sistem.
4. Dodajanje novega vnosa v blog.

4.2.1 Dostop do aplikacije

Na sliki 4.2 vidimo diagram zaporedja za začetni HTTP zahtevek odjemalca napram CouchDB strežniku, na katerem gostuje blog Sofa.

Sledi opis zaporedja dogodkov, ki se izvrši ob zahtevku odjemalca CouchDB strežniku:

1. Odjemalec izvrši zahtevek HTTP GET na naslov `http://sofa.mycouch/`. Erlang HTTPD v strežniški nastavitvi Virtual Hosts prebere, da se klici, usmerjeni na poddomeno sofa, usmerjajo na lokalni naslov `/sofa/_design/sofa/_rewrite/`, pri čemer komponente URL pomenijo sledeče:
   - `/sofa/ ...` predstavlja bazo z imenom sofa,
   - `_design/sofa ...` predstavlja aplikacijski dokument znotraj baze sofa,
   - `_rewrite/ ...` predstavlja URL prepisovalnik (ang. rewrite handler), ki v okviru aplikacijskega dokumenta omogoča nastavitve interpretacije naslovov URL.
2. Erlang HTTPD usmeri zahtevek na URL prepisovalnik, ki na podlagi pravil, zapisanih v aplikacijskem dokumentu, izvaja preusmeritve naslovov.



Slika 4.2: Diagram poteka operacije prvega dostopa do aplikacije Sofa.

3. V konkretnem primeru je bilo uporabljeno prepisovalno pravilo na sliki 4.3. Pravilo pomeni, da če ob naslavljanju prepisovalnika ne podamo URL parametrov (atribut `from`), potem se klici privzeto usmerijo na podani naslov (atribut `to`). Poleg tega nam prepisovalnik omogoča še določanje privzetih query parametrov (atribut `query`).

4. V konkretnem primeru zahtevek usmerimo na:

   - `/_list/ ...` predstavlja upravljalnik seznamov, ta ukaz strežniku pove, da želimo prikazati CouchDB seznam,
   - `index/ ...` ime seznama, katarega želimo prikazati,
   - `recent-posts/ ...` ime pregleda, ki ga seznam uporabi za vhod.

5. CouchDB izvede JavaScript funkcijo za generiranje seznamov pod imenom `index`. Ta funkcija vsebuje vso programsko logiko, na podlagi katere CouchDB lahko izdela željen rezultat. Funkcija `index` omogoča generiranje bodisi HTML dokumenta, bodisi Atom vira, odvisno od tega, kakšen format smo specificirali v glavi HTTP zahtevka. Glede na, to da

```
{
    "to": "_list/index/recent-posts",
    "from": "",
    "query": {
        "limit": 10,
        "descending": true
    }
}
```

Slika 4.3: Primer prepisovalnega pravila.

v našem primeru želimo kot rezultat dobiti HTML dokument, funkcija za generiranje seznama pripravi ogrodje HTML dokumenta, ki ga potem popolni z izvlečki dokumentov iz pregleda `recent-posts`. V našem primeru zgenerirani HTML dokument vsebuje 10 najnovejših objav v blogu, sortiranih padajoče po datumu.

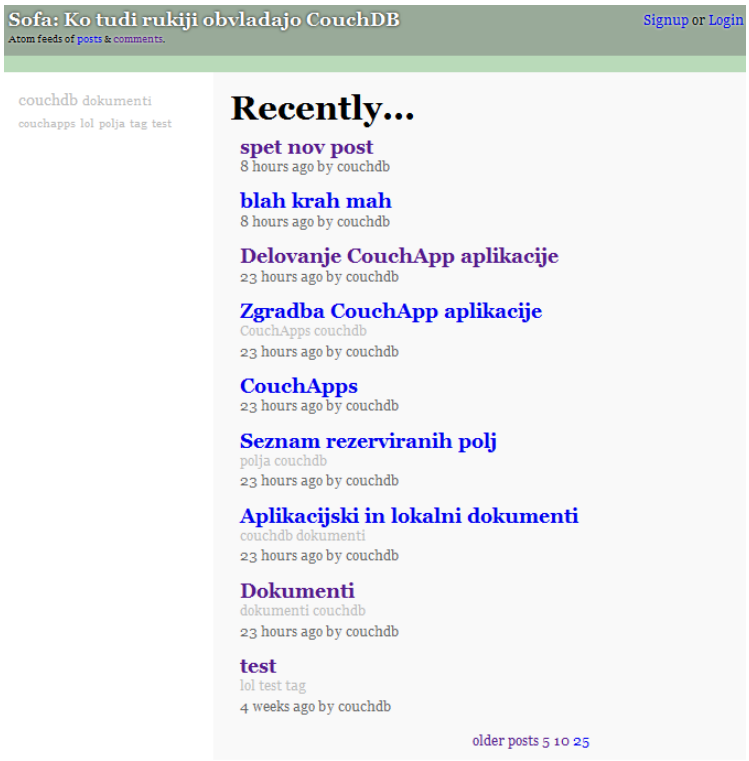
6. V korakih 6-10 s slike 4.2 se HTTP zahtevek odvije in brskalniku se vrne surov HTML dokument.

7. Koraki 11-14 s slike 4.2 predstavljajo zahtevke strežniku za dodatne vire, kot so odjemalske JavaScript knjižnice, CSS označevalne datoteke, slike, itd. Ti viri so vsi zapisani v obliki datotek, pripetih na aplikacijski dokument.

Na sliki 4.4 vidimo končni rezultat poizvedbe na naslov `http://sofa.mycouch/`.

Za lažje razumevanje dogajanja v ozadju pa si bomo tukaj podrobneje ogledali, kaj se dogaja v ozadju korakov 4 in 5 iz zgornjega zaporedja dogodkov.

Kot smo že omenili, se pri prikazovanju vsebin uporabljata dva mehanizma, seznam in pregled. Pregled je poljuben indeks dokumentov, ki ga strežnik ustvari na podlagi funkcije za preslikavo. Na sliki 4.5 vidimo funkcijo, na podlagi katere se izvrši generiranje pregleda z imenom recent-posts. Proces, ki se ukvarja z indeksiranjem dokumentov v strežniku CouchDB, ob shranitvi vsakega dokumenta nad njim izvrši vse preslikovalne funkcije v okviru matične baze dokumenta. Ob zahtevku za osvežitev indeksa pa izvrši preslikovalno funkcijo nad vsemi dokumenti, ki so bili shranjeni od zadnjega osveževanja indeksa.



Slika 4.4: Domača stran aplikacije Sofa.

```
function(doc) {
  if (doc.type == "post") {
    emit(new Date(doc.created_at), doc);
  }
};
```

Slika 4.5: Preslikovalna funkcija pregleda recent-posts.

Preslikovalna funkcija s slike 4.5 je sestavljena iz dveh operacij. Prva je preverba vrednosti polja `type`. S pomočjo te preverbe zagotovimo, da se v pregledu nahajajo zgolj vrednosti s tistih dokumentov, katerih lastnosti nam ustrezajo. Druga operacija pa je v CouchDB vgrajena funkcija, s podpisom `emit(ključ, vrednost)`. Funkcija `emit()` dovoljuje uporabo kompleksnih objektov za ključe in vrednosti. V našem primeru za ključ uporabimo datum nastanka in za vrednost celoten dokument. Na sliki 4.6, vidimo primer pregleda, zgeneriranega na podlagi zgornje funkcije. Pomembna lastnost funk-

cij za ustvarjanje pregledov je, da morajo biti čiste funkcije. Čiste funkcije morajo zadostiti dvema zahtevama: 1. Funkcija se na iste vhodne podatke vedno odzove enako in se pri izračunih ne sme zanašati na skrite informacije, ki bi se lahko spremijale tekom izvajanja programa oz. med posamičnimi izvedbami programa. 2. Preračun razultata ne sme povzročati semantičnih stranskih učinkov, kot je npr. mutacija objektov ali proženje vhodno izhodne naprave [38]. Poleg tega v omenjenih funkcijah ne moremo uporabljati zunanjih JavaScript knjižnic.

```
{
    "total_rows": 2,
    "offset": 0,
    "rows": [{
        "id": "Delovanje-CouchApp-aplikacije",
        "key": "2011-01-09T22:14:31.366Z",
        "value": {
            "_id": "Delovanje-CouchApp-aplikacije",
            "_rev": "1-ef62e1a16affb076193d26f216a5987b",
            "type": "post",
            "format": "markdown",
            "author": "couchdb",
            "body": "Kot smo omenili ...",
            "title": "Delovanje CouchApp aplikacije",
            "tags": [""],
            "created_at": "2011-01-09T22:14:31.366Z"
        }
    }, {
        "id": "Zgradba-CouchApp-aplikacije",
        "key": "2011-01-09T22:13:36.073Z",
        "value": {
            "_id": "Zgradba-CouchApp-aplikacije",
            "_rev": "1-cab5682766090f3a35fab37e76c39a1d",
            "type": "post",
            "format": "markdown",
            "author": "couchdb",
            "body": "CouchApp aplikacija ...",
            "title": "Zgradba CouchApp aplikacije",
            "tags": ["CouchApps", "couchdb"],
            "created_at": "2011-01-09T22:13:36.073Z"
        }
    }]
}
```

Slika 4.6: Pregled recent-posts.

Kot vidimo, je zgeneriran pregled povsem običajen JSON objekt s tremi lastnostmi:

- `total_rows`: koliko vnosov vsebuje celoten pregled.
- `offset`: odmik prve vrnjene vrednosti od začetka pregleda.
- `rows`: seznam JSON objektov, pri čemer so objekti sortirani po ključu. Lastnosti objektov pa so sledeče:
  - `id`: unikatni identifikator dokumenta,
  - `key`: ključ ki je bil podan kot prvi parameter `emit()` funkcije,
  - `value`: vrednost, podana kot drugi parameter `emit()` funkcije.

V poljubni spletni aplikaciji bi lahko do podatkov dostopali tudi neposredno preko pregledov z uporabo AJAX [25] operacij ter za formatiranje podatkov potem skrbeli v brskalniku. Vendar se to v našem primeru izvede na strežniku z uporabo mehanizma seznamov. Kot smo že omenili, so seznami funkcije, s pomočjo katerih poskrbimo za željeno oz. primerno predstavitev podatkov, ki se nahajajo v pregledih. Najenostavnejša definicija je, da je to funkcija, ki se izvrši nad vsemi elementi pregleda in poskrbi, da odjemalec podatke prejme v željenem formatu. Ravno tako kot funkcije za generiranje pregledov morajo biti tudi funkcije za generiranje seznamov čiste funkcije in ne smejo imeti stranskih učinkov. Lahko pa uporabljamo zunanje JavaScript module, le-te nalagamo neposredno iz aplikacijskih dokumentov.

Koda seznama `index` je na voljo na povezavi [14], na naslovu [15] pa se nahaja predloga Mustache [28], uporabljena za izdelavo dokumenta HTML. Na tem mestu bi samo poudarili nekatere splošne lastnosti delovanja seznamov ter konkretne izvedbene značilnosti seznama `index`.

JavaScript funkcije, uporabljene v seznamu index

CouchDB ima več vgrajenih funkcij, s pomočjo katerih si olajšamo pisanje seznamov. Sledi kratek opis delovanja funkcij, uporabljenih v seznamu `index`:

- `require(path)` : funkcija `require()` se uporablja za nalaganje zunanjih JavaSript modulov. S parametrom `path` povemo strežniku, kateri modul naj naloži. Module naslavljamo relativno glede na aplikacijski dokument. Edina omejitev zunanjih modulov je, da morajo biti napisani v CommonJS [22] narečju jezika JavaScript.

- `provides(format, fun())`: funkcijo `provides()` uporabimo za izbiro formata, v katerem naj bo izražen seznam. Tako za vsak podprt format v seznamu napišemo svojo funkcijo `provides()`, v kateri s parametrom `format` prijavimo tip rezultata, medtem ko s parametrom `fun()` zgeneriramo vsebino v zahtevanem formatu,
- `send(data)`: funkcijo uporabljamo za vračanje podatkov odjemalcu,
- `getRow()`: funkcija vrne naslednji zapis iz pregleda, ko zapisov zmanjka, vrne vrednost `null`.

Generiranje kode HTML

Najenostavnejši način generiranja predstavitvene kode je, da v seznamu željene vsebine izpisujemo neposredno v medpomnilnik za odgovor klientu, npr.: `send («!DOCTYPE html><html> ... </html>")`, tak pristop je sicer enostaven, vendar ne pretirano robusten. Alternativa je uporaba ogrodja za generiranje dokumentov na podlagi predlog. V seznamu `index` sta uporabljena oba pristopa. Za generiranje vsebine v Atom formatu je uporabljen enostavnejši pristop, skladno z enostavnostjo formata. Za generiranje kode HTML pa je avtor uporabil JavaScript orodje za delo s predlogami Mustache. Izdelava rezultata izgleda tako, da najprej pripravimo ogrodje (npr. dokument HTML), v katerega vstavimo začasne vrednosti, označene z dvojnimi zavitimi oklepaji npr. `{{uporabniskoIme}}`. Zgled takšnega ogrodja lahko vidimo na sliki 4.7.

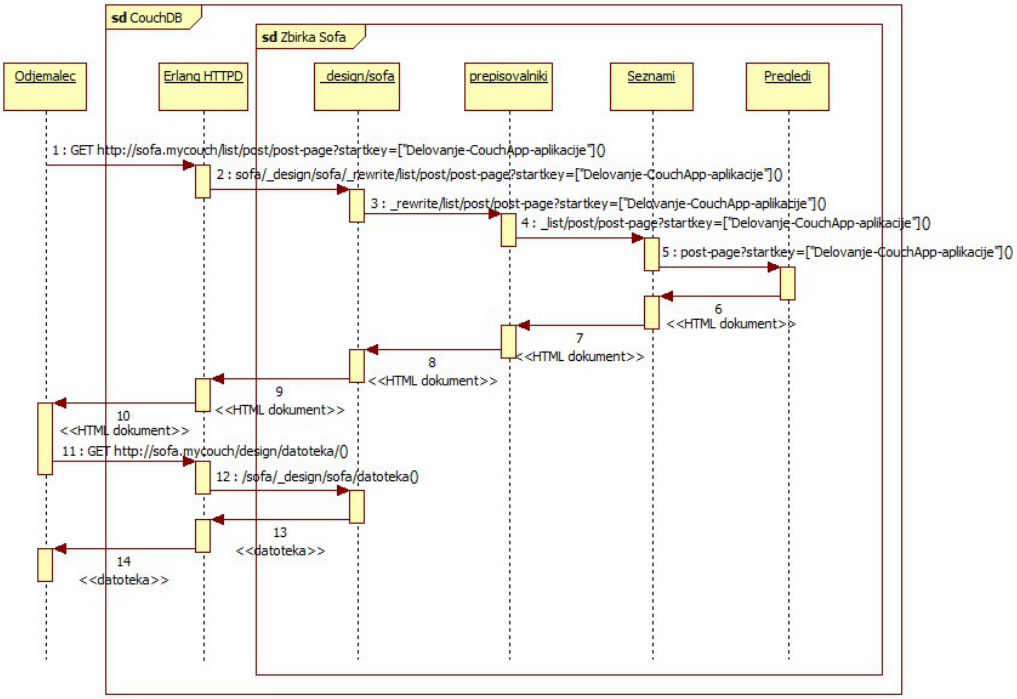
```
1 <div class="avatar">
2   {{#gravatar_url}}<img src="{{gravatar_url}}"/>{{/gravatar_url}}
3   <div class="name">
4     {{nickname}}
5   </div>
6 </div>
7 <p>Hello {{nickname}}!</p>
8 <div style="clear:left;"></div>
```

Slika 4.7: Primer Mustache ogrodja.

V seznamu nato pripravimo JSON objekt, v katerem ustvarimo lastnosti, katerih imena se ujemajo z imeni začasnih vrednosti v ogrodju. Orodje Mustache vrednosti iz objekta zduži z ogrodjem in rezultat vrne brskalniku.

4.2.2 Pregled objave v blogu

Tukaj si bomo ogledali, kaj se v aplikaciji Sofa zgodi ob dostopu do posamične objave v blogu. V izogib podvajanju bomo tukaj obravnavali zgolj tiste dogodke, ki se razlikujejo od zaporedja, predstavljenega na sliki 4.2.



Slika 4.8: Diagram poteka pregleda posamičnega vnosa v blogu.

Na sliki 4.8, vidimo ponazoritev zaporedja dogodkov, ki se izvrši ob zahtevku za pregled posamičnege objave v blogu. O zaporedju velja omeniti sledeči podrobnosti:

1. Odjemalec izvrši HTTP GET zahtevek na naslov `http://sofa.mycouch/list/post/post-page?startkey=["Delovanje-CouchApp-aplikacije"]`. Podobno kot v prejšnjem poglavju se izvede Virtual Host prevedba naslova in prevedba v prepisovalniku:
   - `/list/ ...` se prevede v `/_list/` - upravljalnik seznamov.
   - `post/ ...` ime seznama, ki ga želimo izvršiti.

- `post-page/ ...` ime pregleda, ki ga želimo podati kot vhod seznamu.
- `?startkey=["Delovanje-CouchApp-aplikacije"] ...` dodatni parametri poizvedbe. Na podlagi poizvedbenih parametrov nam strežnik pripravi množico dokumentov oz. rezultatov, na podlagi katerih potem npr. zgeneriramo seznam.

2. Preostanek operacije se izvrši po istih korakih kot zgled v prejšnjem poglavju, slika 4.2.

Za razliko od prejšnjega poglavja, kjer je bil rezultat poizvedbe seznam objav, je rezultat te poizvedbe ena sama objava, kar vidimo na sliki 4.9.

Slika 4.9: Rezultat zahtevka za posamično objavo.

Pri tem primeru se splača za trenutek ustaviti in si ogledati zanimiv implementacijsiki detajl. Pri izvedbi te operacije se nam postavi zanimivo vprašanje: zakaj uporabljamo funkcijo `_list` in ne `_show`, ki naj bi bila namenjena prikazovanju posamičnih dokumentov. V konkretnem primeru se je avtor odločil

za način izvedbe, kjer na podlagi dveh različnih tipov dokumentov, objave in komentarja, ustvarjamo en pregled ter posledično tudi HTML dokument ustvarimo v enem klicu strežnika.

Aplikacija omenjeno obnašanje doseže tako, da za generiranje pregleda uporabi funkcijo s slike 4.10.

```
1  function(doc) {
2    // !code helpers/md5.js
3
4    if (doc.type == "post") {
5      emit([doc._id], doc);
6    } else if (doc.type == "comment") {
7      if (doc.commenter && doc.commenter.email && !doc.commenter.
           gravatar_url) {
8        doc.commenter.gravatar = hex_md5(doc.commenter.email);
9      }
10     emit([doc.post_id, doc.created_at], doc);
11   }
12 };
```

Slika 4.10: Preslikovalna funkcija pregleda post-page.

Funkcija, prikazana na 4.10, deluje tako, da v pregled izpisuje dva tipa dokumentov, pri čemer za dokumente tipa objava ("post") za ključ uporabi unikatni identifikator dokumenta, medtem ko za dokumente tipa komentar ("comment") za ključ uporabi kompleksen objekt, sestavljen iz unikatnega identifikatorja objave, na katero se nanašata komentar, in čas nastanka dokumenta. Rezultat takšne funkcije je pregled, viden na sliki 4.11.

Na ta način je avtor aplikacije izrabil pravila za sortiranje zapisov v pregledih in dosegel, da se dokumenti v pregledu sortirajo tako, da se v pregledu najprej pojavi objava oz. nosilni dokument, neposredno za njim pa še vsi komentarji, sortirani po času nastanka.

Uporaba pregleda post-page v seznamu post nato izgleda tako, da rezultate pregleda omejimo z začetnim ključem npr.: `startkey=["nazivobjave"]`, kar pomeni, da je prvi zapis, ki ga dobi seznam z operacijo `getRow()`, objava, vse sledeče pa komentarji. Seznam zaključi z branjem zapisov iz pregleda, ko prebere prvi naslednji zapis tipa objava ("post").

```
{
    "total_rows": 15,
    "offset": 0,
    "rows": [{
        "id": "Delovanje-CouchApp-aplikacije",
        "key": ["Delovanje-CouchApp-aplikacije"],
        "value": {
            "_id": "Delovanje-CouchApp-aplikacije",
            "type": "post",
            ....
        }
    }, {
        "id": "e0c4c7dac633eff92b789050e90092e8",
        "key": ["Delovanje-CouchApp-aplikacije", "2011-01-10T12
            :24:48.038Z"],
        "value": {
            "_id": "e0c4c7dac633eff92b789050e90092e8",
            "type": "comment",
            ....
        }
    }, {
        "id": "e0c4c7dac633eff92b789050e900512e",
        "key": ["Delovanje-CouchApp-aplikacije", "2011-01-11T12
            :24:48.038Z"],
        "value": {
            "_id": "e0c4c7dac633eff92b789050e900512e",
            "type": "comment",
            ....
        }
    }]
}
```

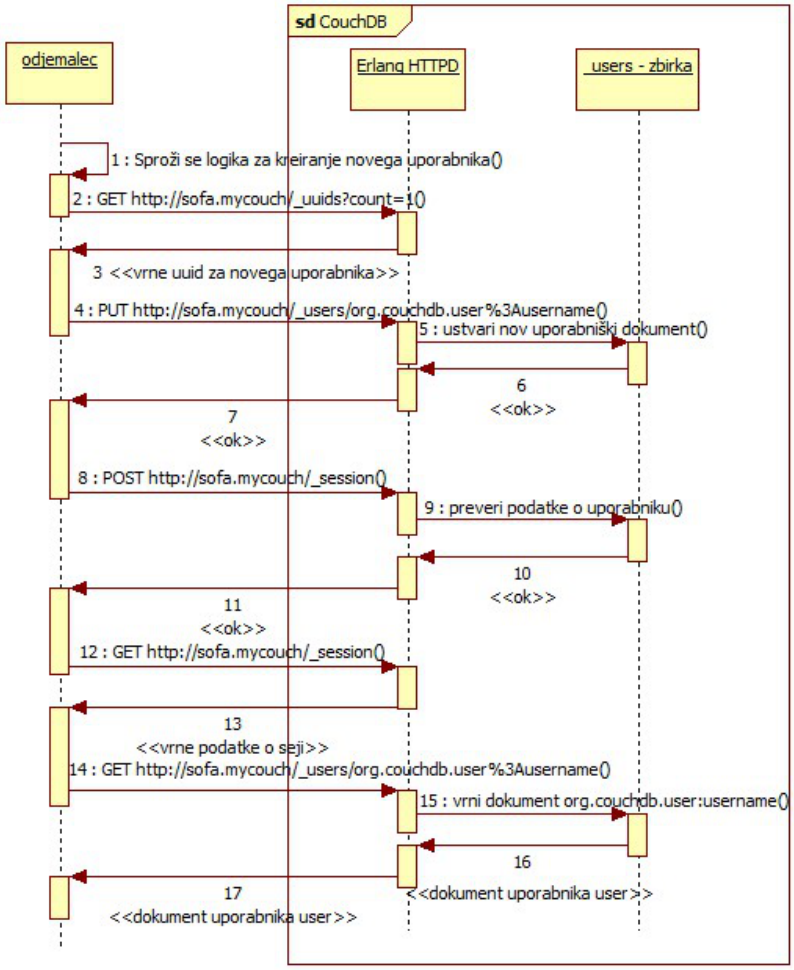
Slika 4.11: Rezultat pregleda post-page.

4.2.3 Prijava v aplikacijo

Za dodajanje in spreminjanje vsebin v aplikaciji Sofa morajo biti uporabniki prijavljeni, oz. morajo dokazati svojo identiteto ob izvedbi določenih operacij. Te operacije so bodisi omejene s strani strežnika v primeru administratorskih operacij [1], sicer pa so lahko omejene še na podlagi vlog oz. članstva v skupinah na nivoju posamičnih baz [20]. Tretji način avtorizacije je preverjanje uporabniških pravic v `validate_doc_update` funkcijah [19].

Na tem mestu si bomo ogledali, kako sta v aplikaciji Sofa izvedena mehanizma ustvarjanja in preverjanja identitete uporabnikov. Aplikacija Sofa za

avtentikacijo uporablja mehanizem z uporabo piškotkov (ang. cookie-based authentication) [9]. Na sliki 4.12 vidimo sekvenco dogodkov, ki se izvrši ob kreiranju novega uporabnika skozi aplikacijo Sofa.



Slika 4.12: Diagram poteka ob registraciji novega uporabnika.

Opis sekvence dogodkov, ki se izvrši ob registraciji novega uporabnika:

- V koraku 2 odjemalec izvrši klic na naslov `http://sofa.mycouch/_uuids?count=1`, kjer se nahaja storitev, ki odjemalcem omogoča generiranje UUID. Ta storitev je koristna tudi ob ustvarjanju dokumentov. Prido-

bimo lahko poljubno količino UUID ključev, tako da vnesemo poljuben `count` parameter URL.

- V koraku 4 odjemalec zgenerira nov uporabniški dokument in ga zapiše v bazo uporabnikov na strežniku.
- V koraku 8 odjemalec izvrši prvo prijavo z novim uporabnikom. To izvede tako, da uporabniško ime in geslo pošlje na storitev `/_session/`. Omenjena storitev preveri pravilnost podatkov v bazi `_users`. Ker so podatki pravilni, strežnik odjemalcu vrne avtentikacijski piškot.
- V koraku 12 odjemalec izvrši povpraševanje po seji uporabnika, strežnik prepozna veljaven avtentikacijski piškot in odjemalcu vrne podatke o seji uporabnika. To so: katere vloge ima uporabnik, uporabniško ime, skozi kateri imenik je prijavljen uporabnik ter po katerem protokolu.
- V koraku 14 odjemalec izvrši poizvedbo po uporabniškem dokumentu, na podlagi katere preveri, če je bil celoten proces izpeljan pravilno.

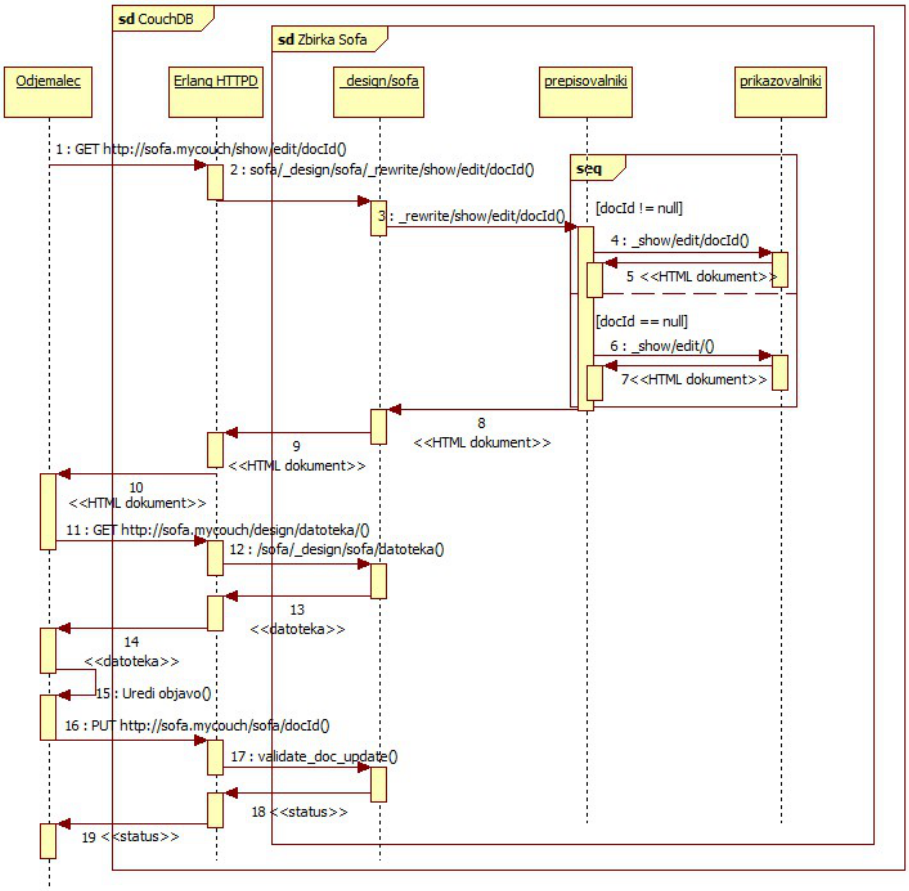
Proces prijave registriranega uporabnika je podoben sekvenci s slike 4.12, vendar se tedaj izvršita samo koraka 8 in 12.

4.2.4 Ustvarjanje nove objave v blogu

Ker pregledovanje in prijavljanje v aplikacijo brez vsebine nima nobenega smisla, si velja ogledati še, na kakšen način lahko izvedemo novo objavo v blogu. Na sliki 4.13 je prikazan tok dogodkov, ki se izvrši ob urejanju obstoječe oz. ob dodajanju nove vsebine.

Poleg že poznanih in videnih operacij se izvede:

- Koraki od 3 do 8 prikazujejo zahtevek prikazovalniku (ang. show function) `/_show/edit/docId`, na podlagi katerega aplikacija zgenerira HTML formo, ki uporabniku omogoči urejanje objave. Isti prikazovalnik je uporabljen tako za nove kot za obstoječe dokumente.
- V korakih od 15 do 19 uporabnik zaključi urejanje objave in sproži kodo za objavo vsebine. Odjemalec naslovi HTTP PUT zahtevek na naslov `http://sofa.mycouch/sofa/docId`, pri čemer se kot del procesa shranjevanja izvršijo posebne funkcije `validate_doc_update()`. Pozoren bralec bo opazil, da je zahtevek usmerjen neposredno na matično bazo. To pomeni, da se posledično izvršijo validacijske funkcije vseh aplikacijskih dokumentov v bazi. Če se validacijska funkcija izvede brez



Slika 4.13: Diagram poteka ustvarjanja nove objave.

napake, se dokument shrani v bazo. Validacijske funkcije omogočajo širok spekter uporabnih operacij. Poleg preverjanja pravilnosti podatkov si z njimi lahko pomagamo tako, da npr.:

- Izvajanje funkcije omejimo samo na dokumente z določenimi lastnostmi/vrednostmi,
- Shranjevanje dokumentov omejimo na uporabnike, ki ustrezajo izbranim kriterijem kot so članstvo v skupinah, vrsta uporabniškega računa, itd.
- Ob shranjevanju obstoječih dokumentov le-to dovolimo samo uporabniku, ki je avtor dokumenta.

Poglavje 5

Aplikacija za shranjevanje dnevniških zapisov v formatu Syslog

V nadaljevanju je predstavljen praktični del diplomskega dela. Za podrobnejšo seznanitev z delovanjem strežnika CouchDB sem izdelal enostavno CouchApp aplikacijo, ki se imenuje BoxSpring (v nadaljevanju aplikacija) in je dosegljiva na naslovu `https://github.com/JanezStupar/BoxSpring`. Aplikacija je licencirana pod licenceo Apache 2.0 [17].

Z izdelavo aplikacije sem želel udejanjiti dva cilja. Tako aplikacija predstavlja zametek dveh projektov h katerima se bom v prihodnosti še vračal. Prvi cilj je izdelava odprtokodnega sistema za zbiranje, pregledovanje in analizo dnevniških zapisov. Danes so po eni strani informacijski sistemi sestavljeni iz množice komponent, za katere praviloma velja, da njihovi dnevniki niso enotne narave, predvsem pa so razdrobljeni širom sistema. Po drugi strani se iz različnih razlogov pojavljajo potrebe po analizi in hrambi dnevniških zapisov za potrebe analize delovanja in spremljanja dostopa do podatkov. To velja predvsem za sisteme, v katerih hranimo občutljive podatke. Glede na to, da sem se pri svojem delu že srečeval s potrebo po takšnem orodju in pri tem naletel na veliko praznino na tržišču, sem se odločil, da želim izdelati lasten odprtokodni sistem za zbiranje in analitiko dnevniških zapisov. Pri tem Apache CouchDB s svojimi funkcijami ponuja idealno osnovo za izdelavo takšne aplikacije.

Drugi cilj, ki se je izrazil skozi samo delo na aplikaciji, je poskus izdelave ogrodja za hitrejše in lažje izdelovanje aplikacij določenega tipa. Pri tem mislim na aplikacije, ki so zgrajene okrog delno strukturiranih podatkov in niso

zahtevne s stališča uporabniškega vmesnika, si pa želimo hitro in enostavno prilagodljivost dostopa do podatkov. S takšnim orodjem po eni strani razvijalcu omejimo svobodo izražanja, po drugi strani pa lahko razvijalcu pohitrimo in olajšamo delo pri izdelavi podpornih aplikacij. Pri izdelavi aplikacije sem se ukvarjal z eksperimentiranjem na temo nastavljivega uporabniškega vmesnika. Bodoči načrti na to temo so popisani v poglavju ideje za izboljšave.

5.1 Funkcije aplikacije

Aplikacija je enostavno, na spletnih tehnologijah temelječe orodje za zbiranje, pregledovanje in analizo dnevniških zapisov. Njene funkcije obsegajo uvoz dnevniških zapisov strežnika CouchDB, podporo za dnevniške zapise v formatu Syslog ter nastavljiv spletni uporabniški vmesnik za pregledovanje vsebine podatkovne zbirke.

5.1.1 Uvoz podatkov iz dnevnika dogodkov

Podatke iz dnevnika dnevniških zapisov v podatkovno zbirko uvažamo z uporabo Python [34] skripte, ki za parametre prejme pot do CouchDB dnevnika, število dnevniških zapisov za prenos in naslov podatkovne zbirke CouchDB. Skripta prebere podano število dnevniških zapisov ter jih v Syslog formatu zapiše v podatkovno zbirko.

Za komunikacijo skripte s podatkovno zbirko CouchDB sem uporabil CouchDB gonilnik za Python, imenovan Couchdbkit [21].

5.1.2 Strežniški del

Strežniški del aplikacije zgolj streže podatke spletnemu odjemalcu, in ne vsebuje aplikacijske logike. Strežniški del poleg strežbe multimedijskih vsebin nudi še dva spletna servisa, ki spletnemu odjemalcu strežeta potrebne podatke:

- Seznam config odjemalcu v JSON formatu vrača podatke, potrebne za generiranje uporabniškega vmesnika. Pregled po imenu config je namenjen uporabi s tem seznamom.
- Seznam gridData spletnemu odjemalcu v JSON formatu vrača podatke o dnevniških zapisih, skrbi za ustrezno formatiranje in ostranjevanje podatkov. Podatki so formatirani za uporabo v jQGrid komponenti za prikazovanje tabelaričnih podatkov [27]. S tem seznamom se uporabljajo pregledi: `byhost`, `bypriority` in `logfiles`.

5.1.3 Spletni odjemalec

Spletni odjemalec je namenjen pregledovanju podatkov v podatkovni zbirki. Odjemalec ponuja sledeče funkcije:

- Pregled vseh dnevniških zapisov - funkcija je namenjena prikazu in pregledu celotne vsebine podatkovne zbirke.
- Pregled dnevniških zapisov po vrsti oz. prioriteti - funkcija omogoča pregled podmnožice zapisov glede na prioriteto, tako imamo možnost prikazovati samo napake, informativna obvestila, itd.
- Pregled dnevniških zapisov glede na vir - funkcija nam na podlagi enega od izbranih vnosov prikaže vse dnevniške zapise za izbrani izvor.
- Pregled posamičnih dnevniški zapisov.

Poleg omenjenih vsebinskih funkcij spletni odjemalec omogoča tudi nastavitev uporabniškega vmesnika, to pomeni, da lahko z uporabo nastavitvenih dokumentov spreminjamo uporabniški vmesnik oz. mu dodajamo nove funkcije.

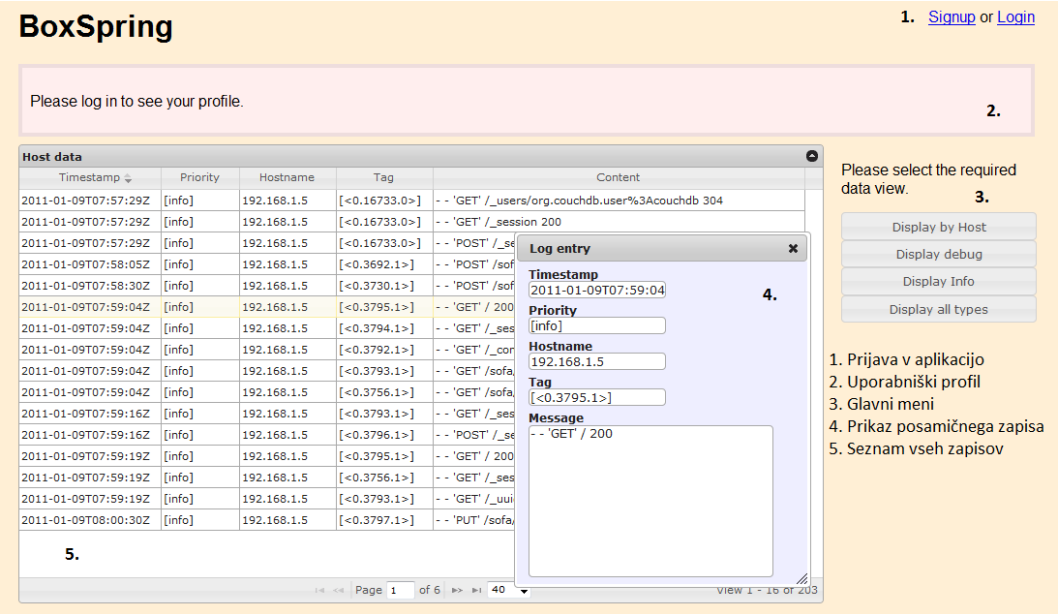
5.2 Zgradba aplikacije

Aplikacija je spletna aplikacija, zgrajena po zgledu CouchApp aplikacij. Glavni gradniki aplikacije so spletni servisi na strežniku, nastavitveni dokumenti za krmiljenje uporabniškega vmesnika, podatkovni dokumenti, HTML predloga in napravice (ang. widgets), iz katerih zgradimo uporabniški vmesnik. Na sliki 5.1 vidimo uporabniški vmesnik aplikacije BoxSpring.

V nadaljevanju so predstavljeni posamični gradniki.

5.2.1 Nastavitveni dokumenti

Z uporabo nastavitvenih dokumentov iz izvorne kode izoliramo različne parametre pogosto uporabljenih gradnikov, kar ob primerni uporabi olajša vzdrževanje in nadgrajevanje kode. V tej verziji aplikacije uporabljamo dva tipa nastavitvenih dokumentov. Z enim tipom krmilimo vsebino in obnašanje glavnega menija, z drugim pa krmilimo prikaz rezultatov. Nastavitveni dokumenti sicer po svoji naravi spadajo med podatkovne dokumente, ločeno pa jih obravnavamo zaradi njihove vloge v sistemu.



Slika 5.1: Uporabniški vmesnik aplikacije BoxSpring.

Nastavitve glavnega menija

Nastavitvene dokumente glavnega menija uporablja napravica, ki skrbi za ustvarjanje in upravljanje akcij v meniju. Na sliki 5.2 vidimo primer dokumenta, ki definira vnos v glavnem menuju.

Nastavitveni dokument mora imeti sledeča polja:

- `doc_type`: služi osnovnemu filtriranju dokumentov po tipu. Z uporabo tega polja v pregledih prikazujemo samo dokumente željenega tipa.
- `config_type`: definira tip nastavitvenega dokumenta. Z uporabo tega polja razločujemo tipe nastavitvenih dokumentov. V našem primeru je tip `menuitem`, kar pomeni da dokument predstavlja posamičen vnos/gumb v meniju.
- `config_name`: naziv nastavitvenega dokumenta. Z uporabo tega polja ločujemo nastavitve posamičnih scenarijev uporabe napravice.
- `config_value`: vsebuje dejanske nastavitve napravice:
  - `menuitemname`: nastavitev uporabljamo za HTML ID značko

```
{
    "_id": "89d06b608f119808fea7ed0f8c00b771",
    "_rev": "10-868dc579bf204cb187166933e9232a42",
    "doc_type": "config",
    "config_type": "menuitem",
    "config_name": "syslog",
    "config_value": {
        "menuitemname": "All",
        "menuitemlink": "",
        "menuitemtext": "Display all types",
        "type": "grid"
    }
}
```

Slika 5.2: Nastavitveni dokument glavnega menija.

gumba in kot osnovo za identifikacijo pripadajoče nastavitve za prikaz seznama zapisov.

- `menuitemlink`: v primeru, da bi gumb prožil povezavo URL, bi v tem polju navedli ciljni naslov.
- `menuitemtext`: besedilo, ki se izpiše na gumbu,
- `type`: ta nastavitev nam pove kakšnega tipa je ciljna napravica, v našem primeru gre za seznam vnosov (ang. grid).

Nastavitve prikazovalnika zapisov

Delovanje napravice za prikazovanje zapisov krmilimo z uporabo nastavitvenih dokumentov prikazovalnika zapisov. V nastavitvenem dokumentu shranjujemo informacijo o zgradbi in nastavljivih vidikih delovanja prikazovalnika. Na sliki 5.3 lahko vidimo primer nastavitve za prikazovalnik.

Nastavitveni dokument prikazovalnika v polju `config_value` vsebuje sledeče lastnosti, opis ostalih polj je na voljo pri opisu nastavitev glavnega menija:

- `colnames`: nazivi podatkovnih stolpcev, kot se prikazujejo v tabeli na uporabniškem vmesniku.
- `colmodel`: definicija stolpcev za prikaz podatkov, vsebuje podatke o imenu stolpca, sortiranju, širini, formatu zapisa podatkov, itd.
- `startkey`: CouchDB preged na podlagi tega parametra določi prvi vnos množice rezultatov.

```
{
    "_id": "89d06b608f119808fea7ed0f8c009645",
    "_rev": "16-e058ef368616667c05c28039b7908a42",
    "config_name": "Info",
    "config_type": "jqgrid",
    "doc_type": "config",
    "config_value": {
        "colnames": [
            "Timestamp",
            "Priority"
        ],
        "colmodel": [
            {
                "name": "tstamp",
                "index": "tstamp",
                "width": 150
            },
            {
                "name": "pri",
                "width": 80,
                "sortable": false
            }
        ],
        "startkey": "[\"[info]\"]",
        "endkey": "[\"[info]\",{}]",
        "url": "http://syslog.mycouch/list/gridData/bypriority/"
    }
}
```

Slika 5.3: Nastavitveni dokument prikazovalnika.

- `endkey`: CouchDB pregeld na podlagi tega parametra določi zadnji vnos množice rezultatov.

- `url`: naslov servisa na katerem se nahajajo podatki.

5.2.2 Podatkovni dokumenti

V podatkovnih dokumentih hranimo vsebino aplikacije. Vsak podatkovni dokument vsebuje po en dnevniški zapis v formatu Syslog. Na sliki 5.4 je primer takšnega dokumenta.

Vsaka komponenta formata Syslog *(prioriteta, časovni žig, izvor, proces in sporočilo)* je zapisana v ločenem polju. Tak pristop je uporabljen zato, da si

```
{
    "_id": "bf30cd74ec3f7657ed5396d4d738eb0a",
    "_rev": "1-f1f121a4ab7c5c4b0009043a4b81b976",
    "doc_type": "CouchSyslogDoc",
    "format": "syslog",
    "timestamp": "2011-01-09T08:00:30Z",
    "hostname": "local",
    "priority": "[debug]",
    "tag": "[<0.3797.1>]",
    "content": "OAuth Params: []",
    "type": "log"
}
```

Slika 5.4: Primer dokumenta za zapisovanje dnevniških zapisov.

olajšamo indeksiranje dokumentov. V nadaljevanju sledi popis polj podatkovnega dokumenta:

- `doc_type`: služi osnovnemu filtriranju dokumentov po tipu. Z uporabo tega polja v pregledih prikazujemo samo dokumente željenega tipa.
- `format`: polje nosi informacijo o formatu zapisov. Praviloma so vsi zapisi v formatu Syslog, vendar s tem dopuščamo bodočo širitev aplikacije na druge formate.
- `timestamp`: časovni žig, nosi informacijo o tem, kdaj je zapis nastal na izvoru.
- `hostname`: povzročitelj dogodka, to polje praviloma vsebuje naslov IP oz. naslov v drugem formatu.
- `priority`: prioriteta dogodka nam pove kako resen je dogodek oz. napaka.
- `tag`: oznaka procesa, ki je zabeležil oz. sporočil dogodek. Sem zapisujemo naziv oz. številko procesa, ki je povzročil zapis v dnevnik.
- `content`: telo sporočila, vsebina je odvisna od vloge in stanja izvora.
- `type`: tip je metapodatek, ki nosi informacijo o izvoru dnevniškega zapisa.

5.2.3 Gradniki uporabniškega vmesnika - napravice

V aplikaciji je celoten uporabniški vmesnik zgrajen na podlagi dogodkovno krmiljenih napravic (ang. event driven widgets). Vsaka napravica je samostojen modul, ki upravlja določen vidik uporabniškega vmesnika oz. aplikacijske logike.

Za osnovo napravic sem uporabili ogrodje Evently. Ogrodje Evently poenostavlja registracijo, proženje in povezovanje dogodkov. Dogodki so lahko v brskalnik vgrajeni dogodki (*npr. click, onmouseover, onmouseout,itd.*), lahko so povsem uporabniško definirani, obstaja pa še tretja kategorija t.i. vgrajeni Evently dogodki. Vgrajeni Evently dogodki omogočajo hitro in enotno izvedbo asinhronih AJAX operacij. S svojnim naborom funkcij pokrivajo večino potreb po asinhronih operacijah na enoten način. Na sliki 5.5 vidimo primer JSON objekta, kakršnega ogrodju Evently podamo za definicijo dogodkov. Kot komentar velja omeniti podatek, da ima vsak dogodek lahko poljubno število poddogodkov.

```
{
  _init: {
    mustache: "...",
    async: "...",
    data: "..."
  },
  mustache: "...",
  async: "...",
  data: "",
  selectors: {
    'a[href=#...]': {
      mustache: "...",
      click: "..."
    }
  }
}
```

Slika 5.5: Primer strukture Evently dogodka.

V nadaljevanju sledijo kratki opisi vgrajenih evently dogodkov ter opis njihovega delovanja oz. uporabe:

- `_init`: dogodek `_init` si velja predstavljati kot konstruktor dogodka in je vedno prva operacija oz. dogodek, ki se izvrši ob klicu dogodka.
- `async`: ta dogodek je povratni klic (ang. callback) za vršenje AJAX zahtevkov na strežnik.

- `data`: ta dogodek je povratni klic, ki se izvrši po `async` dogodku in služi pripravi podatkov za dogodek mustache,
- `after`: je dogodek, ki se izvrši na koncu, ko so vsi ostali dogodki že zaključili svoje izvajanje.
- `selectors`: je dogodek, v katerem nanizamo jQuery selektorje s pomočjo katerih Evently poišče ustrezne DOM elemente, potem pa jim pripne navedene dogodke oz. uporavljalce dogodkov.
- `mustache`: dogodek služi podobnemu namenu kot Mustache predloge v CouchApp ogrodju. V `mustache` dogodku hranimo predlogo HTML za posamičen dogodek. S pomočjo `mustache` dogodka tako po eni strani ločimo HTML od kode, po drugi strani pa je HTML koda še vedno lahko del same napravice.

Evently napravice se nahajajo v CouchApp aplikacijskem dokumentu, v konkretnem primeru `_design/syslog`, v polju `vendor`, kot lastnosti JSON objekta `couchapp.evently`.

Za potrebe aplikacije sem razvil dve napravici. Ena se imenuje `mainmenu` in je namenjena upravljanju z menijem, druga pa se imenuje `maingrid` in skrbi za izrisovanje prikazovalnika zapisov. Sledi podrobnejši opis obeh napravic.

Napravica mainmenu

Naloga napravice `mainmenu` je, da s strežnika pridobi podatke nastavitvenih dokumentov za generiranje glavnega menija in na njihovi osnovi izdela gumbe v meniju. Na sliki 5.6 vidimo zgradbo oz. podpis napravice `mainmenu`.

```
1 {
2   _init: "function(){...}",
3   createButton: {
4     after: "function () {...}",
5     data: "function (e,rows) {...}",
6     mustache: "...",
7     selectors: "..."
8   }
9 }
```

Slika 5.6: Podpis napravice mainmenu.

Dogodki v sklopu napravice `mainmenu` opravljajo sledeče vloge:

- `_init`: ta dogodek izvede AJAX klic servisa za strežbo nastavitev in zahteva nastavitve za vse elemente menija. Nato pa s proženjem dogodka `createButton` za vsako nastavitev ustvari vnos v meniju.
- `createButton`: je dogodek, ki vsebuje poddogodke:
  - `data`: na podlagi podatkov, dobljenih od dogodka `_init`, pripravi JSON objekt, ki ga Evently potem posreduje dogodku `mustache`.
  - `mustache`: vsebuje HTML fragment, v katerem ogrodje Mustache začasne vrednosti zamenja z vrednostmi iz objekta, ki smo ga ustvarili v dogodku `data`. V konkretnem primeru se v ogrodju nahaja predloga za neurejen seznam, v katerem vsak vnos predstavlja eno povezavo oz. akcijo. Podatke iz dogodka `data` uporabimo za vrednost id HTML elementa in besedilo gumba.
  - `selectors`: vsaki akciji oz. gumbu v meniju registriramo dogodek `click`, ki bo ob izvedbi sprožil generiranje prikazovalnika.
  - `after`: vse do tega trenutka običajne HTML povezave v okviru glavnega menija pretvori v jQueryUI napravice tipa gumb. S tem HTML značke `<a></a>` spremenimo v lične gumbe.

Napravica maingrid

Naloga napravice `maingrid` je, da skrbi za izrisovanje prikazovalnika zapisov, preko katerega uporabniki dostopajo do željenih zapisov. Na sliki 5.7 vidimo zgradbo oz. podpis napravice `maingrid`.

```
1 {
2   _init: "function(e,gridname){...}",
3   drawGrid: {
4     async: "function(callback) {...}",
5     data: "function(gridname,result)",
6     mustache: "...",
7     after: "..."
8     }
9 }
```

Slika 5.7: Podpis napravice maingrid.

Dogodki v sklopu napravice `maingrid` opravljajo sledeče vloge:

- `_init`: dogodek izvede inicializacijo napravice.

- `drawGrid` je dogodek, ki vsebuje poddogodke:
  - `async`: izvede klic servisa z nastavitvami in pridobi nastavitvene podatke za željeni tip prikazovalnika, te pa posreduje preostalim dogodkom.
  - `data`: podatke pridobljene iz dogodka `async` pretvori v JSON objekt in ga posreduje dogodku mustache.
  - `mustache`: vsebuje HTML predlogo, ki združena s podatki iz dogodka `data` služi kot osnova za prikazovalnik.
  - `after`: na podlagi rezultatov ostalih dogodkov ustvari prikazovalnik, ki je izveden s pomočjo jQGrid komponente za prikaz tabelaričnih podatkov.

5.3 Primeri uporabe aplikacije

Na tem mestu bi se nekoliko ustavili še pri scenarijih uporabe aplikacije BoxSpring. Pisanje in ustvarjanje informacijskih sistemov je že samo po sebi lahko razvijalcu v zadovoljstvo in veselje. Vendar je ponavadi zadovoljstvo še toliko večje, če so naše storitve koristne in služijo praktičnim namenom.

Aplikacija BoxSpring je, kot že omenjeno, namenjena zbiranju, hrambi in analizi dnevniških zapisov. Primarno je namenjena za okolja, kjer se razvijalci in vzdrževalci srečujejo z veliko raznolikostjo ekosistema, za delovanje katerega so zadolženi. To pomeni bodisi sisteme z velikim številom strežnikov, bodisi porazdeljene sisteme, v katerih želi izvajalec zagotavljati visoko stopnjo kakovosti storitve. V takšnih sistemih je iskanje zapisov o napakah lahko izjemno zamudno opravilo. Zaradi tega se po mojem mnenju pogosto dogaja, da napake, ki niso kritične, ostanejo spregledane, dokler njihov vpliv ne postane rušilen. Po drugi strani, lahko tudi kritične napake ostanejo neopažene bistveno dlje, kot bi bilo željeno ali potrebno.

Z aplikacijo, kot je BoxSpring, lahko zagotovimo, da se dnevniški podatki iz našega informacijskega sistema stekajo v enoten, standardiziran sistem, od koder jih potem lahko z uporabo vgrajenih mehanizmov bodisi zbiramo na enem mestu za hrambo in obdelavo, bodisi jih po omrežju med seboj povezanih BoxSpring aplikacij usmerjamo k odgovornim osebam in jim na ta način olajšamo upravljanje in nadziranje informacijskega sistema. Pri tem nismo omejeni zgolj na uporabo delovne postaje za dostop do oddaljenega strežnika. Če predpostavimo, da imajo naši uporabniki CouchDB nameščen na svojih mobilnih telefonih, potem lahko z uporabo replikacije najnujnejše zahtevke

posredujemo neposredno na mobilne naprave odgovornih oseb, s tem pa lahko zelo zmanjšamo odzivne čase in izboljšamo kvaliteto storitev.

Drugi večji scenarij je uporaba aplikacije za hrambo in upravljanje zapisov, povezanih z varovanjem podatkov v informacijskih sistemih. V skladu z veljavno zakonodajo, kot sta Zakon o varovanju dokumentarnega in arhivskega gradiva [11] in Zakon o varstvu osebnih podatkov [12], je potrebno v informacijskih sistemih, povezanih s hrambo arhivskih podatkov oz. osebnih podatkov beležiti, po potrebi pa tudi prikazovati podatke o tem, kdaj je kdo dostopal do katerih podatkov. Ker je v sodobnem informacijskem sistemu, ki je zgrajen iz mnogo gradnikov, ponavadi zelo zamudno, včasih celo nemogoče pridobiti in restavrirati revizijske sledi, velja razmisliti tudi o uporabi specializiranih orodij za hrambo in obdelavo dnevniških zapisov.

5.4 Ideje za izboljšave

Ker je pri vsaki dejavnosti vedno prostor za izboljšave, se je v ambicijah potrebno umiriti in najti pravo ravnovesje med željenim in v danih razmerah dosegljivim oz. uresničljivim. Velja pa vedno voditi seznam potencialnih izboljšav in pripomb uporabnikov, saj si s takšnim pristopom lahko močno olajšamo načrtovanje in izvedbo podočih dograditev. Tako tudi jaz v nadaljevanju ponujam par konceptov in idej za nadaljne dograditve.

Povsem na vrhu seznama izboljšav se nahaja izdelava uporabniškega vmesnika, prilagojenega za uporabno na pametnih telefonih, s katerim bi uporabnikom močno izboljšali uporabniško izkušnjo ter se še bolj približali željenemu cilju. Takšen uporabniški vmesnik bi omogočal enostavno uporabo najosnovnejših funkcij sistema. Mobilni vmesnik namreč ni namenjen običajni uporabi sistema temveč si ga predstavljam bolj kot orodje za upravljanje kritičnih informacij, s katerim bi bili uporabniki preko mobilnega telefona lahko obveščeni o najnujnejših dogodkih v sistemu. V paket dodatnih funkcij za podporo mobilnim telefonom bi dodal še vmesnik za upravljanje replikacij, s katerim bi lahko krmilili prenos podatkov med replikami zbirke.

Za lažje iskanje podatkov bi veljalo v aplikacijo vključiti podporo za iskanje po polnem besedilu. Z iskanjem podatkov je povezana še funkcija izvajanja korelacije med dnevniškimi zapisi iz različnih sistemov, s pomočjo katere bi uporabnikom omogočili spremljanje akterjev v sistemu ne glede na to, da različni sistemi nimajo vseh podatkov o identiteti akterjev. Na primer: spletni strežnik ima informacijo o odjemalčevem naslovu IP, aplikacijskemu strežniku je na voljo njegovo uporabniško ime, naslov IP ter identifikator seje na baznem

strežniku, medtem ko bazni strežnik pozna zgolj uporabnikov identifikator seje.

Kot pomembno potencialno izboljšavo bi izpostavil še podporo večim formatom dnevniških zapisov. Aplikacija sicer omogoča, da vanjo katerikoli sistem pošilja podatke z enostavno uporabo vmesnika HTTP, prav tako so za CouchDB že razviti mnogi gonilniki, ki olajšajo delo z vmesnikom HTTP. Vendar je po mojem mnenju za pridobivanje novih uporabnikov zelo pomembno, da je uvedba uporabe aplikacije čim enostavnejša. Zato bi veljalo razmisliti tudi o razvoju gonilnikov za nekatera pogosto uporabljana orodja za upravljanje z napakami, kot so npr. log4j, log4js, log.net, itd.

Poglavje 6

Zaključek

Inženirji se vsakodnevno srečujemo z najrazličnejšimi izzivi, pri iskanju rešitev zanje pa smo enkrat bolj, drugič pa manj uspešni oz. iznajdljivi. Pri iskanju boljših rešitev se prepogosto ukvarjamo samo s tistim, kar nam je domače in poznano, pri tem pa spregledamo drugačne, nepoznane in nepriljubljene pristope in v iskanju popolnosti gradimo rešitve, ki niso ne uspešne, ne učinkovite ter praviloma niso elegantne.

Menim, da sodobni distribuirani informacijski sistemi ne morejo biti monokulture, saj trgi od informacijske industrije pričakujejo in zahtevajo čedalje kompleksnejše rešitve, pri katerih snovanju enotirno razmišljanje in delovanje predstavlja huda tveganja in nevarnosti. S tem v mislih mora imeti sodoben informacijski obrtnik v svoji orodjarni na voljo širok spekter specializiranih orodij, s spretno uporabo katerih lahko zadovolji potrebe strank in si pridobi nove priložnosti.

V okviru izdelave tega diplomskega dela sem se temeljito seznanil z delovanjem in uporabo podatkovnega strežnika CouchDB. Menim, da CouchDB ponuja prepričljiv nabor funkcij in možnosti, saj je med drugim zaradi odprtosti podatkovnega modela izrazito primeren za hitro prototipiranje informacijskih sistemov. Njegovi vgrajeni replikacijski in varnostni mehanizmi omogočajo njegovo uporabo v implementaciji distribuirane podatkovne infrastrukure za informacijske sisteme, leni (ang. lazy) replikacijski mehanizmi pa zagotavljajo nemoteno delovanje informacijskih sistemov v primerih, ko so nekatera vozlišča občasno ali pa dalj časa namerno ali nenamerno komunikacijsko odrezana od preostanka sistema. Nenazadnje pa CouchApps aplikacije ponujajo revolucionaren pristop k širjenju in izmenjavi informacij.

Končni rezultat diplomske naloge je splošen pregled funkcij in lastnosti podatkovnega strežnika Apache CouchDB. Diplomsko delo sem pisal z name-

nom, da služi kot informativni pregled nekaterih najzanimivejših in najpomembnejših konceptov, lastnosti in informacij o delovanju in uporabi Apache CouchDB.

Slike

Literatura

[1] J.C. Anderson, J. Lehnardt, N. Slater, *CouchDB the definitive guide*, Sebastopol: O'Reilly Media, 2010, pogl. 22.

[2] P. Buneman, "Semistructured data," v zborniku *Proceedings of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems*, Tucson, Arizona, maj 1997, strani 117-121.

[3] E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, št. 13, zv. 6, str. 377-387, 1970.

[4] R.T. Fielding, J. Gettys, J.C. Mogul, H.F. Nielsen, L. Masinter, P.J. Leach, T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1" *IETF Network Working Group, RFC2616*, jun. 1999. Dostopno na: http://www.ietf.org/rfc/rfc2616.txt

[5] R.T Fielding, R.N. Taylor, "Principled Design of the Modern Web Architecture," *ACM Transactions on Internet Technology (TOIT)*, New York: ACM, št. 2, zv. 2, str. 115-150, 2002.

[6] D. Flangan, *Javascript the definitive guide*, Sebastopol: O'Reilly Media, 2006,pogl. 1.

[7] J. Gray, P. Helland, P. O'Neil, D. Shasha, "The dangers of replication and a solution," v zborniku *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, Montreal, Quebec, Canada, jun. 1996, str. 173-182.

[8] E. Hammer-Lahav, "The OAuth 1.0 Protocol" *IETF Network Working Group, RFC5849*, April 2010. Dostopno na: http://tools.ietf.org/html/rfc5849

[9] D. Kristol, L. Montulli, "HTTP State Management Mechanism" *IETF Network Working Group, RFC2965*, okt. 2000. Dostopno na: http://tools.ietf.org/html/rfc2965

[10] Z. Krolikowski, T. Morzy, "Database Systems: from File Systems to Modern Database Systems" *Handbook on data management in information systems*, New York: Springer, 2003, str. 22-25.

[11] ZVDAGA, *Uradni list Republike Slovenije*, Št. 30/2006

[12] ZVOP-1, *Uradni list Republike Slovenije*, Št. 86/2004

[13] (2010) J.C. Anderson. Dostopno na: https://github.com/jchris/sofa

[14] (2010) J.C. Anderson. Dostopno na: https://github.com/jchris/sofa/blob/master/lists/index.js

[15] (2010) J.C. Anderson. Dostopno na: https://github.com/jchris/sofa/blob/master/templates/index.html

[16] (2011) Apache Foundation. Dostopno na: http://www.apache.org/

[17] (2004) Apache Foundation. Dostopno na: http://www.apache.org/licenses/LICENSE-2.0.html

[18] (2008) Apache Foundation. Dostopno na: http://couchdb.apache.org

[19] (2011) Apache Foundation. Dostopno na: http://wiki.apache.org/couchdb/Document_Update_Validation

[20] (2011) Apache Foundation. Dostopno na: http://wiki.apache.org/couchdb/Security_Features_Overview

[21] (2011) Couchdbkit, CouchDB Python Database Driver. Dostopno na: http://couchdbkit.org/

[22] (2009) K. Dangoor, CommonJS. Dostopno na: http://www.commonjs.org/specs

[23] (2011) Ericsson Computer Science Laboratory. Dostopno na: http://www.erlang.org/

[24] (2011) Erlang Computer Science Laboratory. Dostopno na:
http://www.erlang.org/doc/apps/inets/http_server.html

[25] (2005) J.J. Garret, Adaptive Path. Dostopno na:
http://www.adaptivepath.com/ideas/essays/archives/000385.php

[26] (2011) Javascript Object Notation. Dostopno na:
http://www.json.org

[27] (2011) jQGrid, jQuery plugin. Dostopno na:
http://www.trirand.com/blog/

[28] (2010) J. Lehnardt, Github. Dostopno na:
https://github.com/janl/mustache.js

[29] (1999) C. Mohan, IBM Almaden Research Center. Dostopno na:
http://www.almaden.ibm.com/u/mohan/domino_sigmod99.pdf

[30] (2011) Mozzila Foundation. Dostopno na:
http://www.mozilla.org/js/spidermonkey/

[31] (2011) NoSQL. Dostopno na:
http://nosql-database.org

[32] (2006) NIST Guide to Computer Security Log Management. Dostopno na:
http://csrc.nist.gov/publications/nistpubs/80092/SP80092.pdf

[33] (2005) Object Management Group, UML 2.0 Specification. Dostopno na:
http://www.omg.org/spec/UML/2.0

[34] (2011) Python programming language. Dostopno na:
http://python.org/

[35] (2011) Wikimedia Foundation. Dostopno na:
http://en.wikipedia.org/wiki/Database_management_system

[36] (2011) Wikimedia Foundation. Dostopno na:
http://en.wikipedia.org/wiki/Documentoriented_database

[37] (2011) Wikimedia Foundation. Dostopno na:
http://en.wikipedia.org/wiki/Eventual_consistency

[38] (2011) Wikimedia Foundation. Dostopno na:
http://en.wikipedia.org/wiki/Pure_function

[39] (2011) Wikimedia Foundation. Dostopno na:
http://sl.wikipedia.org/wiki/HTML