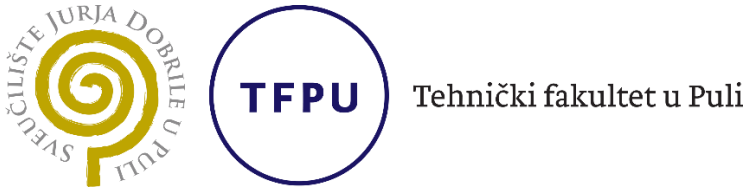
Metode rješavanja konflikta u stvarnom vremenu za kolaborativno uređivanje teksta

Kukić, Filip

Sveučilište Jurja Dobrile u Puli

Tehnički fakultet u Puli



Filip Kukić

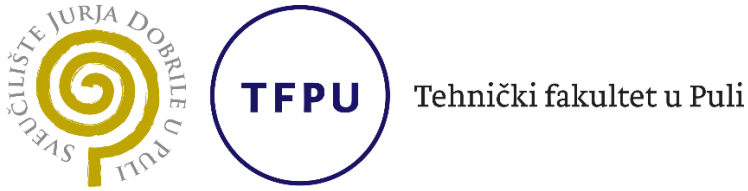
Metode rješavanja konflikta u stvarnom vremenu za kolaborativno uređivanje teksta

Diplomski rad

Pula, rujan 2024.

Sveučilište Jurja Dobrile u Puli

Tehnički fakultet u Puli



Filip Kukić

Metode rješavanja konflikta u stvarnom vremenu za kolaborativno uređivanje teksta

Diplomski rad

JMB: 0303090364, redovan student

Studijski smjer: Računarstvo

Predmet: Raspodijeljeni sustavi

Znanstveno područje: Tehničke znanosti

Znanstveno polje: Računarstvo

Znanstvena grana: Računarstvo

Mentor: doc. dr. sc. Nikola Tanković

Pula, rujan 2024.

Zahvala

Ovim putem bi se htio zahvaliti svom mentoru doc. dr. sc. Nikoli Tankoviću na izdvojenom vremenu i savjetima prilikom pisanja ovog diplomskog rada.

Zahvalio bi se i Sveučilištu Jurja Dobrile u Puli na mogućnosti stjecanja ovoga iskustva.

Također bi se htio i zahvaliti svima koji su me podupirali tijekom studiranja na fakultetu. Najviše hvala mojim roditeljima koji su imali strpljenja tijekom svih ovih godina i koji su me konstantno podupirali kroz čitav moj život. Hvala i mom bratu Marku koji mi je pomagao kad god je bilo potrebno. I za kraj mojim prijateljima koji su me pomagali čitavo vrijeme.

Sadržaj

1. Uvod

Rad će se baviti metodama rješavanja konflikta u stvarnom vremenu kod kolaborativnog pisanja teksta. Detaljno će se obraditi kako i zašto uopće zajedno pisati. Definirat će se što su konflikti i kako nastaju. Objasnit će se detaljno sve metode rješavanja tih konflikta i kako to rade. I na kraju će se pokušati implementirati neke od tih metoda u vlastiti program kako bi ih se testiralo.

1.1. Hipoteza rada

Hipoteza rada je da metodama rješavanja konflikta u potpunosti rješavamo probleme i omogućavamo korisnicima da pišu tekstove bez smetnji i zabuna.

1.2. Predmet istraživanja

Predmet istraživanja rada su metode rješavanja konflikta koje će se u radu detaljno istražiti i usporediti.

1.3. Problem istraživanja

Potrebno je detaljno objasniti sve metode i zatim ih implementirati u vlastiti program, te testirati i usporediti efikasnost.

1.4. Ciljevi rada

Detaljno objasniti način rada metoda rješavanja konflikta te pokušati uspješno implementirati dvije ili više metoda i usporediti njihovu efikasnost.

1.5. Metodologija rada

Za stvaranje programske aplikacije koristit će se programski jezici Python i JavaScript uz alate FastAPI i VueJS.

1.6. Struktura rada

Prvo će se kratko navesti povijest pisanja i tehnologije kojima je ono napredovalo. Nakon toga će se definirati tehnologije koje su pridonijeli razvoju pisanja. Objasnit će se pojmovi kolaborativnog pisanja i točno što su konflikti prilikom zajedničkog pisanja. Nakon što se definira sva potrebna teorija navesti će se najkorištenije metode rješavanja konflikta i detaljno objasniti. Detaljno će se opisati način rada svake metode i dodatno pojasniti uz primjere. Na kraju će se detaljno objasniti programska

implementacija odabranih metoda u napravljenom programu. Tu će se metode usporediti s gledišta programera i korisnika.

2. Motivacija

Čovječanstvo je kroz čitavu svoju povijest pisalo, od ranih početaka gdje je tekst bio u obliku simbola pa sve do danas gdje se većinom piše na računalima. Pismo se može definirati kao sustav vizualnih ili taktilnih znakova kojima se bilježe i čuvaju jezične poruke. U nastavku će se navesti neki od razloga za pisanje. [1].

2.1. Zašto pisati

Glavni razlog zašto su ljudi počeli pisati jest da komuniciraju s drugima. Kako se čovječanstvo razvijalo i raslo potreba za jasnim načinom komuniciranja na daljinu je rasla, a kako bi se to postiglo, počelo se koristiti pismom koje se mogao prenijeti na velike udaljenosti. Prije pisama se na daljinu komuniciralo dimnim signalima, glasnim zvukovima rogova ili jednostavno usmenom predajom ljudi koji su putovali od pošiljatelja do primatelja. Ali ti načini su samo bili efektivni za kratke, jednostavne poruke. Razvojem pisma se taj problem riješio i omogućio jasnu komunikaciju na velike udaljenosti [1].

Komunikacija nije bila jedini razlog pisanja, osim komunikacije pisalo se još kako bi se spremale informacije. Bilo to u skladištu gdje se moralo znati koje stvari su u inventaru, kako bi se zauvijek pamtio neki veliki povijesni događaj, kako bi pamtili preminule osobe i slično. Osim zapisivanja stvarnih događaja, pismom se još spremalo i izmišljene priče. Čovječanstvo je i prije pisama stvaralo umjetničke priče kako bi prenijeli neku emociju ili lekciju sljedećim generacijama, ali se to samo moglo raditi usmenom predajom. Stvaranjem pisma se moglo lakše prenositi priče, ali je i omogućilo stvaranjem puno dužim i kompliciranijim pričama koje se ne bi mogle prenositi usmeno, te se time počela stvarati književnost. Prvo takvo djelo za koje znamo, odnosno koje je preživjelo od tad, je ep o Gilgamešu, to je najstarije poznato književno djelo koje je sačuvano. Napisano je na glini i sadrži niz legendi i pjesama [3].

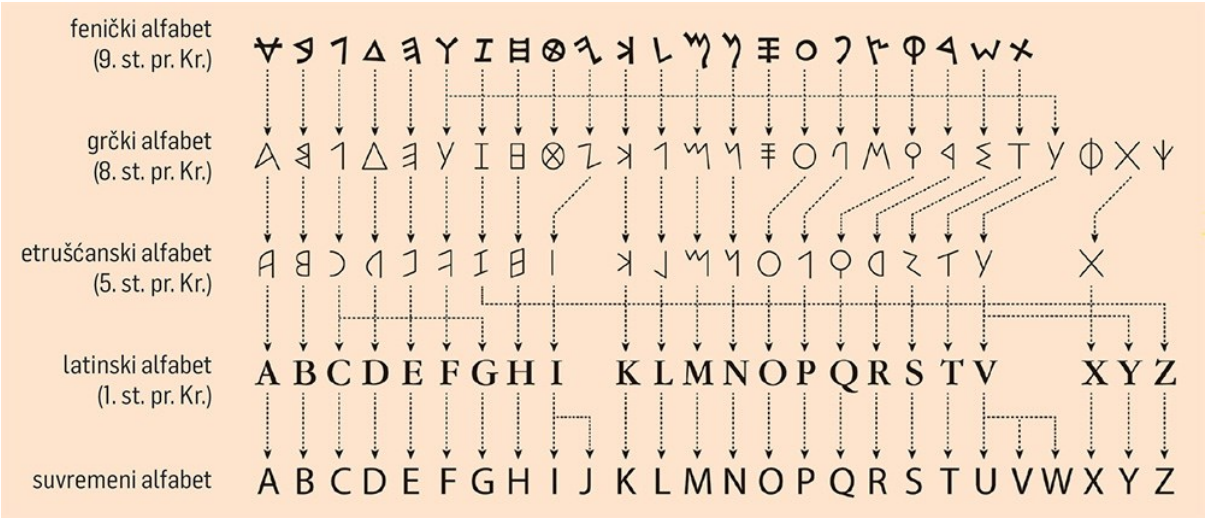
Zadnji najvažniji razlog pisanja je radi spremanja i razvijana znanosti. Pismo je omogućilo znanstvenicima da spremaju svoje ideje što je olakšalo budućim generacijama jer su mogli samo nastaviti istraživati na već otkriveno umjesto da svaki put kreću ispočetka. Zbog toga se znanost počela naglo razvijati i širiti, što i pokazuje podatak da se većina znanstvenih postignuća postigla nakon otkrivanja pisanja [3].

2.2. Početci pisanja

Prije pojave pisma se koristilo slikama i jednostavnim linijama za prenošenja poruke. Prvi ljudi su koristili zidove špilja za prenošenje poruka, slikali bi svoja postignuća ili neke poruke koje bi htjeli da ostanu zapamćene. Prvo korištenje piktografskih ploča bilo je za jednostavno zapisivanje stanje zaliha pomoći običnih linija.

Jedan od najstarijih oblika pisama je klinopis, odnosno klinasto pismo, razvili su ga Sumerani ali se brzo proširilo i na okolne civilizacije. Pisalo se na pločicama gline tako što su se znakovi utisnuli komadićem trske, a pismo je dobilo takvo ime jer su otisci imali oblik klina. U početku se najčešće pisalo odozgo prema dolje, ali se s vremenom prebacilo na pisanje s lijeva na desno zbog lakšeg procesa pisanja. Osim klinopisa, drugi jako poznat način pisanja su hijeroglifi koji su razvili stari Egipćani. Pisalo se u obliku znakova koji su imali više različitih fonetskih vrijednosti. Nije bilo zadan smjer pisanja, ali su postojala neka pravila kada pisati u kojoj orijentaciji [4].

Nakon godina pisanja se klinasto pismo počelo pojednostavljivati, od sličica koje se pisale na početku počelo se pisati slogove, te na kraju i glasove čime se razvija preteča alfabeta. Može se reći da su Feničani najzaslužniji za razvoj modernog alfabeta. Oni su u 11. stoljeću razvili vlastiti alfabet koji je uvelike pojednostavnio način pisanja te su ga trgovanjem raširili kroz Sredozemlje. Zatim su taj alfabet preuzeli i doradili Grci zbog kojih je i tekst dobio ime alfabet prema prva dva grčka slova alfa i beta. Rimljani kasnije preuzimaju alfabet od Grka i stvaraju podlogu za moderni alfabet. Taj tijek jezika se vidi u slici 1 [4].



Slika 1.Tijek razvoja modernog alfabeta [4]

2.3. Moderni izumi za pisanje

Jedan od najvažnijih izuma koji su pridonijeli razvoju pisma je papir. Papir je uvelike olakšao pisanje, čitanje i spremanje teksta, te se i koristi danas zbog istih tih razloga. Iako se dosta pisanja danas obavlja na računalu, još uvijek se dosta tih tekstova printa na papir. Preteča papiru je papirus koji je dobio ime prema istoimene biljke od koje se pravi. Papirus je bio podloga za pisanje u antici, te se zbog svojih prednosti puno koristio. Nije bio otporan na vlagu, ali u suhim uvjetima bi mogao biti sačuvan godinama [5].

Prvi mehanički razvoj koji je promijenio načine pisanja bio je pisaći stroj. Dugo se pokušalo ubrzati i olakšati proces pisanja s pomoću mehaničkih strojeva, te je pisaći stroj rezultat dugogodišnjih pokušaja. Pisaći stroj funkcionira tako da se prilikom pritiska na odgovarajuće slovo metalna poluga spušta na postavljeni papir čime se to slovo utiskuje na njega tintom. Papir se kreće sa svakim unosom slova te se tako pišu rečenice [6].

Pisaći stroj je prethodnik modernih računala, mehanički komponenti stroja su se polako zamijenili s električnim, ali jedna stvar je ostala skoro ista a to je orijentacija slova. QWERTY ili QWERTZ ovisno o regiji je naziv za modernu i danas standardnu orijentaciju slova na tipkovnici. Prvi puta se koristila na pisaćim strojevima i organizirana je tako kako bi razdvojila najčešće korištena slova kako bi se što više smanjilo rizik zaglavljenja mehanizma. Stroj bi prilikom pritiska slova pokrenuo mehanizam kojem je potrebno neko vrijeme prije novog pritiska kako se ne bi zaglavio, te su slova zbog toga organizirana kako bi usporili proces pisanja. Razlog korištenja tog rasporeda tipkovnice danas je samo praktični, jer svi ga već znaju i koriste te bi bilo kompliciranо stvoriti i implementirati novi, a pomalo i nepotrebno [7].

Najvažniji moderan izum za pisanje općenito jest računalo. Danas se većina stvari piše na računalu, iako neka mjesta još uvijek čuvaju važan tekst na papiru jer žele ili je potrebno zakonom, taj tekst je većinom pisan na računalu i naknadno isprintan na papir. Razlozi pisanja na računalu su veliki. Računalo omogućuje jednostavno brzo pisanje s mogućnošću lakog ispravljanja grešaka. Osim toga s pomoću raznih programa moguće je uređivati pisani tekst na dosta različitih načina, te i pisanje različitim fontovima.

Prvi izum koji je stvarno omogućio istovremeno kolaborativno pisanje teksta je internet. Prije toga se općenito pisalo samostalno pa čak i na računalu. Iako se moglo pisati zajedno bilo je potrebno da jedna osoba prestane kako bi druga imala mogućnost pisanja. Razlog tome je jednostavan, koji god mediji pisanja koristili bilo je nezgodno da dvoje ili više osoba pokuša istovremeno pisati. Internet je omogućio spajanjem više računala čime se i ubrzo stvorila mogućnost pisanja zajedno. S pomoću njega se problemi kolaborativnog pisanja riješili, a postojeće programske podrške za pisane su počele stvarati vlastite opcije suradničkog pisanja.

2.4. Programska podrška za pisanje

Programi za pisanje na računalu se općenito zovu uređivači teksta. To su računalni programi koji omogućuju stvaranje, pisanje i uređivanje tekstualnih dokumenata. S pomoću njih moguće je pisati i uređivati tekst na različite načine, spremiti ih na računalu, te i isprintati ih korištenjem printera.

Prvi uređivači teksta na računalima su bili takozvani uređivači linija, kojima je programer mogao mijenjati linije koda određenog programa i pisati komande. Imali su samo mogućnost unosa teksta koji se nije mogao uređivati, pa se nije moglo s njim stvarati tekstualni dokument. Jedan od prvih pravih uređivača teksta bio je WordStar koji se koristio na mikroračunalima. Ime je dobio od engleskog naziva za uređivač teksta „Word procesor“, a na prvi pravi uređivač teksta se odnosi to što ima osnovne mogućnosti uređivanja teksta. Za razliku od prijašnjih gdje se moglo samo pisati, WordStar je omogućavao i jednostavno uređivanje, spremanje te i ispis tog teksta [8].

Danas se može reći da svako računalo na svijetu ima neku vrstu programa za pisanje. Bilo to direktan program koji je instaliran na računalu ili danas sve popularniji programi koji se mogu pristupiti preko običnog internet preglednika. Najpoznatiji i najkorišteniji program za pisanje je sigurno Microsoft Word. Word ima ogromnu količinu načina upravljanja i uređivanja teksta. Od osnovnih kao mijenjanje fonta i veličine proreda, pa sve do stvaranja tablica i matematičkih formula. U odnosu na prvih uređivača teksta se količina opcija formatiranja uvelike povećala, što otvara mnoštvo mogućnosti za stvaranje različitih kompleksnih tekstualnih dokumenata [9].

Osnovna funkcija uređivača teksta je manipulacija teksta. Omogućavaju korisnicima da brzo umetnu, kopiraju i zalijepe tekst. Ta mogućnost brzog i jednostavnog manipuliranja teksta je najvažnija karakteristika tih programa. Uz to moguće je i na

različite načine mijenjati karakteristike fonta, bilo to veličina ili debljina, pa i izgled, moguće ih je promijeniti na bilo koji željeni način. Osim samog teksta moguće je i uređivati raspored stranice na kojoj se nalazi tekst tako da se mijenja veličina stranice ili margina, te i stvaranje zaglavlja i podnožja. Moguće je i dodavanje različitih grafičkih elemenata kao što su tablice, grafovi, slike, pa i videa. Velika količina modernih uređivača teksta nudi i ugrađene provjere pravopisa i gramatike koje automatski prepravljaju pogreške ili ih označe [9].

Jedna od novih popularnih mogućnosti pisanja teksta je podrška za kolaborativno pisanje. Kako programi stvaraju mogućnost njihovog korištenja preko internet preglednika umjesto da ih se mora skidati, stvorila se prilika za suradničko pisanje. Prije toga se općenito nije ni razmišljalo o kolaborativnom pisanju jer su programi bili lokalno instalirani, ali korištenjem preglednika koji je već spojen na internet suradničko pisanje postalo je lakše ostvarivo.

3. Kolaborativno pisanje i metode otklanjanja konflikta

Kolaborativno pisanje je proces pisanja gdje dvije ili više osoba zajedno pišu isti tekstualni dokument. Umjesto jedne osobe, kod kolaborativnog pisanja dvije ili više ljudi odluči da je zbog zajedničke koristi najbolje da određeni tekstualni dokument pišu zajedno. Pisanje se općenito smatra individualnom aktivnosti gdje jedna osoba odluči da piše nešto o nekoj određenoj temi, ali ako smatraju da će biti lakše pisati zajedno ili da će rad na kraju biti bolje kvalitete, onda mogu odlučiti da podijele svoje znanje i napišu zajednički rad [10].

Razlozi za pisanje zajedno su mnogi. Glavna prednost prilikom pisanja zajedno, u odnosu na samostalno, je mogućnost dovođenje novog gledišta na problem ili temu. Kad dvoje ljudi pišu zajedno, daju jedni drugome nove ideje i perspektive na temu o kojoj pišu. Samostalan pisac pruža samo jedan pogled na temu, dok s dvoje ili više ljudi jedni drugima daju više načina razmišljanja čime uvelike raste kvaliteta rada. Osim toga zajedničkim pisanjem se skraćuje vrijeme samog pisanja, jer radom više ljudi se tekst može podijeliti svakome na dio koji mora napisati, umjesto da jedna osoba sama piše sve. Dodatne prednosti koje kolaborativno pisanje pruža je poboljšanje timske dinamike i komunikacije članova tima, te i mogućnost učenje uz pisanje od svojih kolega [11].

3.1. Povijest pisanja zajedno

Prije pojave interneta se pisalo zajedno ali u puno manjoj mjeri nego danas. Ako se pisalo zajedno općenito su to radili znanstvenici ili književnici, a prosječni ljudi su općenito pisali sami. Najveći razlog tome je kompleksnost zajedničkog komuniciranja o tome što pišu. Ako se pisalo nešto kratko onda su se jednostavno mogli uživo sastati i sve odmah napisati, ali kad je u pitanju neki veći rad kao knjiga onda to nije bilo moguće. Danas korištenjem interneta je moguće komunicirati s bilo kojeg dijela svijeta u sekundi, ali tada kako bi se dogovarali kako nešto napisati ili kako bi poslali jedni drugima na pregled već napisano trebalo je dosta vremena. To bi se većinom radilo korištenjem pošte što bi trebalo dosta vremena i uvelike usporavao proces.

Čak ni uvođenjem računala nije se riješilo neke od problema kolaborativnog pisanja. Jer iako je tekst bilo digitalan, kako bi se prenio na veće udaljenosti morao se spremiti na fizički uređaj koji se opet morao prenijeti do primatelja ručno. No uvođenjem

interneta u svakodnevicu se otvorila prilika za rješavanjem tih problema, ali ta promjena se nije dogodila odmah. Bilo to zbog ograničene brzine, ili zbog male potražnje za kolaborativnim pisanjem, uvođenje zajedničkog pisanja za prosječnog korisnika se nije odmah dogodila. Tek kad se već internet uvrštavao u sve moguće programe je kolaborativno pisanje postala opcija na skoro svakom programu i aplikaciji za pisanje.

Odmah nakon što se internet razvio u koristan alat se pokušao implementirati u obične uređivače teksta kako bi se postiglo kolaborativno pisanje. Rezultat toga su prvi kolaborativni uređivači teksta koji su imali samo ograničene mogućnosti. Većina prvih programa su radili samo na internim mrežama, te su se koristili samo unutar organizacija. Kasnije se razvojem interneta omogućilo i razvoj kolaborativnih programa da se koriste zajedno preko čitavog svijeta. Prvi program koji je popularizirao kolaborativno pisanje prosječnom korisniku je bio Google Docs [12].

3.2. Google Docs

Za razliku od Microsoft Word koji se mora instalirati lokalno na računalo, Google Docs se može pristupiti online preko običnog internet preglednika. Iako Word ima mogućnost za kolaborativno pisanje, teže je za pristupiti i koristiti u odnosu na Google Docs. Word koriste ljudi svih razina, od osnovnih korisnika koji samo pišu običan tekst, pa sve do profesionalaca koji pišu znanstvene radove. Dok je Google Docs većinom namijenjen za osnovne korisnike što se i vidi u razlici mogućnosti formatiranja i uređivanja teksta. Google docs nudi samo osnovne funkcionalnosti koje Word ima, što nije ni čudno jer je on besplatan, dok se Microsoft Word mora platiti kako bi se koristio. Ali zato što je namijenjen za prosječnog korisnika dizajniran je tako da bude jednostavan za koristiti, što se i odnosi na mogućnost kolaborativnog pisanja. Zbog svoje jednostavnosti korištenja preko preglednika i laganog stvaranja teksta, te i mogućnost besplatnog korištenja, je postao brzo popularan. S pomoću njega su prosječni korisnici vidjeli prednosti kolaborativnog pisanja čime se ono proširilo kao opcija stvaranja teksta [13].

Iako prosječni ljudi nemaju toliku potrebu za pisanjem zajedno kao što imaju pisci ili znanstvenici, ipak postoje situacije kad je to korisno. Najbolji primjer za to su grupni projekti. Bili oni u školi, faksu ili na poslu ako se nešto stvara skupa kolaborativni

programi omogućuju bolji, brži i lakši rad. Rješavaju sve poteškoće prilikom rada s više ljudi i omogućuju skupini da se fokusiraju na ostvarenje projekta [14].

3.3. Proces pisanja zajedno

S pomoću programa kao Google Docs se proces pisanja zajedno uvelike olakšao i ubrzao. Prije takvih programa se još uvijek moralo sastajati kako bi se dogovorili kako će se podijeliti, odnosno tko će pisati koji dio teksta i okvirno o sadržaju, ali za razliku od danas, tada nisu imali mogućnosti svi pisati zajedno u istom dokumentu. Nakon dogovora o raspodijeli i okvirnom sadržaju mogli su krenuti pisati. Ovisno o veličini zadatka bi proces bio drugačiji. Ako bi bio neki manji tekst mogli su zajedno uživo svi pisati svoj dio svatko za sebe i komunicirati jedni s drugima o kvaliteti napisanog. Zajedno bi na kraju pregledali sve dijelove, popravili greške i spojili sve to u jednu cjelinu, ali ako je veći rad koji se mora pisati onda jedno popodne nije dovoljno vremena. Zato bi morali nakon dogovora svi otići kući i zasebno pisati svoj dio. Mogli bi pisati tako da prvi napiše svoj dio te naprimjer e-mailom pošalje drugima, oni bi pogledali kad stignu i označili greške te mu poslali nazad na ispravak. Tek nakon što se svi dogovore o kvaliteti bi se tekst mogao poslali drugome za nastavak što je bilo jako spor proces. Za drugi način pisanja zajedno bi mogli svi istodobno pisati svoj dio, pa nakon što završe poslati svima na viđenje u neku grupu. Svi bi zatim pročitali druge dijelove i dali uputstva za potencijalni popravak, nakon što bi svi pročitali bi se opet razdvojili i ispravili svoj dio te ga na kraju spojili kad se svi dogovore o adekvatnoj razini kvalitete. No taj pristup ima veliki problem kompleksnosti. Iako je puno brži od prošlog procesa, ako svi istovremeno pišu svoj dio onda može doći do problema kontinuiteta teksta. Jer ako svi pišu jedan nakon drugog onda mogu pravilno nastaviti tematikom teksta gledajući prethodni dio koji je već napisan, no ovako svatko piše svoj dio zasebno i nemaju se na ništa osloniti. Osim toga nakon što svi završe svoj dio, proces ispravljanja i pregleda postaje vrlo složen. Svatko mora odjednom pregledati više dijelova i dati savjet za potencijalni popravak svih, a na kraju bi dobili savijete od drugih koji su najvjerojatnije brzo napisani kako bi uspjeli na vrijeme sve pregledati. Za razliku od prvog načina gdje bi za svaki dio potpuno mogli dati savijete i ideje, te i odmah na početku zajedno dogovoriti o načinu na kojem će tematika teksta ići [15].

Uvođenjem programa za kolaborativno pisanje se riješilo mnogih od tih problema. Svi imaju zajednički tekstualni dokument u kojem pišu. To pomaže u praćenju onoga što drugi članovi tima pišu. Tako se odmah može pregledati potencijalne greške kako

nastaju i vidjeti dali tematika teksta ide u pravome smjeru. Neovisno o tome koja metoda ili procese pisanja se odabere, s pomoću kolaborativnog programa svi članovi tima imaju stalan i u stvarnom vremenu prikaz teksta drugih članova. Tako se konstantno pregledava kvaliteta napisanog teksta i s time i ubrzava proces samog pisanja.

3.4. Konflikti pisanja zajedno

Kao konflikte u stvarnom vremenu kod kolaborativnog pisanja teksta se odnosi na konflikte koje se događaju prilikom istodobnog unosa teksta više korisnika. Nije nikako povezano ni s kakvim konfliktima ljudi nego na pojavu koja se može dogoditi prilikom istodobnog pisanja. Znači na konflikte u ovom kontekstu se odnosi na problem koji se pojavljuje prilikom kolaborativnog pisanja teksta. Kako većina kolaborativnih programa u današnje vrijeme pruža mogućnost pisanja u stvarnom vremenu, pojava konflikta tijekom pisanja je česta. Pisanje u stvarnom vremenu odnosno „real time“ na engleskom se odnosi na mogućnost programa da dvoje ili više korisnika mogu pisati u istome tekstualnom dokumentu istovremeno. To znači da više ljudi pišu tekst istovremeno čime se mogu pojaviti konflikti.

Ako dvoje korisnika odluči istovremeno da unesu neku riječ na isto mjesto u tekstu dogodit će se konflikt. Program pisanja u stvarnom vremenu funkcioniraju tako da nakon svakog unosa slova, kod bilo koji od spojenih korisnika, sinkronizira stanje sa svim ostalima korisnicima. To znači da ako na dvoje spojenih instanci programa se unose različita slova na isto mjesto onda dolazi programu do problema, jer obadvije instance se pokušaju sinkronizirati s ostalima, a šalju različit sadržaj. Ta pojava se zove konflikt. Čak i ako se dogodi da su obadva korisnika upisala istu riječ na isto mjesto još uvijek se može dogoditi konflikt. To je zbog načina na koji program sinkronizira stanje teksta. Rezultat konflikta ovisi o implementaciji programa te može poprimiti dosta različitih stanja, no skoro uvijek rezultira lošim i ne namijenjenim tekstom.

Iako konflikt prilikom pisanja u stvarnom vremenu nije jedini problem koji se pojavljuje tijekom kolaborativnog pisanja, sigurno je jedan od najvećih. Ako program za kolaborativno pisanje teksta nema direktan način rješavanja tih konflikta onda se korisnicima mogu pojaviti nepravilnosti u tekstu. Nuspojave konflikta mogu biti razne ovisno o tome kako je program iskodiran i kako je logika sinkronizacije teksta

implementirana. Neki od najčešćih nuspojava je da program prilikom pokušaja sinkronizacije teksta stvori kombinaciju unesenih riječi. To se može dogoditi na nekoliko različitih načina. Program bi mogao spojiti dvije riječi jedne do druge, na primjer ako jedan korisnik napiše „dan“ a drugi „noć“, nakon sinkronizacije bi se moglo svim korisnicima sinkronizirati kao „dannoć“. Druga mogućnost je da pomiješa slova obadvije riječi tijekom pisanja, pa bi se sinkroniziralo i prikazalo „dnaonć“. Ili bi moglo jednostavno samo sačuvati jednu od riječi, naprimjer ona koja je prva bila upisana i nju sinkronizirati svima. No te nuspojave nisu toliki problem jer jedan korisnik može lagano izbrisati netočnu riječ i prepraviti ju čime bi se ona pravilno sinkronizirala svima. Najopasnije nuspojave su one koje slome program ili oštete datoteku. Prilikom konflikta i pokušaja sinkronizacije programa može se dogoditi da se poremeti sinkronizacija. U nekim slučajevima se za svakog korisnika spremi njihova napisana riječ bez viđenog problema te naknadno nakon pokušaja spremanja teksta bi program spremio riječi kao jednu od kombinacija. Također bi program mogao uspjeti sinkronizaciju ali za svakog korisnika prikazati različitu od te dvije riječi ovisno o tome koju je prije primila od korisnika, čime se opet pojavljuje problem spremanja. Najgora moguća nuspojava konflikta prilikom kolaborativnog pisanja je totalno rušenje sinkronizacije. U tom slučaju sinkronizacija se lomi i tekst više nije kolaborativan. U najboljem slučaju bi ponovnim pokretanjem programa trebalo vratiti zadnje spremljeno stanje teksta, ali se može dogoditi i oštećenje datoteke i trajnim gubljenjem teksta [16].

Naravno kako ne bi došlo do tih problema svaki danas popularni program za kolaborativno pisanje ima neku vrstu načina rješavanja tih konflikta. Ti načini rješavanja takvog problema se zovu metode rješavanje konflikta. Postoje mnogo različitih metoda za rješavanje takvih konflikta i svaka ima svoje najbolje uvjete za primjenu. Od onih koje su namijenjene za jednostavne programe do onih koji su za veće, svaka ima svoju prednosti i manu. Te se mora pregledati stanje programa kako bi se odlučilo koju metodu implementirati.

3.5. Načini rješavanja konflikta

Kako bi se osiguralo sigurno i kvalitetno pisanje za svih korisnika, za kolaborativne uređivače teksta napravili su metode kojima se sprječavaju ili rješavaju konflikti ovisno o metodi. Metode za rješavanja konflikta u stvarnom vremenu u kolaborativnim programima za pisanje su načini rješavanja problema koji se pojavljuje prilikom istodobnog unosa različitog teksta više korisnika tijekom pisanja u datoteci. Oni

nastaju kada dva ili više korisnika istodobno na isto mjesto upiše različitu riječ, čime dođe do problema programu prilikom sinkronizacije stanja teksta. Iako se nisu odmah počeli razvijati istovremeno kad i kolaborativni programi za pisanje teksta, kako su se ti programi razvijali sve je više rasla potreba za metodama rješavanja konflikta. Prvi takvi programi su bili jako jednostavni i mogli se samo pristupiti lokalno, pa nije bilo posve potrebno za njima. No kako su se oni razvijali i postajali sve kompleksniji te i dobili mogućnost da više korisnika pišu zajedno i s udaljenih mjesta, potreba za takvih metoda koje će riješiti potencijalne konflikte je rasla. Zbog toga su prve široko korištene kolaborativne aplikacije odmah pri stvaranju imale jednu ili više metoda rješavanja konflikta odmah ugrađene.

Metodama rješavanja konflikta se otklanjaju problemi koji se pojavljuju prilikom pisanja u stvarnom vremenu. Ti problemi mogu biti razni, od loše napisanog teksta, pa sve do rušenja sinkronizacije programa. Načini rješavanja tih problema su raznih. Zato i postoje različite metode rješavanja konflikta koji rade na različite načine. Može ih se općenito podijeliti na one koje rješavaju problem i one koje ga izbjegavaju. Pod izbjegavanjem se podrazumijeva da ne pokušavaju direktno riješiti problem pisanja zajedno nego ga ograničiti da se ne dogodi, odnosno onemoguće situacije u kojem bi se konflikt mogao dogoditi. Te metode su zato i najlakše za implementirati. Jedna takva metoda rješavanja konflikta ograničava pisanje na samo jednog korisnika istovremeno. Tom se metodom onda ne mogu stvarati konflikti, ali se ograničava kolaborativne funkcionalnosti aplikacije jer bi samo jedan korisnik mogao pisati odjednom. Dorada te metode je da se umjesto čitav tekst, ograniči samo pisanje na jedan ulomak. Time još uvijek više korisnika može istodobno pisati samo ne na istome ulomku teksta. Druge vrste metode su teže za implementirati zato što direktno pokušava riješiti problem konflikta. One rade tako da upravljaju kako program reagira kada pokuša sinkronizirati stanje a dogodi se konflikt. Najčešće implementiraju logiku koja osigurava da svi korisnici vide isti sinkronizirani prikaz tekstualnog dokumenta. Drugim riječima uvode konstantan način rješavanja konflikta koji olakšava rad korisnicima [17].

Kao prva takozvana metoda rješavanja konflikta se može smatrati obična komunikacija. Prije metoda rješavanja konflikta kakve danas poznajemo, prvi kolaborativni programi su ovisili na dogovor korisnika koji pišu. Tadašnji kolaborativni uređivači teksta su bili jednostavniji te su se koristili samo lokalno, pa nije ni bila

potreba za konkretnim metodama. Korisnici bi najvjerojatnije bili spojeni na lokalnoj mreži i komunicirali jedni s drugima dok bi pisali. Na raj način su mogli jednostavno zaobići bilo kakve konflikte jednostavnim dogovorom o tome gdje će tko pisati i kad će netko uređivati koji dio teksta. U slučaju da i konflikt nastane, zajedničkim pregledom teksta bi mogli naći situacije gdje se to dogodilo i prepraviti ga. Iako je ta mogućnost rješavanja odnosno izbjegavanja konflikta moguća i danas, kako se u sadašnje vrijeme sve više piše zajedno preko interneta i s mogućnošću pisanja više ljudi, bilo bi kompliciranije se dogovarati oko toga, te bi mogućnost stvaranja konflikta bila puno veća nego prije. Zbog toga se odlučilo ne riskirati i direktno implementirati metode kojima će se to izbjegavati i riješiti [15].

3.6. OT (Operational Transformation)

OT je metoda za rješavanje konflikta u stvarnom vremenu u kolaborativnim uređivačima teksta. Kratica OT stoji za „Operational transformation“ što bi se moglo s engleskog direktno prevesti na operativna transformacija, ili transformacija operacija kojom se više očitava što je točno OT i kako radi. Radi tako da transformira konfliktne operacije jednog korisnika kako bi bile kompatibilne s operacijama drugog korisnika. To omogućuje korisnicima istovremeno pisanje i uređivanje teksta bez stvaranja konflikta. OT tako omogućava konzistentnu i sigurnu sinkronizaciju stanja teksta između svih korisnika. Osnovno način funkcioniranja OT-a je da se odnosi na operacije kao funkcije koje prate stanje dokumenta kao input i stvori novo stanje tog dokumenta kao output. Operacije u OT-u se smatra svaka promjena stanja teksta. Kao operacije se smatraju svake promjene teksta koje korisnik napravi, koje mogu biti pisanje slova, brisanje slova ili promjena izgleda riječi. Umetanje i brisanje slova usred rečenice se može smatrati kao posebna operacija jer su okružene drugim slovima što stvara dodatnu kompleksnost prilikom obavljanja funkcija, odnosno sinkronizacije. Dokaz o kvaliteti OT-a kao kvalitetne metode rješavanja konflikta je podatak o njegovoj korištenosti danas u kolaborativnim uređivačima teksta. Neki od programa koji koriste OT za rješavanje konflikta prilikom zajedničkog pisanja su Google Sheets, Quip te i potencijalno najveći Google Docs [18].

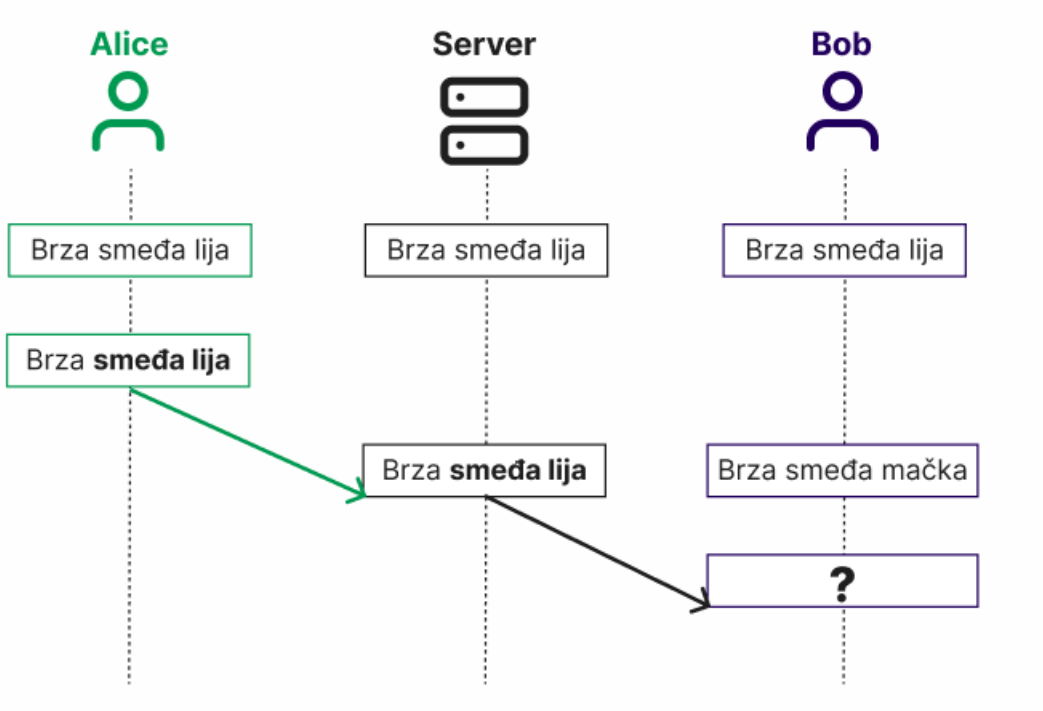
3.6.1. Prije OT-a

Prije uvođenja metode transformacija operacija, kolaborativni programi kao Google Docs koristili su proces uspoređivanja stanja verzija dokumenata. Program bi

pogledao stanje dokumenata povezanih korisnika te bi ih usporedio i spojio, odnosno sinkronizirao da imaju isti sadržaj.

Ako uzmemo za primjer dvoje korisnika Alice i Bob koji zajedno uređuju jedan tekstualni dokument na nekom kolaborativnom programu koji ima osnovnu metodu rješavanja konflikta kao što je imao Google Docs kad je tek stvoren. I njihovi dokumenti odnosno tekst je trenutno sinkroniziran i isti, ako naprimjer Alice napravi neku promjenu teksta, program učita da dokumenti nisu više isti te ih usporedi i nađe što se točno razlikuje između njih. Nakon toga ih pokuša sinkronizirati odnosno spojiti u jedan dokument. Kada ih spoji pošalje novu verziju Bobu kojemu se prikaže promjena koju je Alice napravila. Ako je Bob u međuvremenu napravio promjene koje nisu sinkronizirane onda se njegova verzija usporedi s primljenom i ponovno spoje u novu verziju koja mu se prikaže. Taj proces se ponavlja za svaku promjenu koju Alice i/ili Bob naprave. No taj način rješavanja konflikta ne funkcionira uvijek kako je željeno [19].

Koristit ćemo primjer u slici 2, gdje Alice, Server i Bob imaju sinkronizirano stanje teksta i svima na istome mjestu u dokumentu pište rečenica „Brza smeđa lija“. U slučaju da Alice podeblja tekst „smeđa lija“, a dok ona to radi Bob izbriše riječ „lija“ u tekstu i zamijeni ju s riječju „mačka“ i pretpostavimo da promjena koju je Alice napravila prije stigne do servera. Server onda tu promjenu šalje Bobu koji je već i sam napravio drugu promjenu. Tu dolazi do problema jer program mora spojiti te promjene. Pravilan način spajanja njihovih promjena u ovom kontekstu bi bila rečenica „Brza **smeđa mačka**“, gdje bi program shvatilo da Alice želi podebljati zadnje dvije riječi te rečenice, a da Bob želi promijeniti riječ „lija“ u „mačka“. No server nema takve informacije, za njega postoji nekoliko različitih načina spajanja tih promjena koje bi sve mogle biti pravilne. Iz perspektive servera, rečenice „Brza **smeđa lija** mačka“, „Brza **smeđa** mačka“, „Brza **smeđa** mačka **lija**“ i slično su sve pravilni načini spajanja njihovih promjena. Zbog toga ovaj način rješavanja konflikta nije pouzdan i nije više korišten danas [19].



Slika 2. Primjer konflikta u kolaborativnom pisanju, napravljen u alatu Figma prema slici [19]

3.6.2. Rješavanje konflikta s operacijama

Najnovija verzija Google Docs i sličnih kolaborativnih aplikacija su zbog toga prešle na drugi pristup rješavanja konflikta. Dokument se sprema kao sekvenca operacija koje se primjenjuju redoslijedom kojem su nastali. Za to se koristi totalni redoslijed nastanka operacija, što znači da se za svaku operaciju koja nastane uvijek sprema točan zadani redni broj nastanka. Korištenjem totalnog redoslijeda se za svaku operaciju zabilježi njeno vrijeme nastanka koje nikako ne može biti isto kao kod neke druge nastale operacije. Time se osigurava konstantnost i sljednost obavljanja operacija prilikom sinkronizacije teksta. Onemogućava situaciju gdje bi se operacije izvodile pogrešnim redoslijedom ili situacije kad bi dvije operacije imale isto vrijeme izvođenja čime bi onda program morao nagađati koju prvo da izvodi. Zbog tog načina izvođenja, gdje se koriste operacije umjesto uspoređivanje dokumenata, se ova metoda nazvala metodom transformacije operacija odnosno OT [19].

Glavna odluka prilikom uvođenja metode OT je definiranje što se to točno smatra kao operacijama. Sve radnje koje uzrokuju promjenu teksta i koje se moraju sinkronizirati se svrstavaju pod određenu operaciju. Naprimjer Google Docs sve operacije svrstava pod 3 moguće vrste:

- Umetanje teksta
- Brisanje teksta
- Primjena stila

Kada korisnik uredi dokument, sve promjene se spreme u zapisnik promjena (eng. ChangeLog) u jedno od tih 3 vrsta operacija. Zapisnik promjena je virtualni zapisnik ili dnevnik u kojem se bilježe sve promjene koje se dogode u tom dokumentu. Bilo kada itko od spojenih korisnika napravi bilo kakvu promjenu teksta, ona se zapiše u taj zapisnik s točnom vremenskom oznakom nastanka kao jedno od dogovorenih vrsta. Pod operacijom umetanja teksta se zapisuje bilo koja operacija kojom se zapisuje odnosno umetne tekst u dokument. To može biti nastavljanje pisanja rečenice ili dodavanje slova unutar postojeće rečenice. U zapisnik se bilo kakvo brisanje slova ili riječi sprema pod operaciju brisanja teksta. A pod primjenom stila se zapisuje bilo kakvu promjenu stila riječi ili čitavog teksta. Te promjene mogu biti mijenjanje fonta, veličine, dodavanje podebljanja i slično. No promjene se ne moraju svrstavati u samo te tri vrste, svaki program može odlučiti drugačiji način svrstavanja, ali svi rade na sličan način [19].

3.6.3. Način rada OT

Kako bi jednostavno objasnili način rada metode OT, ponovno ćemo prikazati primjer dvoje korisnika Alice i Bob. Koji opet zajedno pišu isti tekst u nekim od kolaborativnih uređivača teksta koji koriste metodu OT rješavanja konflikta. Zajedno započinju uređivati tekst koji ima istu početnu rečenicu „Pozdrav svime“. Alice odluči popraviti tu rečenicu tako da izbriše slovo 'E' s njenog kraja i zamijeni ga sa slovom 'A'. U programu se to očituje kao što je prikazano u slici 3, gdje je slikovito i tablično prikazano kako program „vidi“ promjene u tekstu. Program označuje svako slovo sa zasebnim indeksom nad kojim vrši određene operacije na temelju promjene korisnika. Radi lakšeg objašnjenja su u rečenici slova označena jednostavnim indeksom od 0 do 1. Kad Alice izbriše slovo 'E', program to pretvori u operaciju {Izbriši @12} koja mu označava da mora izbrisati sadržaj odnosno slovo koje se nalazi na indeksu 12. Zatim

kada Alice upiše na to isto mjesto slovo 'A', program to pretvara u operaciju {Umetni 'A', @12} koja mu označava da mora na indeks 12 umetnuti slovo 'A' [19, 20].

	Alice												
Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12
	P	O	Z	D	R	A	V		S	V	I	M	E
Izbriši @12	P	O	Z	D	R	A	V		S	V	I	M	
Umetni 'A', @12	P	O	Z	D	R	A	V		S	V	I	M	A

Slika 3. Prikaz promjena teksta, slika napravljena u programu Excel prema slici [19, 20]

Istovremeno dok Alice pravi tu promjenu, Bob odluči upisati riječ „Veliki“ na početak rečenice. Program unos prvog slova pretvori u funkciju {Umetni 'V', @0}, te zatim svako sljedeće slovo učita kao {Umetni 'Slovo', @indeks} čime umetne na početak rečenice svako slovo pojedinačno kako je uneseno s tipkovnice kao što je prikazano u slici 4. A kako nije izbrisano slovo na koje se postavlja novo, program jednostavno pomiče sva ostala slova na jedan indeks veće. Tako program omogućuje jednostavan i neprekidan unos novih riječi korisnika [19, 20].

	Bob																			
Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	P	O	Z	D	R	A	V		S	V	I	M	A							
Umetni 'V', @0	V	P	O	Z	D	R	A	V		S	V	I	M	A						
Umetni 'E', @1	V	E	P	O	Z	D	R	A	V		S	V	I	M	A					
Umetni 'L', @2	V	E	L	P	O	Z	D	R	A	V		S	V	I	M	A				
Umetni 'I', @3	V	E	L	I	P	O	Z	D	R	A	V		S	V	I	M	A			
Umetni 'K', @4	V	E	L	I	K	P	O	Z	D	R	A	V		S	V	I	M	A		
Umetni 'I', @5	V	E	L	I	K	I	P	O	Z	D	R	A	V		S	V	I	M	A	
Umetni ' ', @6	V	E	L	I	K	I		P	O	Z	D	R	A	V		S	V	I	M	A

Slika 4. Prikaz promjene teksta, slika napravljena u alatu Excel prema slici [19, 20]

Opet će se pretpostaviti da su i Alice i Bob istovremeno napravili te promjene čime dolazi do konflikta, te da je promjenu od Boba program primio prvi. U slučaju da program u kojem oni kolaborativno pišu koristi samo metodu operacija bez da ih transformira, onda se može dogoditi situacija kao kod slike 5. U toj situaciji je program pravilno dodao riječ „Veliki“ koju je Bob napisao, ali kad je krenuo izvršiti operaciju koju je Alice napravila onda je izbrisao pogrešno slovo. To se dogodilo jer je program od

Alice primio operaciju {Izbriši @12} koja mu govori da izbriše slovo indeksa 12, a zato što su komandama koje je Bob uradio pomaknuti indeksi svih slova, onda je program izbrisao pogrešno slovo. Umjesto slova 'E' koje je sad na indeksu 18, izbrisano je slovo 'A' koje se nalazilo na 12 indeksu nakon promjena koju je Bob napravio [19, 20].

	Bob																		
Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	V	E	L	I	K	I		P	O	Z	D	R	V		S	V	I	M	E

Slika 5. Rezultat konflikta, slika napravljena u alatu Excel prema slici [19, 20]

Zato program mora transformirati tu „izbriši“ operaciju kako bi se pravilno primijenila. Metoda OT zato transformira svaku operaciju kako bi radila ono što treba na točnom mjestu. S pomoću nje se operacija {Izbriši @8} transformira u operaciju {Izbriši @18} kojom se izbriše pravilno slovo „E“. To je simbolički prikazano u slici 6 [19, 20].

{Izbriši @8} → {Izbriši @18}

Slika 6. Simbolički prikaz transformacije operacije izbriši, napravljeno u alatu Word prema slici [19]

Tako se transformira i sljedeća operacija koju je Alice uradila, gdje je umetnula slovo 'A'. Opet se na isti način transformira operacija {Umetni 'A', @12} u operaciju {Umetni 'A', @18} kako bi se slovo unijelo na pravilno mjesto. To je simbolično prikazano u slici 7 [19, 20].

{Umetni 'A', @12} → {Umetni 'A', @18}

Slika 7. Simbolički prikaz transformacije operacije umetni, napravljeno u alatu Word prema slici [19]

Nakon što se transformiraju te operacije na pravilan izraz, mogu se primijeniti na tekst koji je Bob napisao Tako se dobije pravilan tekst kao što je prikazan u slici 8 gdje su obadvije promjene Alice i Boba pravilno i na željeni način primijenjene [19, 20].

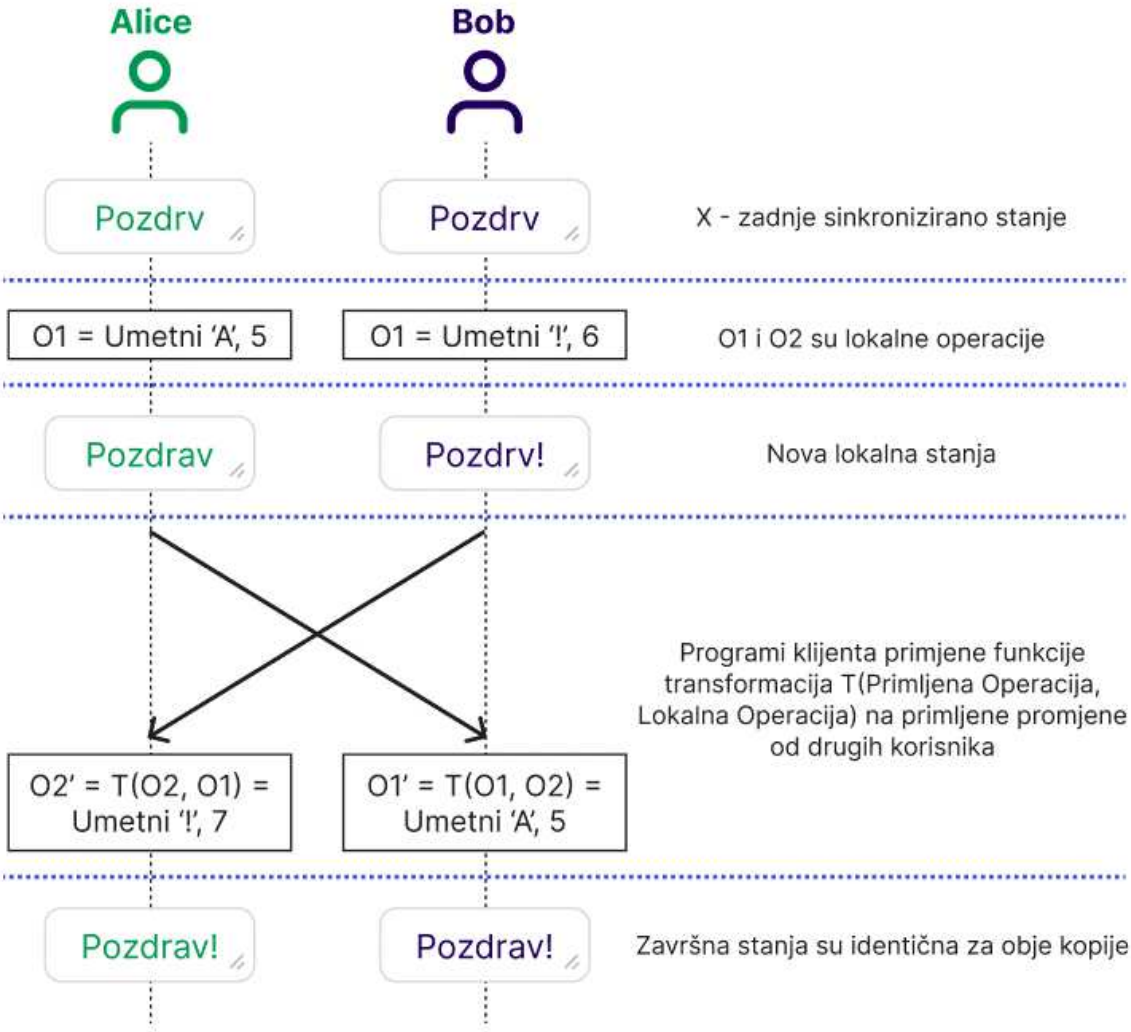
	Bob																		
Indeks	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	18
	V	E	L	I	K	I		P	O	Z	D	R	A	V		S	V	I	M	A

Slika 8. Rezultat metode OT, napravljeno u alatu Excel prema slici [19]

Zbog toga procesa gdje se promjene teksta korisnika pretvaraju u operacije koje se naknadno transformiraju kako bi se pravilno izvodile, se ova metoda zove transformacija metoda ili OT na engleskom [19, 20].

3.6.4. Transformacija operacija

Kako bi se bolje prikazalo način na koji metoda transformira operacije korisnika koristit će se slika 9. U toj slici je ponovno prikazan primjer gdje Alice i Bob zajedno pišu tekst u kolaborativnom programu koji koristi metodu OT. Ovaj put obadva klijenta prikazuju sinkronizirano stanje riječi „Pozdrv“. Alice umetne slovo 'A' u riječ operacijom „O1 = {Umetni 'A', 5}“, kojom se to slovo umeće u indeks 5 riječi i tako dobiva riječ „Pozdrav“. Bob istovremeno odluči dodati uskličnik '!' na kraju riječi što se pretvori u operaciju „O1 = {Umetni '!', 6}“ koja ga umetne na indeks 6 riječi i dobije riječ „Pozdrv!“. O1 i O2 su imena koja su u primjeru dana operacijama koja se lokalno odvijaju kod obadva korisnika i kratice su za Operacija 1 i Operacija 2. Nakon što se kod svakog korisnika primjeni njihova operacija lokalno, dobije se novo lokalno stanje koje je za njih specifično. Kod Alice stoji riječ „Pozdrav“, dok kod Boba stoji „Pozdrv!“, program to primjećuje i pokreće proces sinkronizacije stanja. U slici je to prikazano strelicama kojima se operacije transformiraju u pravilan oblik. Transformacija se jednostavno može opisati kao matematička funkcija koja osigura da se operacije izvode na pravilnim indeksima slova nakon procesa sinkronizacije. U primjeru su funkcije transformacije prikazane sa slovom „T“, dok su transformirane operacije označene s apostrofom kod operacije, znači transformirane verzije operacija O1 i O2 su prikazane kao O1' i O2'. Svaki korisnik primjenjuje funkciju transformacije operacija na primljenu operaciju od drugog korisnika. To znači da kad Alice primi operaciju koju je Bob napravio, onda program transformira tu operaciju kako bi se mogla pravilno primijeniti na njenu napisanu riječ. Bobova operacija se zato transformira na „O2' = {Umetni '!', 7}“, čijom primjenom se pravilno unosi uskličnik '!' na kraj njene riječ i dobije pravilna kombinacija „Pozdrav!“. Na isti način se kod Boba operacija od Alice transformira u „O1' = {Umetni 'A', 5}“ koja je u ovom slučaju ista kao originalna, te se primjenjuje na njegovu riječ i dobiva „Pozdrav!“ kao i kod Alice [19].



Slika 9. Slikoviti prikaz rada metode OT, slika napravljena u alatu Figma prema slici [19]

Tako obadvoje Alice i Bob nakon sinkronizacije imaju istu željenu riječ napisanu koja je „Pozdrav!“. Kod Boba je postojao slučaj gdje se operacija koja je Alice napravila nije ni morala transformirati kako bi bila pravilna, no to neće uvijek biti situacija. To ovisi o mjestu nastanka operacije i dali su njenom primjenom indeksi mijenjane riječi pomaknuti u nekom smjeru. No nije uvijek sigurno kad će se to dogoditi a kad ne, pa je najpametnije uvijek provesti transformaciju za svaki slučaj, što metoda OT i radi [19].

Rezultat transformacije metode OT je prikazana u slici 10. Tu su operacije korisnika prikazane kao O1 i O2, transformirane verzije tih operacija kao O1' i O2', a s X su označena lokalna stanja kod klijenta. Tu se vidi da je nakon sinkronizacije transformirana operacija O1' koja je primijenjena na operaciji O2 koja se primjenjuje na lokalno stanje X ista kao i transformirana operacija O2' koja je primijenjena na operaciju O1 i lokalno stanje X. Drugim riječima ta slika prikazuje da su obadvije

operacije nakon transformacije iste, odnosno da primjenom obadvije na isto lokalno stanje daje isti rezultat. Tako nakon sinkronizacije obadvoje korisnika ima isto stanje [19].

$$O1'(O2(X)) = O2'(O1(X))$$

Slika 10. Slikoviti prikaz rezultata transformacija operacija, napravljeno u alatu Word prema slici [19]

3.6.5. Prednosti i mane metode OT

Najveća prednost metode OT je konzistentnost prilikom izvođenja jednostavnih operacija. Prilikom zajedničkog pisanja OT osigurava da svi korisnici imaju istu verziju dokumenta s pomoću operacija i njihovim transformacijama. Osim konzistentnosti, OT je i fleksibilna metoda jer omogućava obavljanje različitih vrsta operacija kao umetanja, brisanja i formatiranja. Tako programi koji imaju OT implementiran već unaprijed imaju sposobnost rješavanja različitih vrsta konflikta koji mogu nastati prilikom zajedničkog pisanja. Osim toga OT ima mogućnost skalabilnosti. To znači da dobro funkcionira s raspodijeljenim sustavima, što mu omogućava lakšu primjenu pri povećanja sustava na rad s više korisnika. Tako OT omogućava brzo pisanje zajedno u stvarnom vremenu bez konflikta [21].

No OT nije savršena metoda za rješavanje konflikta u stvarnom vremenu. Iako ju koriste popularni programi poput Google Docs, drugi slični programi su odlučili koristiti alternativne metode. Jedan od razloga zašto je njegova kompleksnost prilikom implementacije. Jedna je od najtežih metoda za implementirati, a još teže je to napraviti pravilno. Zbog tog razloga nije preporučeno ju primijeniti u manjim i jednostavnijim programima, u kojima bi bilo puno jednostavnije uvesti neku drugu metodu. Osim toga, prilikom implementacije teško je osigurati pravilno obavljanje operacija i njihovih transformacijama za veliku količinu spojenih korisnika. Iako ima mogućnost lakšeg skaliranja, to vrijedi samo do neke granice. Nakon što se broj korisnika i potom operacija koje oni prave uvelike poveća, puno je teže osigurati konzistentnost. Zbog toga može doći i do loših performansi prilikom izvođenja sinkronizacije s više operacija istodobno. Još jedna loša strana metode OT su takozvani rubni ili ekstremni slučajevi (eng. edge cases). Prilikom istraživanja performansi metode, otkriveno je da OT ima previše rubnih slučajeva, koji se događaju prilikom pokušavanja transformiranja operacija koji se tijekom sinkronizacije razlikuju

na kompleksne načine. Zbog toga može doći do nepravilnog sinkroniziranja ili u najgorem slučaju njenog totalnog prekida [21].

3.6.6. Zaključak metode OT

Metoda OT rješava konflikte pisanja zajedno s pomoću operacija i njenih transformacija. Omogućava rješavanje različitih vrsta operacija te radi tako da svaku promjenu korisnika pretvori u operaciju koju prilikom sinkronizacije transformira kako bi radila kod svih korisnika pravilno. Tako konzistentno rješava jednostavne konflikte koje nastaju prilikom pisanja. No njezina velika kompleksnost prilikom implementacije uvelike smanjuje njenu vrijednost. Iako ima mogućnosti rada s više korisnika zajedno, prilikom velikog povećanja tog broja, postoji mogućnost kvara. Zato je potrebno puno više vremena provesti prilikom implementacije kako bi se to pravilno napravilo. Najveći problem metode su takozvani rubni uvjeti koji nastaju prilikom kompliciranih konflikta koji mogu nastat tijekom pisanja. Zbog tih razloga nije preporučljiva za koristiti u manjim programima, no ako bi se koristila u većim potrebno je više vremena kako bi se pravilno implementirala. To se vidi kod Google Docs kao i kod drugih programa koji su odlučili da kako imaju adekvatne resurse je ipak za njih najbolja metoda OT, jer ako se provede dovoljno vremena na rješavanje tih problema onda je ta metoda dobar odabir za rješavanje konflikta.

3.7. Metoda CRDT

Kratica CRDT je za „Conflict-free Replicated Data Type“, što se može prevesti s engleskog kao „Tip podataka za replikaciju bez sukoba“. CRDT je struktura podataka koja pojednostavljuje distribuirane sustave za pohranu podataka i višekorisničke aplikacije [21]. Drugim riječima CRDT je termin za svaku vrstu strukture podataka koja se može istodobno replicirati i modificirati preko više stranica, a da se može spojiti bez potrebe za centralnim tijelom koje mora koordinirati promjene [22].

Metoda CRDT rješava konflikte u stvarnom vremenu na drugačiji način od metode OT. Kod metode OT se svakom slovu u pisanom tekstu dodaje indeks po kojem se prati, svaka promjena se pretvori u operaciju nad tim indeksom, te se prilikom sinkronizacije te operacije moraju transformirati kako bi bile pravilno primijenjene kod svih korisnika. Dok kod metode CRDT se slovima dodaje jedinstveni identifikator po kojem se svi razlikuju, tako se prilikom sinkronizacije promjene mogu primijeniti bez potrebom za njihovom transformacijom. U metodi CRDT se svakom unesenom tekstualnom znaku

dodaje jedinstveni identifikator koji je povezan s osobom koja ju je stvorila, odnosno unijela u tekst. Prilikom sinkronizacije svaki program spojenog korisnika dobije tuđu verziju teksta gdje učita promjene i primjeni ih na vlastiti tekst. Svaki uneseni znak u tekst ima uz jedinstveni identifikator i dodatne informacije o potrebnom položaju u tekstu. Tako se proces sinkronizacije može odviti bez loših nuspojava konflikta. To je i najveća razlika između CRDT i OT jer za metodu OT je potreban neki sustav koji prilikom sinkronizacije mora obaviti transformaciju svih promjena, dok se to kod metode CRDT odvija samostalno između programa korisnika [23].

3.7.1. Identifikatori znakova

Kod metode CRDT rješavanja konflikta se svakom unesenom znaku dodaje jedinstveni identifikator zajedno s dodatnim potrebnim informacijama. To znači da prilikom svakog unosa neke vrijednosti s tipkovnice ona se zapiše u tekstu, te i u pozadini spremi uz dodatne informacije. Identifikatori obično sadrže neku vrstu broja kojim bi se taj znak razlikovao od svih drugih koji su napisani i koji će biti napisani uz još i oznaku osobe koja ju je napisala. Ta oznaka je identifikator po kojem se razlikuju svi korisnici koji su spojeni i koji pišu na tom tekstu. Tako program može lakše uskladiti promjene teksta prilikom sinkronizacije. Osim toga svaki znak drži i neku vrstu vremenske oznake kojom bi se moglo riješiti problem redoslijeda u slučaju konflikta. Iako je za razliku od metode OT kod metode CRDT moguće provesti promjene neovisno o vremenu nastanka, nekad je potrebno prilikom konflikta imati podatak o vremenu kako bi se pravilnije obavila sinkronizacija, odnosno kako bi se spojio tekst na što poželjniji način. Jer iako je moguće s pomoću metode CRDT spojiti različite promjene bez podataka o vremenu nastanka, s pomoću njega je u nekim situacijama moguće programu da lakše odlučiti koji od nekoliko mogućnosti spajanja teksta da odabere. Uz to program naravno i sprema podatke o samom sadržaju znaka kako bi znao koji da prikaže u tekstualnom dokumentu prilikom pisanja i sinkronizacije [24, 25].

Potencijalno najvažniji spremljeni podatak u vezi s rješavanjem konflikta u kolaborativnom pisanju je podatak o njegovom položaju u tekstu. Taj podatak sadrži informacije o susjednim elementima u tekstu pored kojih se taj specifičan znak nalazi. U njemu su sadržani identifikatori znakova koji se nalaze prije i poslije njega u tom nizu znakova rečenice. Tako je prilikom sinkronizacije moguće spojiti nastale promjene između korisnika, jer s pomoću tih podataka program točno zna gdje je svaki korisnik

unio točno određeni znak. To je i osnovni način funkcioniranja metode CRDT prilikom rješavanja konflikta, i razlog zašto je moguće promjene raditi bez obzira na vrijeme nastajanja i bez potrebe za njihovom transformacijom [24, 25].

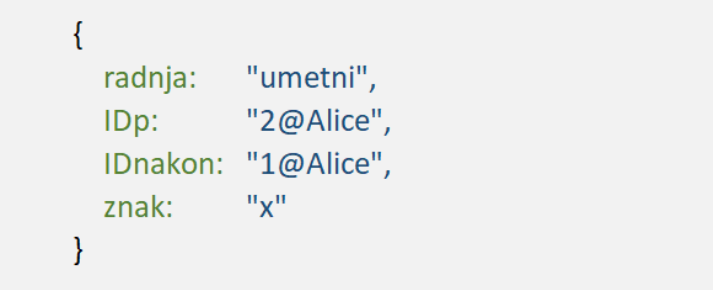
Zadnji podatak koji se još zapisuje uz znak je njegovo stanje aktivnosti. To je najčešće podatak tipa bool koji može samo imati vrijednost 1 ili 0. I u njemu se sprema trenutno stanje tog određenog znaka koji može biti aktivan ili izbrisan. Kod metode CRDT se znakovi prilikom brisanja ne brišu iz memorije, nego se samo smaknu s vida korisnicima koji pišu tekst. Kad korisnik izbriše znak njegovo se stanje mijenja u 0, odnosno izbrisano i ono nestaje s teksta kao da je stvarno izbrisano, ali ipak ostaje u istoj poziciji u kojoj je i bio u memoriji samo se više ne prikazuje korisnicima. Kod metode CRDT se koristi ovaj način brisanja kako bi sinkronizacija bila moguća. Jer u slučaju da jedan korisnik izbriše neki znak, a drugi istovremeno izbriše neki drugi ili isti, programu su potrebne sve informacije o izbrisanom znaku kako bi pravilno obavio sinkronizaciju. Kod metode CRDT je u slučaju konflikta programu potrebna informacija o položaju svakog znaka, a njegovim potpunim brisanjem bi tijekom spajanja falilo informacija kojima bi rješavanje nekih konflikta bilo nemoguće [24, 25].

3.7.2. Način rada metode CRDT

Većina metoda CRDT rade tako da reprezentiraju tekst kao slijed znakova koji svaki ima jedinstveni identifikator uz još i dodatne podatke kao pozicija, vrijednost, aktivnost i vremensku oznaku. Svaki put kad se dogodi promjena u tekstu koja se mora sinkronizirati i ona dobije jedinstveni identifikator kako bi ih program mogao pratiti. Te promjene se ponekad zovu i operacije, no nisu slične kao operacije kod metode OT jer ne rade na isti način i ne transformiraju se. Zbog toga će ih se u tekstu zvati promjenama [24].

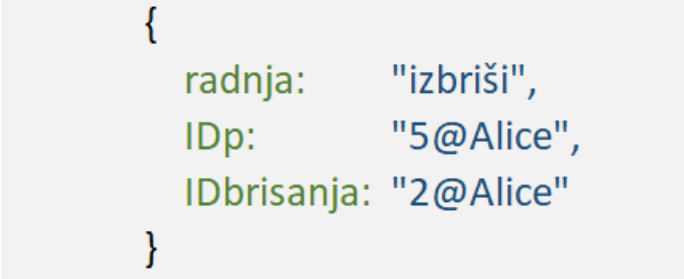
Svakoj se promjeni nakon što je napravljena dodaje njoj jedinstveni identifikator, zajedno s dodatnim informacijama potrebnim za njezino pravilno sinkroniziranje. U slici 11 je prikazan primjer jedne promjene s njenim informacijama. Promjena je ovdje prikazana u formatu JSON zbog lakšeg pregleda, može se i koristiti u stvarnim primjenama metode CRDT, ali se češće odaberu brže opcije zapisa. U slici je prikazan primjer promjene umetanja slova koju je jedan korisnik, u ovom slučaju zvan Alice, napravio. Vrsta promjene je prikazana pod imenom „radnja“, a može biti oblika umetni, izbriši, formatiraj, itd. Pod „IDp“ je stavljen jedinstveni identifikator promjene koji je u

ovom slučaju „2@Alice“. Odabir imenovanja identifikatora mogu biti razni, u ovom slučaju je odabrano ih imenovati na isti način kao i znakove, gdje je redni broj promjene prikazan prvi, zatim ime osobe koja ju je napravila. U ovom primjeru je to druga primjena koju je Alice napravila. Nakon toga stoji identifikator znaka nakon kojeg je unesen novi znak. Taj podatak programu označuje gdje mora stajati ovaj znak nakon sinkronizacije teksta. U ovom primjeru stoji da mora biti nakon znaka s identifikatorom „1@Alice“, odnosno nakon prvo umetnutog znaka koji je Alice napisala. Kako Alice kontinuirano piše svako umetanje se spremi kao promjena u kojoj stoji da taj novi znak mora biti nakon starog. Zadnji podatak koji se nalazi u promjeni je podatak o samom unesenom znaku [24].



Slika 11. Primjer promjene u metodi CRDT, slika napravljena u alatu Excel prema slici [24]

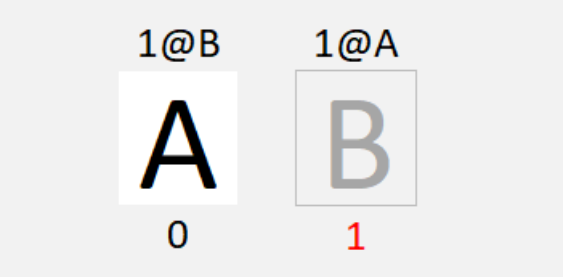
Promjena brisanja slova ima manje podataka za spremiti. Kao što je prikazano u slici 12 još uvijek se sprema koja je operacija i njezin identifikator. Ali za razliku od umetanja, gdje je potrebno spremiti znak koji se umeće i na koju poziciju, ovdje je samo potrebno zapisati identifikator znaka koji je izbrisan [24].



Slika 12. Primjer promjene brisanja, slika napravljena u alatu Excel prema slici [24]

Kako bi grafički prikazali znakove i podatke koji oni sadržavaju tijekom objašnjavanja primjera koristit će se prikaz kao u slici 13. Tu je znak koji je unesen prikazan u sredini kao najveći. Ispod njega je prikazana njegova aktivnost, ako je vrijednost 0 onda je znak aktivan, a ako je 1 onda je neaktivan odnosno izbrisan. U slici je slovo 'A' još aktivno dok je slovo 'B' izbrisano. Iznad znaka je prikazan njegov identifikator koji je

imenovan prema korisniku koji ga je upisao i rednim brojem upisa. U slici je primjer gdje je slovo 'A' upisao Bob kao svoj prvi znak, a slovo 'B' je upisala Alice isto kao svoje prvo slovo [24, 25].



Slika 13. Grafički prikaz znaka za primjer, napravljeno u alatu Excel prema slici [24]

Tako je prikazan primjer u slici 14, gdje Bob i Alice pišu zajedno u kolaborativnom uređivaču teksta koji koristi metodu CRDT rješavanja konflikta. U slici je prikazana jedna rečenica koju su Alice i Bob zajedno napisali. Iz identifikatora iznad riječi je vidljivo da je prvi dio rečenice „lija je“ napisala Alice dok je drugi dio „ skočila.“ napisao Bob. Kako je Bob odlučio da rečenica mora započeti s velikim slovom odlučio je izbrisati prvo slovo i zamijeniti ga s velikim. Kod programa se malo slovo stavlja u neaktivnost i briše s pogleda, dok se veliko postavlja prije njega. Kad Alice primi tu promjenu program s pomoću podataka o promjeni i znaku pravilno sinkronizira promjene i umetne slovo na pravo mjesto. Tako funkcionira metoda CRDT prilikom zajedničkog pisanja [24, 25].

početak	10@B	1@A	2@A	3@A	4@A	5@A	6@A	7@A	1@B	2@B	3@B	4@B	5@B	6@B	7@B	8@B	9@B	kraj
	L	l	i	j	a	␣	j	e	␣	s	k	o	č	i	l	a	.	
Aktivnost	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

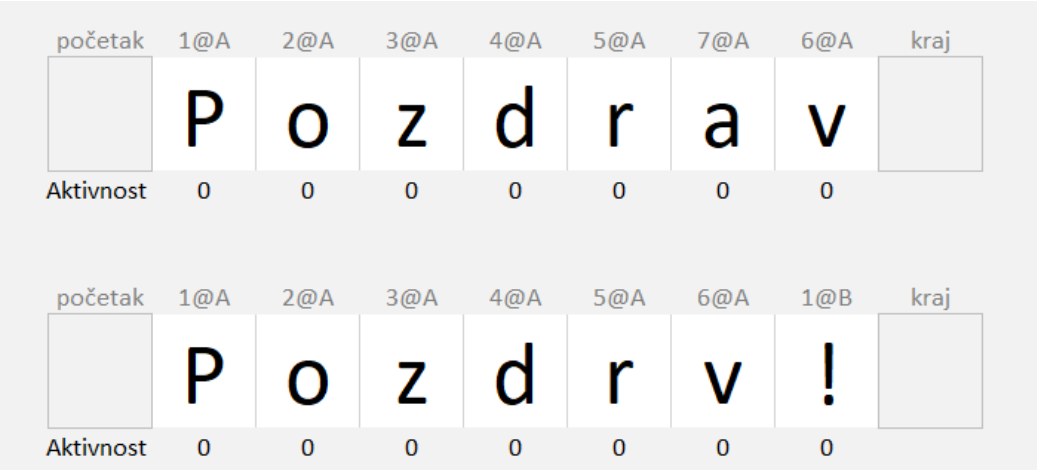
Slika 14. Primjer zajedno napisane rečenice, napravljeno u alatu Excel prema slici [24]

3.7.3. Primjer konflikta kod metode CRDT

Opet će se koristi primjer kao kod metode OT gdje Alice i Bob počinju s istom riječi „Pozdrv“ kao u slici 15 koju je napisala Alice. Ali ovaj put se pretpostavlja da zajedno pišu u kolaborativnom programu koji koristi metodu CRDT rješavanja konflikta.

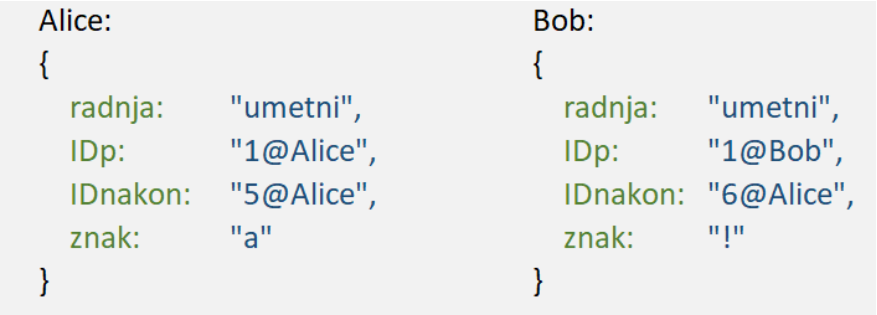
Slika 15. Primjer početne riječi, slika napravljena u alatu Excel prema slici [24]

Kao i prošli put postavlja se situacija gdje Alice i Bob istovremeno odluče promijeniti rečenicu tako što svako unese različiti znak. Alice odluči unijeti znak 'a' između znakova 'r' i 'v' kako bi ispravila rečenicu da pravilno piše „Pozdrav“. Istovremeno Bob odluči da unese znak uskličnik '!' na kraj svoje rečenice, čime kod njega sad piše riječ „Pozdrv!“. To je prikazano u slici 16.



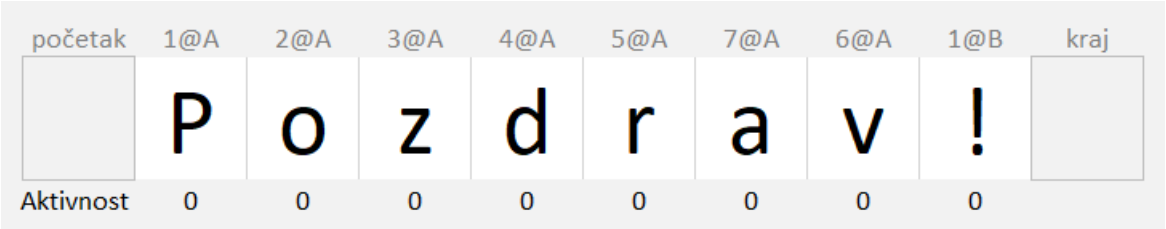
Slika 16. Prikaz promjene rečenice, slika napravljena u alatu Excel prema slici [24]

Nakon njihovih promjena program primijeti da se rečenice razlikuju te ih odluči sinkronizirati. Alice primi promjenu koju je Bob napravio koja je prikazana u slici 17. Promjena programu govori da umetne znak uskličnik '!' nakon znaka s identifikatorom „6@Alice“ koji je znak 'v'. A Bob dobije promjenu koju je Alice napravila koja kaže programu da umetne znak 'a' nakon znaka s identifikatorom „5@Alice“.



Slika 17. Primjer promjena umetni, slika napravljena u alatu Excel prema slici [24]

Nakon tih promjena dobije se pravilna riječ „Pozdrav!“ kao što je prikazano u slici 18. Zbog tog načina spremanja promjena je programu lakše ih sinkronizirati jer neovisno o mjestu nastajanja i redoslijeda je moguće utvrditi poziciju gdje se mora umetnuti. Jer za razliku od metode OT koja radi s pomoću indeksa, kod CRDT se znakovi spremaju pod identifikatorom koji se ne mijenja konstantno prilikom pisanja pa nije potrebno za transformacijama.



Slika 18. Primjer rezultata metode CRDT, slika napravljena u alatu Excel prema slici [24]

3.7.4. Prednosti i nedostatci metode CRDT

Najveća prednost metode CRDT, pogotovo u usporedbi s metodom OT, je manjak potrebe za centralnim tijelom koji mora obavljati operacije. Zbog načina na koji se kod metode CRDT spremaju znakovi, proces sinkronizacije je moguće obaviti bez potrebe za transformacijom tih promjena. Tako spojene instance programa korisnika mogu samostalno rješavati konflikte kako nastaju. Osim toga omogućava jednostavnu implementaciju metode zbog mnoštvo postojećih biblioteka i primjera koji već postoje. Zbog tih razloga je metoda postala sve popularnija, a pogotovo kod manjih programa [24, 25].

Iako je jednostavna za implementirati za manje programe, ako bi se uvodila u veće programe gdje ju je potrebno modificirati, ili iznova dizajnirati postoji problem njene kompleksnosti. Pošto se razlikuje od ostalih metoda prilikom uvođenja u postojeće arhitekture potrebno je dosta truda i modifikacije kako bi pravilno radila. Osim toga ima

i problema sa skalabilnošću prilikom spajanja više korisnika. Najveći njen problem je s performansama, jer iako joj manjak centraliziranog sustava ubrzava rješavanja konflikta, s više korisnika a potom i čvorova može doći do sporijeg vremena sinkronizacije [24, 25].

3.7.5. Zaključak metode CRDT

Metoda CRDT za rješavanjem konflikta u kolaborativnim uređivačima teksta je novija metoda koja koristi strukture podataka kako bi riješila konflikte. S pomoću njih sprema podatke o svim znakovima i promjenama u tekstu kako bi pravilno i efikasno riješila konflikte. Za razliku od OT ovo je novija metoda koja je u rastu pogotovo kod manjih i jednostavnijih uređivača teksta. No zasad se još ne koristi kod većih programa, glavni razlog tome je njena kompleksnost, a još više i način na koji se razlikuje. Jer bi njenim uvođenjem bilo potrebno uvelike promijeniti postojeću strukturu, što zasad nije isplativo. Nastavkom njenog razvijanja, kako budu noviji programi kolaborativnog pisanja nastajali sigurno će se sve više koristiti.

3.8. Metoda locking

Metoda locking, ili prevedeno metoda zaključavanja, je metoda rješavanja konflikta u kolaborativnom uređivanju teksta. Razlikuje se od većinu drugih metoda rješavanja konflikta kao metoda OT ili CRDT po tome što ih ne rješava direktno. Umjesto toga onemogućava nastajanje konflikta tako da osigurava da samo jedan korisnik može istovremeno pisati u tekstu [26].

3.8.1. Način rada

Metoda locking radi tako da onemogući pisanje svim drugim spojenim korisnicima osim onome koji aktivno piše. Iako je metoda rješavanja konflikta ona ga tehnički ne rješava, nego ga samo izbjegava. Nema nikakvu vrstu logike spajanja teksta ili transformacije prilikom sinkronizacije, nego samo blokira pisanje drugim korisnicima sve dok se pisani tekst ne sinkronizira. Pogledamo li opet primjer korisnika Alice i Bob, samo gdje ovaj put pišu u kolaborativnom programu koji koristi metodu locking rješavanja konflikta. i ako pretpostavimo da obadvoje gledaju isti tekst, onda ako obadvoje odluče da krenu istovremeno pisati dogodio bi se konflikt, makar bi da koriste neku drugu metodu. Jer u metodi locking nije moguće istodobno pisati. Čim obadvoje kliknu da pišu u tekstu program bi učitao tko je prvi počeo pisati, makar bilo u milisekundama, i njima omogućio pisanje. Dok bi sporijem samo zamrznulo program i ne bi ni prikazalo ono

što je krenuo pisati. Naprimjer ako pretpostavimo da je Alice bila brža onda ona samostalno može da piše što hoće, a ni ne primjećuje nikakvu promjenu. Dok kod Boba koji je bio sporiji nije ni unijelo ono što je pritisnuo. Kod njega se samo zamrznuo program i označio da netko drugi piše. Tek kad bi Alice prestala pisati i kad bi im se program u potpunosti sinkronizirao, bi Bob mogao upisati ono što je htio [26].

Problem s tim načinom rješavanja konflikta je što onemogućuje pisanje zajedno. Jer s takvim načinom rada, ako netko uređuje samo naslov teksta, netko drugi koji piše na dnu teksta mora čekati da on završi. Zbog toga su se razvile nove verzije metode locking gdje se samo zaključa taj konkretan dio teksta koji se piše. Tako više ljudi može pisati istovremeno samo na različitim dijelovima teksta. To ipak nije savršeno kolaborativno pisanje, no ipak je bolje nego originalna metoda [26].

3.8.2. Prednosti i mane metode locking

Najveća prednost metode locking je njena jednostavnost. Za razliku od većine drugih nema nikakve komplicirane načine pravilnog spajanja teksta prilikom sinkronizacije. Zbog toga je jedna od najlakših metoda za implementiranje. Osim toga je i najbolja u načinu sinkronizacije teksta. Jer kod drugih metoda je moguće da se tekst prilikom konflikta loše sinkronizira na neželjene načine. No to se kod metode locking ne može dogoditi jer se konflikti izbjegavaju općenito. Zbog toga je i vrijeme sinkronizacije puno brže nego kod drugih metoda [26].

No zbog tog načina rada gdje se izbjegavaju konflikti dolazi do problema s kolaborativnim pisanjem. Jer iako se korištenjem te metode piše zajedno, teško je to nazvati pravo kolaborativno pisanje ako se ne može pisati zajedno istovremeno. Iako se s verzijom te metode omogućilo istovremeno pisanje, moguće je samo u različitim dijelovima teksta [26].

3.8.3. Zaključak metode locking

Metoda locking je posebna metoda rješavanja konflikta gdje se konflikti ne rješavaju direktno nego izbjegavaju. Iako se može reći da kvalitetnije obavlja proces sinkronizacije jer nema potencijala za lošim spajanjem teksta, to je zato što totalno onemogućava mogućnost pisanja zajedno. Metoda locking je odličan izbor za male kolaborativne programe koji žele riješiti problem konflikta na lagan, brz i siguran način.

No nije metoda koja bi se preporučila za veće programe gdje bi puno više ljudi morali zajedno pisati.

3.9. Ostale metode

Postoji još nekoliko metoda rješavanja konflikta no nisu toliko popularne i većinom rade na ručnom principu rješavanja konflikta. Kolaborativni programi koji koriste takve vrste metode omogućavaju klasično istovremeno pisanje zajedno, no ako se dogodi konflikt prilikom pisanja, onda upozoravaju korisnike da to moraju sami riješiti na različite načine. Naprimjer kod metode kontrole verzije dokumenta se promjene spremaju u različite verzije i u slučaju konflikta među njima obavještava se korisnike koja je razlika kako bi ju sami popravili. Na sličan način radi i metoda detekcije konflikta samo bez verzija, kod nje se također prilikom konflikta upozorava korisnike i daje različite opcije rješavanja konflikta. Druga vrsta takvih metoda je takozvana metoda s korisničkim ulogama gdje je jedan korisnik glavni pisac, a drugi moraju sugerirati promjene koje on onda odobrava. Sve te metode ovise o korisnicima koji pišu da sami odluče kako da riješe nastale konflikte umjesto da ih program riješi za njih [26].

3.9.1. Prednosti i mane tih metoda

Najveća prednost tih metoda je koliko uključuje same korisnike u proces rješavanja konflikta. Tako se osigura da je rezultat konflikta uvijek poželjan. Za razliku od drugih metoda koji to automatski rade gdje program mora najčešće nagađati kako da sinkronizira tekst. Osim toga zbog načina rada lagano ih je implementirati. Za razliku od većine drugih metoda gdje se mora stvarati komplicirana logika za automatsko rješavanje konflikta, kod ovakvih metoda se smo mora stvoriti način za njihovu detekciju, a samo rješavanje ostaviti korisnicima koji pišu. Zbog tih razloga su te metode jako dobar izbor za manje programe i situacije gdje pišu samo dva ili tri korisnika [26]

No taj proces rješavanja konflikta se može i smatrati njegova najveća mana. Jer u slučaju rada više ljudi može biti frustrirajuće stalno prestajati pisati kako bi se riješilo konflikte umjesto da to program sam radi. Čak i kod manje korisnika neki ne žele taj dodatni posao dok već moraju pisati neki tekst. Osim toga nema većih mana no samo ta je dovoljna kako bi ograničila koji programi koriste ove vrste metoda [26].

3.9.2. Zaključak ostalih metoda

Većina ostalih nespomenutih metoda osim metoda OT, CRDT i locking rade tako da samom korisniku ostavlja njihovo rješavanje. To je odličan način njihova rješavanja, jer sa strane programa olakšava njihovu implementaciju, a sa strane korisnika im daje mogućnost pravilnog rješavanja konflikta. No postoji problem koliko to usporava proces pisanja s više korisnika i ako korisnici ne žele trošiti vrijeme na rješavanje svakog malog konflikta. Zbog toga je ova metoda najbolja za koristiti na manje programe gdje će samo mala količina korisnika pisati i koji bi imali dosta strpljenja tijekom pisanja.

4. Programska implementacija

Kako bi se bolje prikazalo kako metode rješavanja konflikta stvarno rade, stvoren je programski kod uz ovaj rad u kojem su implementirane neke od tih metoda. U programu je odlučeno implementirati metode CRDT i locking, te i opcija pisanja bez ikakve metode kako bi se mogle usporediti. Odlučeno je ne implementirati metodu OT zbog njene kompleksnosti, jer za razliku od drugih metoda koje već imaju napravljenu podršku u nekim alatima, metoda OT bi se morala ručno isprogramirati od početka. A to bi samo po sebi bio previše kompleksan zadatak u usporedbi s ostatkom koda.

Aplikacija ima mogućnosti prijave i registracije korisnika, te pisanjem u jednostavnom uređivaču teksta. Postoje tri različite rute koja svaka koristi različit način rješavanja konflikta, a to su CRDT, locking i bez metode. U aplikaciji korisnici mogu otvoriti željenu metodu u različitim uređajima, ili na isti više puta, kako bi testirali kako određen način rješavanja konflikta radi tako što ga sami stvore. Ova aplikacija je namijenjena samo na testiranju tih navedenih metoda te zbog tog razloga ne sadrži opciju spremanja teksta.

4.1. Korištene tehnologije

Program je iskodiran u programskim jezicima Python i JavaScript. Python je korišten za stvaranje poslužiteljski sloja odnosno pozadinski dio aplikacije, a JavaScript je korišten za stvaranje klijentskog sloja odnosno prednji dio aplikacije koji vide korisnici. Uz Python se još i koristio razvojni okvir FastAPI, odnosno „framework“ na engleskom, kako bi se stvorio brzi i pouzdan poslužiteljski sloj. A uz JavaScript se koristio razvojni okvir VueJS kako bi se stvorio kvalitetan i intuitivan klijentski sloj.

Uz te glavne tehnologije će se još koristiti TipTap, Websocket i Yjs. TipTap je bogati uređivač teksta koji omogućuje pisanje i uređivanje teksta, a koristit će se za stvaranje teksta kojim će se testirati implementirane metode. Websocket je komunikacijski protokol kojim se omogućuje dvosmjerna komunikacija i koji će se koristiti za sinkroniziranje teksta bez korištenja metode. Dok za druge dvije metode će se koristiti Yjs i Hocuspocus za sinkronizaciju. Yjs je implementacija metode CRDT koja se može koristiti uz TipTap kako bi se stvorio kolaborativni program, a Hocuspocus je server koji Yjs-u osigurava Websocket komunikaciju za sinkroniziranje teksta.

4.1.1. Tehnologije poslužiteljskog sloja

Poslužiteljski sloj ove aplikacije iskodiran je u programskom jeziku Python. Odabran je Python zbog svoje jednostavnosti i mnoštvo alata koji se mogu s pomoću njega implementirati. Jedan od tih alata je upravo i FastAPI. To je web razvojni okvir kojim se stvaraju brzi i jednostavni API-ovi s pomoću Pythona. A upravo je i zbog tih razloga odabran FastAPI u odnosu na druge slične alate. API je kratica na engleskom za „Application Programming Interface“ što se može prevesti na programsko sučelje aplikacije. To je sučelje kojim se omogućava međusobna komunikacija između programa. Ono definira pravila aplikacije kojima se drugim aplikacijama govori kako mogu da koristi njene funkcionalnosti. Drugim riječima definira se metode i načini na koji joj se moraju pristupiti kako bi dobio željeni rezultat. Koristi se takav sustav jer se aplikacija nalazi na dvije različite adrese, odnosno aplikacija je podijeljena u dva dijela klijentski sloj i poslužiteljski sloj. A kako bi pravilno i pouzdano radili zajedno potrebna su definirana pravila njihove komunikacije, što se i dobiva korištenjem API-a kao FastAPI [27].

Na poslužiteljskom sloju se još koristi i Websocket tehnologija. To je protokol kojim se omogućava dvosmjerna komunikacija u stvarnom vremenu. FastAPI ima već integriranu podršku za Websocket, pa ga je moguće jednostavno uvesti u kod. Koristit će se za sinkronizaciju teksta na korisničkom sučelju kod primjera bez metode za rješavanja konflikta. Zbog toga nije potreban komplicirani kod, pa je Websocket implementacija u ovoj aplikaciji jako jednostavna [28].

Zadnji dio tehnologija poslužiteljskog sloja je povezana s autentifikacijom, odnosno s prijavom. Iako se podatci za prijavu unose na klijentski sloj, oni se potom šalju na poslužiteljski sloj kako bi se provjerila ispravnost. A taj sustav gdje se radi sa šiframa mora biti siguran, zbog čega se koristi JWT autentifikacija. JWT je kratica na engleskom za „JSON Web Token“ i to je metoda kojom se sigurno prenosi podatke između servera i klijenta. Zbog toga se i koristi prilikom prenošenja važnih podataka poput šifri u aplikacijama. Ima JSON u imenu jer upravo koristi tu tehnologiju za strukturiranje podataka koje šifrira. A šifrira ih s pomoću tajnog ključa koji programer stvara i algoritma za šifriranje. No podatci korisnika nisu spremljeni u poslužiteljskom sloju nego u posebnoj bazi podataka. U ovoj aplikaciji se za to koristi baza podataka MongoDB. Tako aplikacija primi šifrirane podatke od korisnika koje zatim šalje u

poslužiteljski sloj koji ih dešifrira i usporedi sa šifriranim podatcima koji su spremljeni u bazi podataka. Ako su ispravni onda dozvoli ulaz korisnika u aplikaciju [29].

4.1.2. Tehnologije klijentskog sloja

Klijentski sloj je iskodiran u programskom jeziku JavaScript. Odnosno konkretnije koristio se razvojni okvir VueJS koji je iskodiran u JavaScriptu. VueJS je razvojni okvir kojim se prave korisnička sučelja aplikacija. Korišten je jer olakšava stvaranje klijentskog sloja aplikacija s pomoću već dostupnih metoda, funkcija i struktura. Koristan je prilikom stvaranja kompleksnih sučelja, ali je i jako dobar za manja, zbog čega je odličan izbor za ovakvu vrstu aplikacije. Odmah prilikom stvaranja programa se s pomoću VueJS omogućuje preskakanje svih početnih faza stvaranja aplikacije jer s pomoću njega se stvori takozvani kostur aplikacije koji već ima sve osnovne stvari potrebne za njen rad. S pomoću toga se programer može da fokusira na glavni dio svoje aplikacije, umjesto da mora ispočetka sve stvarati. Zbog toga je odličan alat za početnike, ali i za iskusnije programere, jer VueJS daje podlogu za rad na koju programer odlučuje što će da napravi [30].

Kako bi se testirale metode rješavanja konflikta potreban je alat za pisanje, a za tu svrhu je u ovoj aplikaciji odabran TipTap. TipTap je jednostavan uređivač teksta koji omogućava dodavanja raznih dodatke od kojih je jedna i mogućnost kolaborativnog pisanja. Dizajniran je da bude brz, fleksibilan i lagano proširiv novim tehnologijama. Zbog toga ne dolazi s već ugrađenim alatima za uređivanje teksta kao ostali alati, nego se moraju sami dodati. Zbog toga su za obično pisanje bolji odabir drugi uređivači teksta kao Quill koji se u početku planirao koristiti. Jer za razliku od TipTap, on ima već ugrađenu alatnu traku s dosta opcija i jednostavnu mogućnost dodavanja još dodatnih alata. No razlog za implementiranje TipTap u odnosu na druge je njegova opcija za jednostavnu implementaciju za kolaborativnost. Za razliku od drugih uređivača teksta kojima bi se moralo spajati tehnologije koje nisu namijenjene unaprijed da rade zajedno, TipTap već unaprijed omogućuje laganu implementaciju kolaborativnog pisanja. Zbog toga je to najbolji odabir za ovakvu vrstu aplikacije [31].

TipTap omogućuje kolaborativno pisanje s pomoću tehnologija Yjs i Hocuspocus. Yjs je JavaScript biblioteka koja omogućuje kolaborativno pisanje u stvarnom vremenu. Ima već ugrađenu metodu CRDT za rješavanje konflikta kojim olakšava proces implementacije te metode u aplikaciju. Kako bi se ta sinkronizacija mogla obavljati

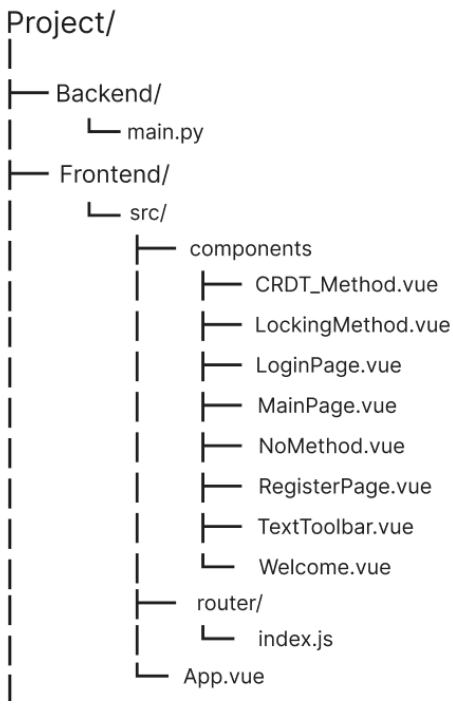
potreban je server, a za to se koristi Hocuspocus. To je NodeJS server koji radi na sličan način kao Websocket gdje služi kao posrednik koji sinkronizira stanje teksta i na kojem Yjs rješava konflikte kako nastaju [32].

4.2. Opis koda

Kod aplikacije se sastoji od mnoštvo dijelova. Dok se kod poslužiteljskog sloja aplikacije od važnijih datoteka smo sastoji od main.py datoteke, na klijentskom sloju se nalazi puno više datoteka. To je većinom zato što se na klijentskom sloju koristi razvojni okvir VueJS kojem je za rad potrebno puno više datoteka. Zbog toga će se samo proći kroz one dijelove koda koji su najvažniji za ovu aplikaciju. A to će biti main.py datoteka koju poslužiteljski sloj ima i glavne VueJS datoteke koje se koriste na klijentskom sloju.S

4.2.1. Struktura projekta aplikacije

Kao što je već spomenuto aplikacija se sastoji od mnoštvo programskih datoteka te će se u radu spominjati samo najvažnije. Ako samo gledamo na te datoteke, struktura projekta izgleda kao u slici 19. Tu se vidi kako izgleda struktura datoteka unutar aplikacije. Poslužiteljski sloj se sastoji od main.py datoteke u kojoj se vrti čitava pozadina aplikacije, dok klijentski sloj većinom sadrži sve Vue.JS datoteke prema kojima se stvara izgled svake rute u aplikaciji.



Slika 19. Prikaz osnovne strukture koda aplikacije, slika napravljena u alatu Figma

Te su rute sastoje od prijave, odjave i svih implementiranih metoda, a sadržane su u datoteci „components“ unutar „src“ datoteke klijentskog sloja. Takva struktura nije slučajno izabrana nego je to način na kojem VueJS organizira svoje datoteke prilikom stvaranja klijentskog sloja. Iako je moguće promijeniti to na drugačiji raspored za to nema potrebe, osim ako netko želi drugačiju strukturu.

4.2.2. Kod poslužiteljskog sloja

Poslužiteljski sloj aplikacije se sastoji od samo „main.py“ datoteke. Tu se odvija čitav dio koda koji se za aplikaciju takozvano odvija u pozadini. Prvo što se nalazi u datoteci su svi uvozi biblioteka potrebni za rad koda. Neki od najvažniji su naravno FastAPI, JWT, Websocket i Pydantic. Nakon toga idu prve linije koda, gdje se prvo inicijalizira FastAPI, odnosno pokreće kao u slici 20. Nakon njega se s pomoću CORS osigurava da poslužiteljski sloj i klijentski sloj mogu komunicirati na željen način. CORS je kratica za „Cross-Origin Resource Sharing“ i to je sigurnosni mehanizam koji ograničava dolazne zahtjeve sa stranih domena od one na kojoj je aplikacija pokrenuta. Koristi se kako strani neželjeni programi ne bi mogli pristupiti rutama programa. Ali kako se u ovoj aplikaciji klijentski sloj nalazi na različitoj domeni, potrebno je programu reći da svi zahtjevi s domene klijentskog sloja su dozvoljeni kako bi aplikacija mogla pravilno komunicirati i raditi. A to se postiže linijama koda prikazanim u slici 20.

```
app = FastAPI()

# CORS configuration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["http://127.0.0.1:8080", "http://localhost:8080"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Slika 20. Prikaz CORS koda, uslikano iz koda aplikacije

Nakon toga slijedi dio koda (Slika 21) kojim se definira token kojim će se šifrirati lozinke korisnika. Mora se definirati tajni ključ i algoritam prema kojima će se lozinke šifrirati prilikom registracije za spremanje i dešifrirati tijekom prijave za provjeru autentičnosti. Osim toga se i definira vrijeme trajanja tokena, čim se odjavljuje korisnika nakon određenog vremena. Ta funkcija se stavlja kako bi onemogućilo drugim ljudima korištenje aplikacije u slučaju da se originalni korisnik sam ne odjavi.

```
#Detalji za token
SECRET_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60
```

Slika 21. Definiranje tokena, uslikano iz koda aplikacije

Poslije toga se definira način spremanja podataka korisnika u bazu. U ovoj aplikaciji se za to koristi baza podataka MongoDB. MongoDB je NoSQL baza podatak koja se koristi za spremanje podataka. Omogućava lagan i siguran način spremanja podataka te je odličan izbor za ovakvu aplikaciju. Prvo se u kodu mora definirati identifikator točne baze u koju se spremaju podatci. Nakon toga se definiraju modeli podataka kao u slici 22. Na temelju njih će se podatci spremati u bazu. U slici je prikazan model tokena na temelju kojeg će baza spremiti token kao podatak tipa string.

```
#Basemodel za Mongo
class Token(BaseModel):
    access_token: str
    token_type: str
```

Slika 22. Model podataka token, uslikano iz koda aplikacije

Nakon što su definirani svi modeli za bazu slijedi dio koda u kojem se definira čitav sustav autentifikacije aplikacije. Definiraju se prvo sve potrebne funkcije kojima se provjerava autentičnost korisnika i sprema ga u bazu. A kako bi se te funkcije aktivirale potrebna je ruta koju klijentski sloj može pozvati. To se postiže rutom „/token“ koja je prikazana u slici 23. Nju se aktivira kad korisnik klikne na dugme prijave na svom sučelju čime se ona poziva s podatcima koje je korisnik unio. Funkcija koja je dio rute zatim koristi pomoćne funkcije kako bi provjerila autentičnost primljenih podataka. U slučaju da nisu ispravne šalje nazad pogrešku, a ako su ispravni stvara token koji zajedno s porukom o ispravnim podatcima šalje na klijentski sloj. Osim te rute još se nalazi i ruta registracije koja prima podatke korisnika koje pomoćnim funkcijama sprema u bazu i tako stvara korisnika. I na kraju rute kojima se provjerava autentičnost prijavljenog korisnika prilikom korištenja aplikacije i ruta za odjavu korisnika.

```python
@app.post("/token")
async def login_for_access_token(response: Response,
    form_data: Annotated[OAuth2PasswordRequestForm, Depends()],
) -> Token:
    user = await authenticate_user(form_data.username, form_data.password)
    if not user:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Incorrect username or password",
            headers={"WWW-Authenticate": "Bearer"},
        )
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.username}, expires_delta=access_token_expires
    )
    response.set_cookie(key="token", value=access_token, httponly=True)
    return Token(access_token=access_token, token_type="bearer")
```

Slika 23. Ruta /token, uslikano iz koda aplikacije

Zadnji dio koda poslužiteljskog sloja je implementacija Websocket. On se koristi za sinkroniziranje teksta na klijentskom sloju kod opcije pisanja bez metode. Zato što se sinkronizira samo običan tekst i potrebno je samo prikazati isti tekst na spojene korisnike, nije potrebna veća implementacija. Pa se koristi samo osnovni Websocket kod koji je prikazan u slici 24.

```python
# Websocket
clients: List[WebSocket] = []

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    clients.append(websocket)
    try:
        while True:
            data = await websocket.receive_text()
            # Broadcast message to all connected clients
            for client in clients:
                if client != websocket:
                    await client.send_text(data)
    except WebSocketDisconnect:
        clients.remove(websocket)
```

Slika 24. Websocket kod, uslikano iz koda aplikacije

4.2.3. Kod klijentskog sloja

Kao što je već spomenuto kod klijentskog sloja se sastoji od mnogo različitih datoteka kako bi pravilno radio, a u tekstu će se spomenuti samo najvažniji. Prvi takav dio koda je takozvani „router“. Nalazi se u istoimenoj datoteci pod nazivom index.js, a u njemu se definiraju sve rute koje se koriste u korisničkom sučelju. Prvo se mora definirati ime datoteke koja će se pokrenuti prilikom pozivanja određene rute. Nakon toga ih se može svaka dodati kao u slici 25 gdje je prikazana ruta metode locking. Prvo se napiše ime putanje, nakon toga ime te rute, i zadnje već prije definirana datoteka koja se otvara prilikom pozivanja te rute.

```
{
  path: '/Locking',
  name: 'LockingMethod',
  component: LockingMethod
},
```

Slika 25. Ruta metode locking, uslikano iz koda aplikacije

Te rute se koriste u App.vue dokumentu aplikacije. U njemu se s pomoću „routera“ stvori navigacijska traka na kojoj su sve te rute izlistane. S pomoću toga korisnik može lagano mijenjati stranice aplikacije po želji. U slici 26 su stvorene rute u navigacijskoj traci koje su prikazane ako je korisnik prijavljen. Tu se vide sve rute za svaku implementiranu metodu uz još i glavnu rutu.

```
<!-- Links visible when user is logged in -->
<template v-if="isLoggedIn">
  <router-link to="/main">Main Page</router-link>
  <router-link to="/CRDT">CRDT</router-link>
  <router-link to="/Locking">Locking</router-link>
  <router-link to="/NoMethod">No Method</router-link>
</template>
```

Slika 26. Rute navigacijske trake, uslikano iz koda aplikacije

Osim tih dijelova, najvažniji još dio koda se nalazi u datoteci koja se zove „components“. Tu se nalaze sve datoteke na temelju kojih se stvara korisničko sučelje koje korisnik vidi. Tu se i nalaze sve implementirane metode koje će se detaljnije objasniti kasnije u tekstu.

4.3. Implementirane metode

U ovoj aplikaciji se implementiralo dvije metode, a to su CRDT i locking, te se i implementirao primjer kolaborativnog pisanja bez metode za usporedbu. Obadvije metode rješavanja konflikta koriste server koji TipTap daje na raspolaganje. TipTap omogućuje jednostavnu implementaciju servera Hocuspocus na kojem će se u ovoj aplikaciji obavljati proces sinkronizacije. Koristi se strani server umjesto da ga se implementira na vlastitom poslužiteljskom sloju zbog jednostavnosti. Jer uz server TipTap nudi i mogućnost korištenja biblioteke Yjs koja već ima implementiranu metodu CRDT rješavanja konflikta. Tako se olakšava stvaranje aplikacije i može se fokusirati direktno na same metode. A za opciju pisanja bez metode se odlučilo implementirati na vlastitom poslužiteljskom sloju. Jer se upravo tu pokušava pokazati primjer pisanja bez metode pa je njena implementacija jednostavna. Zbog toga što nije potrebna nikakva logika rješavanja konflikta je moguće stvoriti samo jednostavni server na poslužiteljskom sloju.

4.3.1. CRDT implementacija

CRDT kao i ostale metode su implementirane u klijentskom sloju aplikacije. Unutar aplikacije se nalazi u „components“ datoteci zajedno s svim drugim metodama i datotekama koje stvaraju korisničko sučelje. Implementacija metode je iskodirana u programskom jeziku JavaScript, a datoteka je tipa VueJS. Pošto je metoda uvezena s pomoću biblioteke Yjs nije ju potrebno direktno implementirati u kod, što ga čini puno jednostavniji za napisati i razumjeti. Prvo što se u kodu definira je obrazac od čega će se korisničko sučelje sastojati. To se radi u HTML dijelu koda koji se kod VueJS definira kao „<template>“ i izgleda kao u slici 27. Pošto ova stranica odnosno ruta sadrži samo tekstni okvir i njegove alate za uređivanje teksta, kod se sastoji od samo nekoliko linija koda. Tekstualni okvir za pisanje se definira pomoću druge linije unutar div klase, dok se u prvoj definiraju svi alati uređivanja teksta koje on koristi.

```
<template>
  <div class="editor-container">
    <TextToolbar :editor="editor" />
    <editor-content :editor="editor" class="editor-content" />
  </div>
</template>
```

Slika 27. <template> metode CRDT, uslikano iz koda aplikacije

Nakon HTML dijela koda slijedi JavaScript dio u kojim se obavljaju funkcije koje osiguravaju pravilan rad tekstnog okvira. Taj se kod nalazi u „<script>“ dijelu koda i kod njega se prvo moraju uvesti sve potrebne biblioteke za rad aplikacije. Prvo se uveze TipTap i sve njegove dodatne biblioteke, nakon toga se uvezu Yjs i biblioteka za kolaborativno pisanje, te i Hocuspocus server. Nakon toga slijedi kod u slici 28 gdje se prvo inicijalizira dokument s pomoću Yjs kako bi proces sinkronizacije i metoda CRDT mogli pravilno raditi. Nakon toga se definiraju komponente koje se koriste u prvom dijelu koda. „EditorContent“ je komponenta koja se uvezla iz TipTap biblioteke i služi za pravilno prikazivanje teksta kako se piše, a „TextToolbar“ je lokalna datoteka koja sadrži sve detalje kako uvezeni alati za uređivanje teksta moraju izgledati. Taj kod se mogao i napisati direktno kod metode, ali zato što svaka metoda koristi iste alate na isti način pametnije ih je napisati jednom i uvesti za svaku nego da se svaki put piše isti kod iznova.

```
const doc = new Y.Doc();

export default {
  name: 'EditorComponent',
  components: {
    EditorContent,
    TextToolbar
  },
```

Slika 28. Inicijalizacija dokumenta, uslikano iz koda aplikacije

Zadnji dio JavaScript koda je funkcija „onMounted“ koja je prikazana u slici 29. Tu se instancira uređivač teksta TipTap prilikom otvaranja stranice. Prilikom toga mora se definirati ime dokumenta i identifikator aplikacije. Na temelju tih podataka server znade kome slati verzije teksta prilikom sinkronizacije podataka. U ovom slučaju je ime datoteke „document.crdt“, a na temelju toga će server slati podatke o stanju teksta svim povezanim korisnicima čiji tekst ima to ime. Tako se osigurava da se sinkroniziraju pravilni dokumenti.

```
  onMounted(() => {
    new TiptapCollabProvider({
      name: 'document.crdt',
      appId: '7j9y6m10',
      token: 'notoken',
      document: doc,
    });
  });
```

Slika 29. Instanciranje TipTap, uslikano iz koda aplikacije

Zadnji dio čitavog koda čini programski jezik CSS, a nalazi se unutar „<style>“ dijela koda. U njemu se definira točan izgled i pozicija uređivača teksta i svih povezanih alata. Na temelju tog koda se odredi točna veličina tekstnog okvira unutar stranice.

4.3.2. Locking implementacija

Metoda locking se nalazi na istom mjestu unutar strukture koda kao i metoda CRDT. Pošto metoda radi tako da zaključava tekstni okvir prilikom pisanja odlučilo se radi jednostavnosti i programiranja i objašnjavanja da se nadogradi na temelju postojećeg CRDT koda. Zato će se i za ovu metodu koristiti Hocuspocus server s Yjs CRDT implementacijom za sinkroniziranje koda. No to neće utjecati na metodu zbog načina na koji metoda locking radi. Jer metoda ne rješava direktno konflikt nego ga izbjegava tako da samo jedna osoba može istovremeno pisati. Zbog tih razloga je odlučeno da je najbolje da se metoda nadogradi na već implementiranu metodu CRDT.

Zbog tih razloga i zato što tekstni okvir ima svugdje isti izgled, kod ove metode se ne razlikuje puno u odnosu na prošle. Jedina razlika su dodatci koji čine metodu locking posebnom. Zbog toga je HTML dio koda skoro isti gdje se definira tekstni okvir i korišteni alati. Jedina razlika je dodatak varijable „locked“ kojom se zaključava tekstni okvir prilikom aktivacije metode, a ta promjena je vidljiva u slici 30.

```
<template>
  <div class="editor-container">
    <TextToolbar :editor="editor" />
    <editor-content :editor="editor" :class="{ locked: isLocked }" class="editor-content" />
  </div>
</template>
```

Slika 30. „<template>“ metode locking, uslikano iz koda aplikacije

Također je početni JavaScript kod isti kao i prošli zbog navedenih razloga. Opet se na isti način definira datoteka i definira server za sinkronizaciju. Novi dijelovi koda su funkcije kojima se pokreće metoda locking prilikom pisanja. Neke od tih funkcija su prikazani u slici 31. Tu su prikazane funkcije „initiateLock()“ koja zaključava tekstni okvir prilikom pozivanja i funkcija „releaseLock()“ koja ga otključava pozivom. Nakon njih ide funkcija „resetLockTimeout()“ koja definira vrijeme trajanja zaključavanja što je u ovom slučaju tri sekunde. Zaključavanje se aktivira čim netko počinje pisati, a nakon tri sekunde neaktivnosti se deaktivira.

```
const initiateLock = () => {
  lockingState.set('lockedBy', editorId.value);
  lockingState.set('isLocked', true);
};

const releaseLock = () => {
  lockingState.set('isLocked', false);
  lockingState.set('lockedBy', null);
};

const resetLockTimeout = () => {
  if (lockTimeout) {
    clearTimeout(lockTimeout);
  }

  // Set a timeout to release the lock after a period of inactivity
  lockTimeout = setTimeout(() => {
    releaseLock();
  }, 3000);
};
```

Slika 31. Funkcije metode locking, uslikano iz koda aplikacije

A proces zaključavanja se pokreće kodom prikazanim u slici 32. To je „EventListener“ što bi se moglo prevesti na slušatelj događaja, a radi tako da prati neko stanje i čeka definiranu promjenu koja mu je zadana da prati. U ovom slučaju prati kada neki korisnik počne pisati, nakon toga pokreće prije navedene funkcije koje zaključavaju tekstne okvire drugih korisnika. Na taj način funkcionira metoda locking u ovoj aplikaciji.

```
// Listen to 'beforeInput' and 'keydown' to catch when the user starts typing
editor.value.view.dom.addEventListener('beforeinput', handleTypingStart);
editor.value.view.dom.addEventListener('keydown', handleTypingStart);
```

Slika 32. Funkcije aktiviranja metode locking, uslikano iz koda aplikacije

Pošto je izgled tekstnog okvira i alata isti kao i prije, CSS kod se nije puno promijenio. Jedini dodatak je detalj o tome kako tekstni okvir mora izgledati kad je zaključan kako bi korisnici znali da netko piše. Taj dio koda je prikazan u slici 33 i u njemu se definira boja kojom se tekstni okvir izbjeljuje kako bi se označilo zaključavanje.

```css
/* Style for when the editor is locked */
.locked {
  pointer-events: none;
  background-color: #e0e0e0;
  cursor: not-allowed;
}
```

Slika 33. Definiranje izgleda zaključanog okvira, uslikano iz koda aplikacije

4.3.3. Implementacija kolaborativnog teksta bez metode

Nakon implementiranja metode je još uveden i primjer kolaborativnog pisanja bez ikakve metode kako bi se mogla usporediti razlika. Pošto TipTap server koji je korišten za prethodne metode ima već ugrađenu metodu, potrebno je bilo koristiti vlastiti server za sinkronizaciju. A kako ovaj način implementacije nema metodu to nije bio previše kompleksan zadatak. Za to postići korišten je postojeći poslužiteljski sloj na kojem se stvorio jednostavan Websocket. Zbog tog razloga je i kod klijentskog sloja puno jednostavniji. Kao prvo nije bilo potrebno uvesti većinu biblioteka kao prije, samo osnovne za tekstni okvir i alate kao i prije. Jedina stvar koja je bila dodana kod koda JavaScript je implementacija Websocket za sinkronizaciju. To je obavljeno kao u slici 34 gdje se definirala ruta poslužiteljskog sloja na kojoj se nalazi i vrsta podataka koji se prenose. Nakon toga se još i definirale pomoćne funkcije za povezivanje.

```javascript
this.socket = new WebSocket('ws://localhost:8000/ws');
this.socket.onmessage = (event) => {
  const json = JSON.parse(event.data);
  this.editor.commands.setContent(json);
};
```

Slika 34. Kod ostvarenja sinkronizacije teksta, uslikano iz koda aplikacije

A na ostatak koda uključujući i CSS se nije moralo dodati ništa više. CSS je opet ostao isti kao i kod metode CRDT jer je odabran isti izgled kao i kod ostalih metoda.

4.4. Usporedba implementiranih metoda

U ovoj aplikaciji su implementirane dvije metode, a to su metode CRDT i locking, a uz njih je i implementirana opcija bez metode. Na temelju iskustva njihove implementacije u toj aplikaciji i njihovog korištenja tijekom testiranja će se usporediti na temelju različitih kriterija. Sa strane programera će se pogledati lakoća implementacije, dok će se sa strane korisnika testirati njihovu funkcionalnost.

4.4.1. Usporedba prema težini implementacije

Što se tiče težine implementacije najteža je definitivno metoda OT. Kad se počelo sa stvaranjem metoda u aplikaciji, krenulo se od pretpostavljeno najteže metode a to je bila OT. Za razliku od metode locking koja ima jednostavnu primjenu i metode CRDT za koju postoje biblioteke za implementaciju, kod metode OT nema direktna biblioteka kojom se može olakšati rad. Iako postoje biblioteke servera na kojima se može pokrenuti OT, one nisu bile kompatibilne s izabranim uređivačem teksta. Zbog tog manjka opcija kompatibilnosti je metoda OT bila najteža za isprogramirati i zašto se na kraju ne nalazi u aplikaciji.

Od metoda kojih su se uspjeli dodati, najteža je bila metoda CRDT. Nakon što se nije uspjelo dodati metodu OT, pokušaj se prebacio na drugu po pretpostavljenoj težini implementacije. Prilikom prvog pokušaja stvaranja aplikacije se koristio tekstni uređivač Quill za pisanje. Quill je bio prvi izbor zbog svoje lakoće implementacije i mnoštvo alata za uređivanje teksta kojih je moguće lagano ugraditi. No nakon nekoliko neuspjelih pokušaja implementacije metode CRDT se prešlo na uređivač teksta TipTap. Iako je TipTap lagano ugraditi u aplikaciju kao i Quill najveći nedostatak su mu alati za uređivanje teksta. Jer TipTap za razliku od Quill nema već ugrađenu alatnu traku nego se alati moraju dodatno dodati. A taj proces dodavanja alata je puno sporiji i kompleksniji, te zahtjeva puno više koda u odnosu na Quill, kod kojeg se to može u samo nekoliko linija. To je zato što TipTap nema prirodnu alatnu traku na kojoj se nadodaju alati, nego se moraju jedan po jedan svaki posebno ugraditi što može potrajati i nepotrebno povećati broj linija koda aplikacije. No najveća prednost TipTap u odnosu na ostale uređivače teksta poput Quill je njegova sposobnost laganog uvođenja kolaborativnog pisanja. TipTap podržava bogatu količinu kolaborativnih biblioteka kojima se brzo i jednostavno omogućava zajedničko pisanje. Osim toga ima i mogućnost korištenja biblioteka Yjs i Hocuspocus koji omogućavaju korištenje

metode CRDT za rješavanja konflikta. Zbog tih razloga je TipTap bio očit izbor za ovakvu vrstu aplikacije. Jer iako se gubi na kvaliteti alata za uređivanje teksta, kod ovakve vrste aplikacije kojoj je glavni cilj kolaborativno pisanje nije potrebna prevelika opcija uređivanja teksta. Na temelju dokumentacije sa službene stranice TipTap-a su bile sve potrebne informacije za stvaranje vlastitog kolaborativnog uređivača teksta, te se na temelju toga i ova metoda implementirala u aplikaciju.

Nakon što se završilo programiranje metode CRDT prešlo se na metodu locking. Kako bi se olakšala implementacija i ostavila slijednost koda da bi bilo što lakše usporediti metode, odlučeno je koristiti isti uređivač teksta i biblioteke iz implementacije metode CRDT. Iako se radi toga još uvijek nalazi metoda CRDT u pozadini koda, zbog načina rada metode locking gdje se izbjegava konflikt umjesto rješavanja ne dolazi se do korištenja metode CRDT. Jer metoda locking upravo radi tako da ga izbjegava, bilo ju je moguće samo nadograditi na temelju metode CRDT. Pošto nije postojala biblioteka kod TipTap-a koji već podržava metodu locking moralo se iznova ugraditi. Najveći izazov je bilo osigurati da se zaključaju svi uređivači teksta korisnika koji ne pišu, a omogućiti onome koji piše da to radi nesmetano. Zbog toga se potrošilo mnogo vremena dok se logika nije sasvim dobro napravila da bi radila konstantno. Na kraju je bila lakša implementacija nego metode CRDT, ali to je zbog podloge koja je ta metoda dala za stvaranje ove.

Nakon metode locking se odlučilo dodati i opciju bez metode kako bi ih se sve moglo usporediti i s uređivačem teksta koji ne koristi nikakvu metodu rješavanja konflikta. Pošto je cilj bio napraviti sinkronizaciju samo bez metode nije bilo moguće koristiti server Hocuspocus koji se koristio u prijašnje dvije metode jer on već sadrži ugrađenu metodu CRDT. Zbog toga se koristio obični Websocket koji se ugradio na postojećem poslužiteljskom sloju. Zbog toga je ovo bilo najlakši dio koda za stvoriti u usporedbi s metodama. Opet se koristio TipTap zbog slijednosti koda s ostalim metodama. A uz njega bilo je potrebno samo ostvariti jednostavnu sinkronizaciju pisanog teksta.

Od svih metoda najteža za implementirati je definitivno bila metoda OT, što se i vidi podatkom da se na kraju nije uspjela dodati u aplikaciju. Ali to je i bilo za očekivati jer je općenito i način rada te metoda najkompliciraniji. Zbog toga i nema puno manjih uređivača teksta koji koriste metodu OT za rješavanje konflikta. Metoda CRDT je bila najteža od obadvije uspješne za dodati u aplikaciju. Ali s pomoću postojećih biblioteka

i opcija bilo ju je moguće uspješno ugraditi. Zbog toga je i ta metoda sve češće korištena, pogotovo kod manjih uređivača teksta kao TipTap. Iako nije bilo preteško dodati i metodu locking to je većinom zbog korištenja metode CRDT kao podlogu. No najveći njeni manjak nije prilikom njenog ugrađivanja u aplikacije nego kod njenog načina rješavanja konflikta zbog čega se manje koristi.

4.4.2. Usporedba prema iskustvu korištenja

Iako se nije uspjelo dodati metodu OT u aplikaciju ona se koristi u dosta postojećih uređivača teksta kao Google Docs, pa je moguće dati neke kritike o njoj sa strane korisnika. No ne bi bilo pravedno uspoređivati rad metoda u stvarnom programu koji koriste milijuni korisnika s radom u jednostavnoj aplikaciji koju je stvorila jedna osoba. Pa se neće ulaziti u kritike rada metode OT u usporedbi s ostalima.

Što se tiče metode CRDT, nedvojbeno je najbolja od svih drugih implementiranih metoda kad se gleda sa strane korisnika. Prilikom pokretanja programa lokalno i testiranjem kolaborativnog pisanja između dvije osobe nije se dogodio nikakav problem. Proces pisanja je bio lagan i brz za oba spojena korisnika. Za razliku od drugih metoda bilo je moguće istovremeno pisanje na dvije različite lokacije u tekstu bez ikakvih poteškoća. Prilikom provođenja primjera konflikta iz prijašnjeg teksta se nije dogodio nikakvo loše sinkroniziranje ili gubitak slova. Prilikom pokušaja istovremenog pisanja više korisnika na isto mjesto bi ih program napisao prema redoslijedu unosa, makar bili razmaknuti samo nekoliko milisekundi. Te tijekom toga nije niti jedanput progutao slovo, odnosno ne unio unos obadva korisnika.

Prilikom testiranja metode locking nije također bilo problema konflikta. No to je zbog njenog načina rada. Bilo je lagano pisati i zbog zaključavanja nije bilo moguće stvaranja konflikta. Zbog toga je bilo moguće lagano stvarati tekst, no bilo je potrebno pisati jedan po jedan čime se gubi doživljaj kolaborativnog pisanja. Prilikom pokušaja istovremenog pisanja više korisnika bi uvijek dala prednost jednom korisniku koji smatra da je prije kliknuo dugme, pa bi zbog toga progutao tekst jednog od korisnika i samo prikazalo tekst drugoga. Zbog toga nije najbolja metoda rješavanja konflikta i zato se ne koristi toliko često u stvarnoj primjeni. Iako se razvila i verzija metode gdje se ne zaključa čitav tekstni okvir nego samo dio teksta na kojem se piše, ipak postoji mogućnost gubitka teksta u slučaju da više korisnika odluči istovremeno pisati na tom istom dijelu teksta.

Na kraju se u program dodala opcija pisanja bez korištenja metode. U usporedbi s metodama je sigurno najgora. Iako za razliku od metode locking omogućava istovremeno pisanje, zbog načina sinkroniziranja postoji velika mogućnost gubitka teksta. Zato što ne koristi metodu, tekst se samo preslikava s jednog korisnika na drugoga. Zbog toga načina rada, kad se jedan korisnik spoji na dokument, on neće vidjeti tekst koji je neki drugi korisnik već napisao sve dok se ne dogodi promjena teksta. Ako već spojeni korisnik napravi promjenu, novome će se sve pravilno sinkronizirati i prikazati. Ali ako taj novi korisnik napravi bilo kakvu promjenu na njegov prazan tekst, taj tekst će se čitav sinkronizirati na drugoga koji već ima nešto napisano i sve mu zamijeniti s novim. Što se tiče pokušaja pisanja zajedno, iako je moguće, ne može se pisati na više mjesta istovremeno. To je zbog jednostavne implementacije gdje svi korisnici dijele jedan pokazivač pisanja. Zbog toga bi se moralo čekati da jedan završi kako bi drugi pisao, osim ako obojica misle pisati na tom istom mjestu gdje svaki piše slovo po slovo. A ako se dogodi situacija da istovremeno više korisnika napišu neku riječ, njena slova se upišu vremenskim redoslijedom kojim su upisani, makar većinom vremena jer postoji velika mogućnost gubitka slova.

Na kraju od svih implementiranih metoda je bilo najbolje pisati s pomoću metode CRDT. Prilikom pisanja teksta kod nje, nije se dogodio nikakav problem, čak i prilikom pisanja istovremeno. A u slučaju da se unese tekst na isto mjesto se sav upiše bez problema. To ju i čini najboljom metodom od obadvije implementirane. Metoda locking je bila prihvatljiva tijekom testiranja. Iako ima svoje poteškoće zbog načina rada, na kraju kvalitetno radi ono što namjerava. No nije najbolji izbor za rješavanje konflikta. Na kraju kad se one usporede s pisanjem bez metode se vidi kako je ipak bolje pisanje s bilo kojom metodom nego bez ijedne.

5. Zaključak

U ovom radu se definiralo točno što je konflikt u kolaborativnom pisanju. Definirane su i detaljno objašnjene najkorištenije metode rješavanja konflikta. Zatim su implementirane dvije od tih metoda zajedno s opcijom bez metode i testirane na različite načine. Na temelju toga su uspoređene i navedene prednosti i mane svake metode.

Nakon testiranja svih metoda je zaključeno da je metoda CRDT najbolja metoda rješavanja konflikta. Ona je jedna od samo dvije navedene koje direktno rješavaju konflikt kako nastaje. Dok druge metode ih ili izbjegavaju kao locking ili samo daju korisniku za rješavanje. Jednine metode koje ih direktno rješavaju su metode OT i CRDT. Iako se metoda OT može smatrati jednakom po načinu rješavanja konflikta s metodom CRDT, njezina težina implementacije joj smanjuje korisnost. Iako danas većina velikih uređivača teksta koriste metodu OT, sve više novijih uređivača teksta se odlučuje za CRDT čime je u porastu korištenja. Zbog tih razloga se CRDT potvrđuje kao najbolja metoda rješavanja konflikta u kolaborativnom uređivanju teksta.

Ako opet pogledamo hipotezu koja je definirana na početku gdje se pretpostavilo da metodama možemo u potpunosti riješiti nastale konflikte i omogućiti korisnicima lagano pisanje, možemo reći da je djelomično točna. Prilikom korištenja metoda CRDT i OT je ta hipoteza u potpunosti točna. Jer oni točno to i omogućavaju prilikom pisanja. No to ne vrijedi za sve metode. Iako ostale rješavaju konflikt, ne rade to na način koji omogućava lagano pisanje korisnicima. Zbog toga je ova hipoteza samo točna za određene metode.

Sažetak

U ovom radu su opisane metode rješavanja konflikta prilikom kolaborativnog pisanja. Definira se što su to konflikti i kako nastaju prilikom zajedničkog pisanja. Detaljno se opisuje svaka metoda rješavanja konflikta i pokazuje način rada kroz jednostavne primjere. Pokazuje se pokušaj implementacije tih metoda na stvarnom kodu gdje se pokreću i testiraju. Na temelju toga se detaljno uspoređuju i navode prednosti i nedostatci svake metode. Na kraju se definira najbolja metoda i objašnjavaju razlozi njenog odabira.

Ključne riječi: kolaborativno pisanje, konflikt, metode rješavanja konflikta, OT, CRDT, Locking

Abstract

This thesis describes methods of conflict resolution during collaborative writing. It defines what conflicts are and how they occur during joint writing. It describes each method of conflict resolution in detail and shows how they work through simple examples. An attempt to implement these methods on real code is shown where they are run and tested. Based on this, the advantages and disadvantages of each method are compared in detail. At the end, the best method is defined and the reasons for its selection are explained.

Keywords: collaborative writing, conflict, conflict resolution methods, OT, CRDT, Locking

Literatura

[1] „Studying Effectively“, Britannica, https://www.nottingham.ac.uk/studyingeffectively/writing/writer/whywrite/index.aspx, datum pristupa: 19.08.2024.

[2] T. Wolter, “Utnapishtim”, Britannica, https://www.britannica.com/topic/Utnapishtim, datum pristupa: 19.08.2024.

[3] R. Bhateja, “Why Did We Start Writing?”, Medium, https://medium.com/practice-in-public/why-did-we-start-writing-a774cdb24441, datum pristupa: 19.08.2024.

[4] T. Bednjanec, M. M. Pejić, “Razvoj pisma”, https://edutorij-admin-api.carnet.hr/storage/extracted/efe6be83-aab0-4849-a20d-065172b7f7c0/razvoj-pisma.html, datum pristupa: 20.08.2024.

[5] M. Ray, “papyrus”, Britannica, https://www.britannica.com/topic/papyrus-writing-material, datum pristupa: 20.08.2024.

[6] R. Lewis, “typewriter”, Britannica, https://www.britannica.com/technology/typewriter, datum pristupa: 21.08.2024.

[7] K. T. Hanna, “QWERTY keyboard”, TechTarget, https://www.techtarget.com/whatis/definition/QWERTY-keyboard, datum pristupa: 21.08.2024.

[8] M. Bellis, “WordStar Was the First Word Processor”, ThoughtCo, https://www.thoughtco.com/wordstar-the-first-word-processor-1992664, datum pristupa: 22.08.2024.

[9] “What is a word processor?”, Microsoft, https://www.microsoft.com/en-us/microsoft-365/word/word-processor, datum pristupa: 22.08.2024.

[10] L. Kramer, “How to Write Collaboratively”, Grammarly, https://www.grammarly.com/blog/how-to-write-collaboratively/, datum pristupa: 23.08.2024.

[11] W. Fleming, “Collaborative Writing”, Open Oregon, https://openoregon.pressbooks.pub/lbcctechwriting/chapter/5-1-collaborative-writing/, datum pristupa: 23.08.2024.

[12] R. Miguel, “The Evolution of Text Editors and Word Processors”, Linkedin, https://www.linkedin.com/pulse/evolution-text-editors-word-processors-miguel-rivas-perez-ilwbc, datum pristupa: 23.08.2024.

[13] “Zajednički razvijte svoje najbolje ideje u programu Google Docs”, Google, https://www.google.com/docs/about/, datum pristupa: 24.08.2024.

[14] “Collaborative Writing”, Vanderbilt University, https://www.vanderbilt.edu/bold/tools/collaborative-writing/, datum pristupa: 25.08.2024.

[15] M. Gala, “The enigma of Collaborative Editing”, Medium, https://medium.com/@mehulgala77/concurrent-collaborative-editing-d10192e55d2e, datum pristupa: 25.08.2024.

[16] A. Y. Wang, Z. Wu, C. Brooks, S. Oney, “Don't Step on My Toes: Resolving Editing Conflicts in Real-Time Collaboration in Computational Notebooks”, AIModels.fyi, https://www.aimodels.fyi/papers/arxiv/dont-step-my-toes-resolving-editing-conflicts, datum pristupa: 26.08.2024.

[17] “Implementing Real-Time Collaborative Editing in Web Applications”, Redstone, https://redstone.software/implementing-real-time-collaborative-editing-in-web-applications, datum pristupa: 27.08.2024.

[18] A. Verma, “The Basics of Operational Transformation”, Dev, https://dev.to/sudoboink/the-basics-of-operational-transformation-288j, datum pristupa: 27.08.2024.

[19] A. Zagorskii, “Operational Transformations as an algorithm for automatic conflict resolution”, Medium, https://medium.com/coinmonks/operational-transformations-as-an-algorithm-for-automatic-conflict-resolution-3bf8920ea447, datum pristupa: 28.08.2024.

[20] J. Day-Richter, “What’s different about the new Google Docs: Conflict resolution”, Google Drive Blog, https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs_22.html, datum pristupa: 28.08.2024.

[21] A. Herron, “To OT or CRDT, that is the question”, tiny, https://www.tiny.cloud/blog/real-time-collaboration-ot-vs-crdt/, datum pristupa: 30.08.2024.

[22] “About CRDTs”, CRDT.tech, https://crdt.tech/, datum pristupa: 30.08.2024.

[23] R. Mazzarini, “cola: a text CRDT for real-time collaborative editing”, nomad, https://nomad.foo/blog/cola, datum pristupa: 30.08.2024.

[24] G. Litt, S. Lim, M. Kleppmann, P. van Hardenberg, “Peritext A CRDT for Rich-Text Collaboration”, ink & switch, https://www.inkandswitch.com/peritext/#plaintext-insertions, datum pristupa: 31.08.2024.

[25] G. Litt, S. Lim, M. Kleppmann, P. van Hardenberg, “Peritext: A CRDT for Collaborative Rich Text Editing”, ACM Digital Library, Nov 2022. https://groups.csail.mit.edu/sdg/pubs/2022/Peritext_PACM_HCI_2022.pdf, datum pristupa: 01.08.2024.

[26] S. Citro, J. McGovern, C. Ryan, “Conflict Management for Real-Time Collaborative Editing in Mobile Replicated Architectures”, RMIT University, https://dl.acm.org/doi/pdf/10.5555/1273749.1273763, datum pristupa: 01.09.2024.

[27] “FastAPI”, FastAPI, https://fastapi.tiangolo.com/, datum pristupa: 04.09.2024.

[28] “WebSockets”, FastAPI, https://fastapi.tiangolo.com/advanced/websockets/, datum pristupa: 04.09.2024.

[29] “OAuth2 with Password (and hashing), Bearer with JWT tokens”, FastAPI, https://fastapi.tiangolo.com/tutorial/security/oauth2-jwt/, datum pristupa: 04.09.2024.

[30] “What is Vue?”, Vue.js, https://vuejs.org/guide/introduction.html, datum pristupa: 04.09.2024.

[31] “The editor suite to build products with”, TipTap, https://tiptap.dev/, datum pristupa: 04.09.2024.

[32] “Make your editor collaborative”, TipTap Docs, https://tiptap.dev/docs/collaboration/getting-started/overview, datum pristupa: 04.09.2024.

[33] F. Kukić, “Diplomski_rad_FK ”, GitHub repozitoriji aplikacije, https://github.com/FilipK19/Diplomski_rad_FK.git

[34] F. Kukić, Figma dizajn slika https://www.figma.com/design/ZAYdQC3957bSwrACV0ceQ7/Untitled?node-id=3-130&t=56jjUBvkO2IPnK2s-1

Popis slika