Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/12BE35Mijatovic

NODE.JS BAZIRANI DISTRIBUIRANI SISTEM ZA DUGOTRAJNO ČUVANJE DIGITALNIH DOKUMENATA

NODE.JS BASED DISTRIBUTED SYSTEM FOR DIGITAL DOCUMENTS LONG-TERM PRESERVATION

Marko Mijatović, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljena arhitektura i implementacije sistema za pravljenje kopija digiralnih dokumenata. Sistem prati promene dokumenata i dodavanje novih. Šalje njihove kopije na čuvanje na više lokacija.*

Ključne reči: *čuvanje dokumenata, bekap*

Abstract – *The paper presents the architecture and implementation of a system for making copies of digital documents. The system monitors changing documents and adding new ones. Sends their copies for keeping to multiple locations.*

Keywords: *document storage, backup*

1. UVOD

Tokom korišćenja računara generiše se veliki broj dokumenata različitog formata. Većina dokumenata korisna je samo na kratak vremenski period, ali za neke će biti potrebno trajno čuvanje. Kako bi se osiguralo da vremenom ostanu pouzdani i korisni, neophodno je pripremiti dobro osmišljen i dokumentovan plan za njihovo čuvanje. Kada se hard disk ili drugi skladišni medijum pokvari, a nije napravljena rezervna kopija, tada je već prekasno za bilo kakvo rešavanje problema. Zato je potrebno razmišljati unapred, jer nije pitanje kada će hard disk prestati sa radov, već kada će se to desiti. Postoji i slučaj kada je fajl oštećen, a korisnik ne zna i misli da je fajl ispravan. Kada se nad takvim fajlom napravi rezervna kopija i ona je za korisnika neupotrebljiva kao i originalni fajl. Kada korisnik sazna da su i fajl i kopija oštećeni, nema načina da povrati fajl. Bitno je proveravati važne fajlove na vreme i utvrditi njihovu ispravnost kako bi se pravila njihova rezervna kopija dok su u ispravnom stanju.

Tema ovog rada jeste kreiranje sistema koji bi periodično obavljao posao provere digitalnih dokumenata i njihovo skladištenje kod drugih izabranih korisnika sistema. Provera digitalnih dokumenata podrazumeva detektovanje da li je došlo do izmene na dokumentu ili do njegovog oštećenja ili gubitka.

Dokumente koji su izmenjeni potrebno je ponovo dostaviti korisnicima na čuvanje kako bi se u sistemu uvek nalazile poslednje verzije dokumenata.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivanović, vanr.prof.

2. TEORIJSKE OSNOVE

Sistemi za bekap podataka uglavnom su projektovani kao distribuirani sistemi. Serveri, na koje korisnici postavljaju i na kojima se čuvaju njihovi fajlovi, su distribuirani. Bekap sistemi rade sa digitalnim dokumentima, pa će u ovom poglavlju biti pojašnjen pojam digitalnih dokumenata, uz osljanjanje na knjigu [1].

2.1. Digitalni dokumenti

Preteča digitalnih dokumenata su papirni dokumenti. Kada gledamo iz ugla današnje tehnologije i njenog napretka, oni izgledaju prilično zastarelo. Sa druge strane, oni još uvek imaju svoje prednosti koje se ogledaju u tome što nije potrebno posedovati skup uređaj sa programima, koji se takođe plaćaju, da bi se dokument kreirao i koristio.

Informacije snimljene za dalje korišćenje, pregled i obradu, na način koji zahteva računar ili drugi elektronski uređaj opisane su kao digitalni dokumenti. Razvoj interneta i masovno korišćenje računara, gde su računari međusobno povezani radi lakše komunikacije, dovelo je do situacije gde se na dnevnom nivou kreira ili izmeni velika količina digitalnih dokumenata.

2.2. Bekap podataka

Bekap [2] je mehanizam u sistemima koji su namenjeni za oporavak od katastrofe. Katastrofe koje dovode do gubitka podataka se neretko dešavaju i mi u većini slučajeva ne možemo da utičemo na njih. Možemo samo da sistem i podatke vratimo na stanje pre katastrofe i da ublažimo njene posledice. Bekap mehanizmom vrši se dupliranje, kopiranje podataka na sigurno mesto. Moguće je raditi bekap na dnevnom, nedeljnom ili mesečnom nivou. Potrebno je pravilno izabrati vremenski period uzevši u obzir količinu novih digitalnih dokumenata i izmenu postojećih. Optimalno rešenje je kombinovati više vremenskih perioda, bitne dokumente nad kojima se često vrše izmene bekapovati na dnevnom nivou a ostale na nedeljnom ili mesečnom.

2.3. Dugotrajno čuvanje podataka

Dugotrajno čuvanje podataka [3] predstavlja niz izazova, od tehničkih do društvenih i organizacionih. Tehnički izazvo je osigurati da podaci danas mogu preživeti dugotrajne promene u medijima za skladištenje, uređajima i formatima podataka. Dugotrajno čuvanje podataka ima tri glavna cilja:

- Sačuvati dokument,
- Osigurati pristupačnost,
- Očuvati razumljivost.

2.4. File Fixity

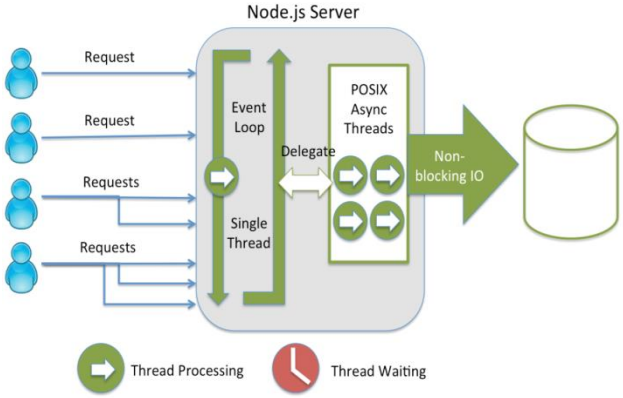
File Fixity [4] predstavlja sigurnost da je digitalni dokument ostao nepromenjen i ispravan. File Fixity se ne odnosi samo na datoteke, već na bilo koji digitalni objekat koji u sebi ima niz bitova, gde je potrebno da je taj niz ostao nepromenjen. Provera ispravnosti datoteka ne osigurava očuvanje digitalne datoteke. File Fixity Omogućava skladištu da identifikuje oštećene datoteke i da pokrene postupak njihove zamene sa ispravnom kopijom.

3. KORIŠĆENE TEHNOLOGIJE

Glavni zahtev sistema bio je da ima mogućnost izvršavanja na različitim operativnim sistemima. Za ispunjenje takvog zahteva bilo je potrebno izabrati tehnologije koje neće biti ograničene izvršavanjem na samo jednoj platformi. Iz tog razloga sistem je implementiran u Node.js i tehnologijama koje su zasnovane na JavaScript jeziku.

3.1. Node.js

Node.js [5] se koristi za razvoj veb aplikacija visokih preformansi. U osnovi je JavaScript okruženje u kojem se izvršavaju programi serverske strane u realnom vremenu. JavaScript je primarno korišćen na klijentskoj strani, gde su skripte napisane u JavaScriptu bile ugrađene u HTML stranice. Node.js omogučava da se JavaScript koristi za pisanje skripti na serverskoj strani. Takva mogućnost je značajna jer nije potrebno poznavanje dodatnih jezika za kompletan razboj veb aplikacije. Node.js ima jednu nit koja obrađuje pristigle zahteve prikazan na slici 1.



Slika 1. *Asinhroni način rada Node.js*

On koristi asinhroni i neblokirajući režim rada. Kod sinhronog izvršavanja, operacija može da krene izvršavanje samo ako je prethodna operacija završila svoj rad. Kod asingronog izvršavanja, operacije ne čekaju jedna drugu nego mogu istovremeno da se izvršavaju. Vreme izvršavanja svake operacije posebno ostaje isto, ali ukupno vreme izvršavanja svih operacija se drastično smanjuje ukoliko svaka može da se izvrši u zasebnoj niti.

3.2. MongoDB

Mongo baza [6] je nerelaciona baza podataka. NoSQL baze podata nisu tabelarne i podatke čuvaju drugačije od relacionih tabela. NoSQL baze imaju drugačije tipove u modelu na osnovu kojih skladište podatke. Glavni tipovi su:

- dokumenti,
- ključ/vrednost
- grafik,
- široke kolone

Pružaju fleksibilne šeme i lako se skaliraju sa velikim količinama podataka i velikim opterećenjem upitima nad bazom. NoSQL baze podataka mogu da čuvaju podatke o odnosima između čuvanih podataka.

Takve baze jednostavno taj podatak čuvaju drugačije u odnosu na relacione baze podataka, povezani podaci ne moraju da se dele u tabele.

NoSQL baze podataka omogućavaju programerima da čuvaju ogromne količine nestrukturiranih podataka, pružajući im veliku fleksibilnost.

4. SPECIFIKACIJA SISTEMA

Čuvanje dokumenata moguće je na različite načine. Većina popularnih sistema za čuvanje korisničkih podataka funkcionišu tako što poseduju velika skladišta podataka na različitim mestima. Korisnici ne znaju gde im se fajlovi nalaze i da li se svi fajlovi skladište na istom mestu. Ovaj sistem zamišljen je tako da nudi zanimljiv način čuvanja korisničkih dokumenata.

Korisnici u sistemu čuvaju kopije svojih dokumenata kod drugih korisnika. Jedna pristupna tačka koja preuzima i skladišti korisničke fajlove dok ih svi korisnici koji su garantovali njihovo čuvanje ne preuzmu. Kada svi korisnici preuzmu fajlove oni se brišu sa pristupne tačke radi oslobađanja njenog ograničenog skladišnog prostora.

Prednosti ovakve arhitekture sistema ogledaju se u tome što korisnici tačno znaju gde i kod koga se kopije njihovih dokumenata čuvaju. Nije potrebna velika količina prostora na jednom mestu.

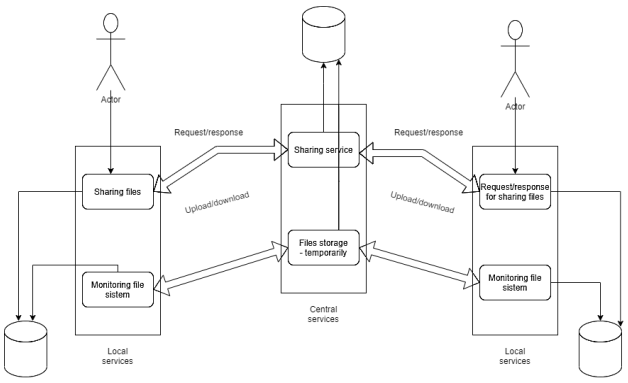
4.1. Arhitektura sistema

Sistem je realizovan kao veb aplikacija, zato je potrebno da su računari, koji će biti korisnici sistema, povezani na internet ili međusobno umreženi sa centralnim serverom. U slučaju da se za centralni server obezbedi javna IP (Internet Protocol) adresa, sistem bi mogao da funkcioniše tako što bi ostali korisnici uz povezivanje na internet mogli da mu pristupe sa bilo kog mesta.

Sistem je zamišljen da posle konfigurisanja može samostalno da radi bez interakcije sa korisnikom. U sistemu na svakoj korisničkoj mašini postoji lokalni servis koji obavlja glavni posao na korisničkim računarima i komunicira sa centralnim serverom. Skladištenje podataka obavlja se na svakom korisničkom računaru i na centralnom serveru. Korisnici imaju i veb aplikaciju pomoću koje je moguć uvid u fajlove koji se bekapuju i slanje zahteva drugim korisnicima za skladištenje kao i prihvatanje i odbijanje zahteva.

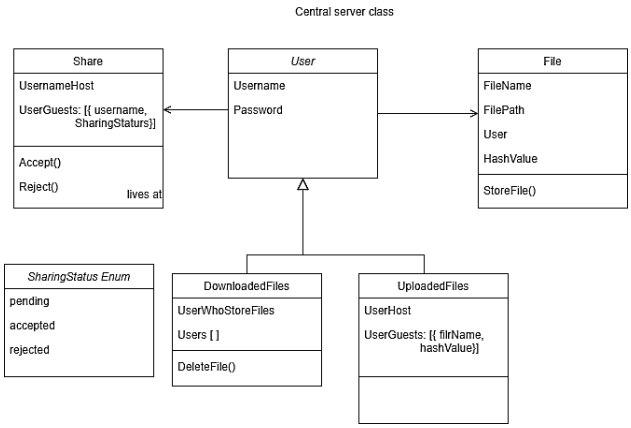
Veb aplikacija svaku komunikaciju sa centralnim serverom obavlja preko lokalnog servisa. Lokalni servis jednog korisnika nema direktnu komunikaciju sa drugim korisnicima. Lokalni servis prihvata svaki zahtev, beleži ga ako je potrebno i prosleđuje centralnom serveru. Na slici 2 može se videti arhitektura sistema.

Svi korisnici sistema imaju iste privilegije. Nije bilo dodatnih funkcionalnosti sistema koje bi zahtevale korisnike sa dodatnim privilegijama ili odvajanja korisnika na grupe.



Slika 2. *Arhitektura sistema*

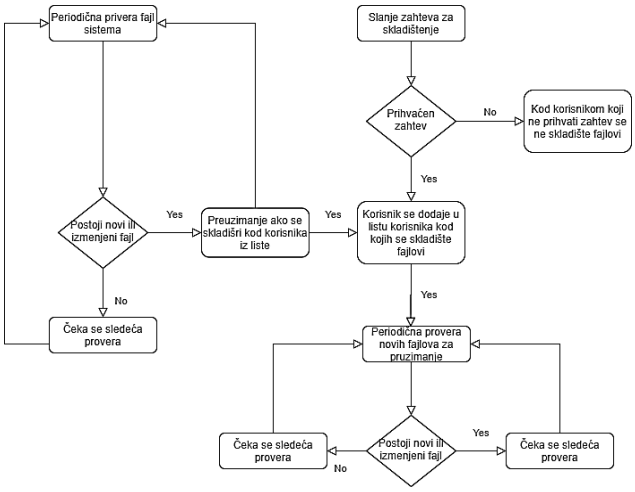
Dijagram klasa prikazan je na slici 3.



Slika 3. *Dijagram klasa sistema*

Sistem ovako zamišljen i implementiran ima tri bitne faze kroz koje prilazi tokom rada, koje se mogu videti i na slici 4:

- slanje i odgovor na zahtev za čuvanje dokumenata
- provera lokalnog direktorijuma i slanje fajla na čuvanje
- provera da li postoje fajlovi drugih korisnika koje treba preuzeti.



Slika 4. *Dijagram toka sistema*

5. IMPLEMENTACIJA

Sistem čine front-end i dve serverske aplikacije. Front-end je napisan u Angular okruženju a serverske strane u Node.js. Sve je razvijano u Visual Studio Code razvojnom okruženju, a za čuvanje podataka korišćena je Mongo baza podataka.

Front-end deo aplikacije sastoji se iz početne strane, pregleda korisnikovih fajlova, slanja zahteva za deljenje i pregleda notifikacija i odgovora na njih. Početna strana nudi izbor prelazaka na neku od ostale tri stranice. Na stranici za pregled fajlova korisnika, prikazani su fajlovi iz direktorijuma izabranog od strane korisnik da se prati njegovo stanje. JavaScript koji se pokreće u pretraživaču ima ograničen pristup lokalnom fajl sistemu, zato je implementacija zahtevala pomoć sa lokalnog servisa. Upućen je zahtev za dobijanje korisnikovih fajlova.

Lokalni servis poseduje otvoren API za front-end deo aplikacije. Korišćen je Express, postavljen body-parser i cors da bi zahtevi mogli da se upute iz Angular aplikacije. Lokalni server je otvoren na portu 52296.

Sledeći deo servisa je periodična provera izabranog direktorijuma. Za taj deo izabrana je node-schedule biblioteka. Podešeno je vreme na koliko će se vršiti provera i definisana je funkcija koja će obavljati proveru, listing 1.

```
schedule.scheduleJob('30 * * * * *',
async function(){
      var allFiles =
      monitoring.AllFiles(dir);
      await
      monitoring.isFileExist(dir,
allFiles);
});
```

Listing 1. *Pokretanje periodične provere*

Za svaki fajl u direktorijumu proverava se da li postoji sačuvan u Mongo bazi. Ako fajl ne postoji, čuvaju se njegov naziv, putanja do fajla, ime korisnika čiji je fajl i računa mu se hash vrednost prikazana u listingu 2.

```
const {hashElement} = require('hash-
folder');
const options = {algo: 'sha1'};
const fileHash = await
hashElement(filePath, options);
```

Listing 2. *Računanje hash vrednosti fajla*

Zatim se fajl šalje centralnom serveru koji ima zadatak da ga prosledi ostalim korsinicima i da ga izbriše iz svog skladišta. U slučaju da fajl postoji sačuvan u bazi, računa se njegova trenutna hash vrednost. Ona se upoređuje sa vrednošću koja je upisana u bazi. Ako se razlikuju te dve vrednosti znači da je došlo do izmene fajla i da je potrebno poslati novu verziju korisnicima na čuvanje. Sa novom verzijom fajla šalje se i njegova nova hash vrednost.

Na sličan način se preiodično obavlja i provera da li na centralnom serveru ima novih fajlova koje su postavili drugi korisnici, a potrebno ih je sačuvati. Na lokalnom serveru implementirana je klijentska strana FTP protokola. Pomoću njene implementacije vrši se slanje i pruzimanje fajlova sa centralnog servera. Prvi korak pre slanja je otvaranje komunikacije sa FTP serverom. To je omogućeno pomoću metode access(), kojoj su prosleđeni

adresa FTP servera, korisničko ime i lozinka iz konfiguracionog fajla. Nakon uspostavljanja komunikacije sledi provera da li se FTP serveru šalje fajl ili direktorijum i nakon toga slanje.

Centralni server ima otvoren API za komunikacije sa lokalnim servisima koji su pokrenuti kod svakog klijenta. Dalje ima podignut FTP server koji prima ili šalje fajlove klijenata. Kao i lokalni servis, koristi konfiguracioni fajl. U njemu je zadata putanja gde će se čuvati fajlovi i adresa na kojoj će biti podugnit FTP server. Podaci se čuvaju u Mongo bazi, a jedan od najbitnijih podataka za sinhronizaciju fajlova je njihova hash vrednost. Kada na centrani server stigne fajl, HTTP zahtevom stižu i informacije o tom fajlu koje sadrže hash vrednost. Korsnici koji preuzimanju fajlove, prvo zatraže informacije o fajlu. Nakon toga sledi provera da li imaju taj fajl i da li je to poslednja verzija. Provera se vrši upoređivanjem hash vrednosti. Kada svi korisnici preuzmu fajlove sa centralnog servera sledi oslobađanje prostora. Provera da li je moguće osloboditi prostor obavlja se tako što se upoređuju dve šeme u Mongo bazi, SharedSchema i DownloadedSchema.

6. ZAKLJUČAK

Tema ovog rada je pravljenje rezervnih kopija digitalnih dokumenata. Predstavljen je sistem koji je implementiran kako bi ispunio potrebe bekapa i skladištenja podataka na više lokacija. Objašnjeni su pojmovi koji su bitni za temu skladištenja podataka. Preporuka za dalji razvoj je uključivanje više bezbednosti u sistem. Protokole koji su korišćeni zameniti sa njihovim bezbednijim verzijama. Skladištenje podataka i konfiguracinog fajla osigurati od neželjenog pristupa. Proširenje sistema korsnicima sa više privilegija kao što su administratori moglo bi biti od koristi. Ako bi sistem bio u široj upotrebi bilo bi korisno da neko ima pristup i mogućnost nadgledanja i upravljanja takvim sistemom, a ostalim korisnicima uskratiti takve funkcionalnosti.

7. LITERATURA

[1] Dragan Ivanović, Branko Milosavljević. "Upravljanje digitalnim dokumentima", 2015.

[2] Azad Adam. "Implementing Electronic Document andRecord Managment Systems", 2007.

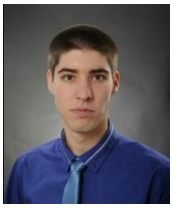
[3] Long-term preservation, https://www.cines.fr/en/long-term-preservation/a-concept-problems-2/long-term-preservation-concept/ (pristupljeno u oktobru 2020.)

[4] File Fixity, https://en.wikipedia.org/wiki/File_fixity (pristupljeno u oktobru 2020.)

[5] Node.js documentation, https://nodejs.org/en/docs/ (pristupljeno u septembru 2020.)

[6] MongoDB Manual, https://docs.mongodb.com/manual/ (pristupljeno u septembru 2020.)

Kratka biografija:



Marko Mijatović rođen je 29.8.1996. godine u Beogradu. Osnovnu školu „Svetozar Marković Toza“ završio je 2011. godine. Gimnaziju „Isidora Sekulić“ u Novom Sadu završio je 2015. godine. Iste godine upisao se na Fakultet tehničkih nauka, smer Primenjeno softversko inženjerstvo. Diplomirao u roku. Školske 2019/2020. godine upisao je master studije, smer Računarstvo i automatika, modul Elektronsko poslovanje. Položio je sve ispite predviđene planom i programom.