TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Reaalajas failide haldamine lõppkasutaja poolse sünkroniseerimisega kasutades tekstiredaktorit

bakalaureusetöö

Üliõpilane: Ivo Uutma
Üliõpilaskood: 112150IAPB
Juhendaja: Raul Liivrand

Tallinn
2014

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

---------- ----------

(*kuupäev*) (*allkiri*)

Annotatsioon

Antud töö eesmärgiks on luua failide haldamiseks ning reaalajas teistega muutmiseks tekstiredaktorile lisand. Lahenduses on kasutusel ka server, mille eesmärgiks on kasutajaid ühendada omavahel. Ise server faile ei hoiusta ega teosta ka sünkroniseerimist. Sünkroniseerimise jaoks luuakse mehhanism, millega oleks garanteeritud, et kõikide kasutajate failid on identsed.

Rakenduse olulisemateks osadeks on projekti failide vahendamine, reaalajas failide sisu sünkroniseerimine ning kasutaja tegevuste tõlgendamine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 13 joonist, 2 tabelit.

Abstract

The aim of this thesis is to create add-on for the text editor to make it a collaborative real-time editor with what users can manage their files with other users and also edit them in real-time.

This solution also uses server, which only purpous is to connect users it does not store the files or take part in the synchronization proccess. For synchronization an algortihm is made what guarantees that all users' files are identical.

Main components of this software are: managing files, real-time synchronization and the interpretation of user activities.

The thesis is in Estonian and contains 35 pages of text, 6 chapters, 13 figures, 2 tables.

Lühendite ja mõistete sõnastik

JSON	***JavaScript Object Notation*** Andmevahetusformaat, mis kasutab võti-väärtus paari elementide kirjeldamiseks [1].
Lõim	***Thread*** Programmiosa, mis on ülejäänud programmist sõltumatult iseseisev [1].
Base64	***Base64*** Formaat, kus kahendkujus andmed on teisendatud tekstilisele kujule [1].
API	***Application programming interface*** Rakendusliides, millega on võimalik tarkvara erinevaid osi kasutada [1].
IDE	***Integrated development environment*** Tarkvarasse sisse ehitatud programmeerimiskeskkond, milles võib sisalduda võimalus koodi kirjutada, koodi kompileerida ja seda siluda [1].
SHA	***Secure Hash Algortihm*** Räsialgoritm, millega sisend teisendatakse sõnumireferaadiks [1].
TCP	***Transmission Control Protocol*** Edastusohje protokoll, mis pakub usaldusväärset andmevahetust ning sõnumite korrektselt kohale jõudmist [1].
ACK	***Acknowledgement*** Signaal, mida kasutatakse andmevahetusel, et teada anda kaaslasele, et tema sõnum jõudis korrektselt kohale [1].

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Antud töös luuakse tekstiredaktorile Sublime Text juurde lisamoodul, millega saadakse mitme avatud redaktori vahel projektifailide sisu jagada ning reaalajas sünkroniseerida.

Rakendus on jagatud kliendipoolseks osaks ja serveripoolseks osaks. Kliendipoolse osa ülesandeks on kasutajale muudatuste kuvamine, muudatuste kogumine ja sõnumite saatmine serverile. Serveri ülesandeks on kasutajate saadetud sõnumitele vastamine või nende edasisaatmine teistele kasutajatele.

1.1 Taust ja probleem

Antud rakenduse jaoks tekkis vajadus, kuna tänapäeval on paljud ettevõtted ülemaailmsed ning korraga tuleb infot vahetada mitmes masinas reaalajas ehk sisuliselt on tegemist kaugpaarisprogrammeerimise liidesega. Rakenduse loojal tekkis vajadus selle järele, sest tema töökohas tuli tarkvara arendada Tallinna kontoris, aga skripti sai käima panna Prahas asuvas masinas ning senini oli vaja pärast igat muudatust uuesti fail üles laadida, mis oli küllaltki tülikas.

Seniste lahenduste puuduseks oli asjaolu, et paljud neist kasutasid sünkroniseerimiseks meetodit, kus failid salvestatakse kolmanda osapoole serverisse ja selle kaudu uuendatakse ühenduses olevaid kliente. Selline lahendus, aga läheb vastuollu paljude ettevõtete reeglitega. Need lahendused, kus aga kasutatakse lõppkasutaja poolset sünkroniseerimist on tihti loodud selliselt kus korraga saab redigeerida ainult üks osapool.

Tähtis on ka see, et ei oleks vaja uut redaktorit selle lahenduse jaoks ega ka veebiliidest, kuna arendajatele muutuvad teatud redaktorid meelepärasemaks ning nende väljavahetamine ei ole alati hea ehk parem on lisada olemasolevale redaktorile liides juurde, millega säilitatakse redaktori kõik kasulikud omadused, aga lisatakse veel võimalus üle võrgu faile jagada.

1.2 Ülesande püstitus

Eesmärgiks on luua kaugpaarisprogrammeerimise rakendus, kus on teostatud nii projekti failide jagamine ühendamisel kui ka reaalajas failide muutmine. Failide muutmisel toimub

sünkroniseerimine lõppkasutaja poolel, serveri eesmärgiks on ainult sõnumeid õigetele kasutajatele edastada. Erinevate kasutajate muudatused peavad olema eristatavad redaktori vaates. Väikeseks eesmärgiks rakenduse juures on ka see, et selle kasutamine ja seadistus peaks peab olema võimalikult lihtne ja kiiresti õpitav.

1.3 Metoodika

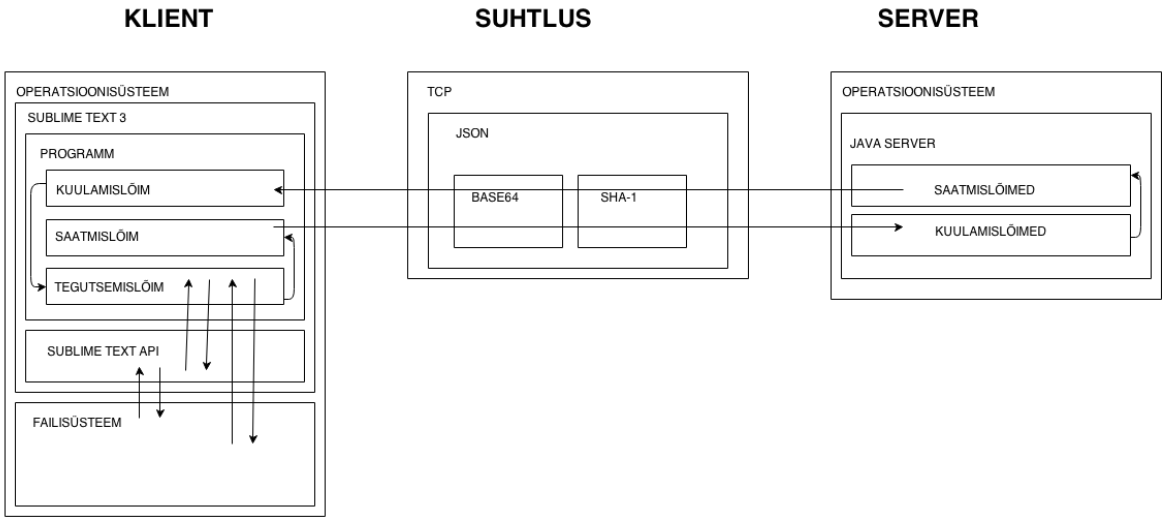
Eesmärkideni jõudmiseks kasutatakse olemasolevat redaktorit Sublime Text ning selle APIt. Nende kahe liidese abiga ehitatakse kliendi poolne osa rakendusest kasutades programmeerimiskeelt Python. Klientide ühendamiseks moodustatakse programmeerimiskeeles Java kirjutatud programm, mis oma olemuselt on server, mis kuulab teatud porti ning võtab vastu ja edastab kasutajate saadetud sõnumeid. Kliendi ja serveri vaheline suhtlus käib kasutades TCP protokolli ning saadetavad sõnumid on JSON formaadis. Tervete failide edastamiseks kasutatakse base64 formaati, sest JSON standard ei toeta kahendkujul andmete saatmist.

1.4 Ülevaade tööst

Töö jaotub nelja ossa. Teises peatükis kirjeldatakse töö arhitektuuri ning selles kasutatavaid tehnoloogiad ja kuidas need rakenduses kasutusel on. Kolmandas peatükis räägitakse täpsemalt rakenduse käivitamisel toimuvatest tegevustest ja rollidest. Neljandas peatükis kirjeldatakse rakenduse põhitsüklit ehk reaalajas failide muutmist. Viiendas peatükis vaadatakse kuidas rakenduse sulgemisel teiste kasutajatega käitutakse.

2. Rakenduse arhitektuur ja tehnoloogia

2.1 Rakenduse arhitektuur



Joonis 1. Rakenduse arhitektuur.

Rakenduse kliendi osa on ehitatud tekstiredaktori Sublime Text 3 peale, kasutades selle Pythoni API-t, mis on loodud pluginate tegemiseks. Serveriks on Java programm, mille ülesandeks on kliente omavahel ühendada. Serveri ja kliendi vaheline suhtlus käib kasutades TCP-d ning sõnumid kodeeritakse JSON formaati. Sõnumites on binaarsed andmed teisendatud base64 formaati.

2.2 Sublime Text 3

Sublime Text on mitmeplatvormiline tekstiredaktor, mis töötab Microsoft Windowsi, Linuxi ja Mac OS Xi peal. Programm on kirjutatud programmeerimiskeeles C++, mille peale on tehtud Pythonis API. Sublime Texti eeliseks teiste sarnaste redaktorite ees on selle sarnasus IDE-dega ning lai valik plugineid. Redaktor on saadaval ka kaasaskantava versioonina, mis tähendab, et installimise asemel on programm lahti pakitav ja asub kasutaja määratud kataloogis. Selle versiooni eeliseks on, et redaktori saab ühes arvutis endale sobivaks seadistada ja siis seda kõikides teistes masinates kasutada.

Redaktori esialgne versioon tuli välja 2008. aastal. Millele järgnes 2012. aastal praegune ametlik versioon Sublime Text 2. 2013. aastal tuli välja Sublime Text 3 beeta versioon, milles

vahetati python 2.7, python 3.3 vastu ning muudeti API funktsionaalsust, et kolmandate osapoolte pluginad ei saaks enam põhirakendust kokku jooksutada.

Sarnaselt IDEdele on selles redaktoris projekti haldamine lihtsaks tehtud. Redaktoris kasutatakse hägusloogikal põhinevaid algoritme ning selle abil leitakse üles projekti piires erinevates failides deklareeritud muutujad ja meetodid. [2]

Redaktoris on olemas võimalus nii kompileerida ja interpreteerida koodi ning seal samas see ka käima panna. Selleks kasutab Sublime Text operatsioonisüsteemi käsurida ning tegelikult käivad kõik kompileerimised selle kaudu, kuid kuna ta loeb sealt väljundid sisse, siis on kõik kohe editoris näha ning kasutajal pole ise vajadust redaktorist lahkuda. [3]

2.3 Olemasolevad paarisprogrammeerimise lisad

2.3.1 Floobits

Floobits on süsteem, kuhu on lisatud mitmed populaarsed tekstieditorid ja lisaks nendele ka veebiliides. Floobitsis on kasutusel *Emacs, Sublime Text, Vim, IntelliJ Idea.*

Floobitsi ideeks on, et kasutajad saaksid kasutada sellist redaktorit nagu nad sooviksid ja kogu rakendus on tsentraliseeritud. Jagatavaid faile hoitakse serveris kasutaja tööjaamas, mida uuendatakse kui kasutajad oma redaktorites muudatusi teevad ja sealt omakorda uuendatakse teiste ühenduses olevate kasutajate faile. Sellise lähenemise miinuseks on, et rakenduse kood asub kolmanda osapoole serveris ning ollakse sellest sõltuvad. Lisaks sellele tuleb teha ka sinna kasutaja.

Floobits ei ole tasuta rakendus, selle kasutamiseks peab ostma litsentsi, mida tuleb uuendada iga kuu. Maksumus sõltub sellest kui palju kasutajaid ja tööjaamu soovitakse. Alates viiest kasutajast ja kümnest tööjaamast, kuni 250. kasutajani ja 600 tööjaamani. Viimane variant on 40 korda kallim kui esimene. [4]

2.4 Sublime Text API

Rakenduse kliendi poolne osa on kirjutatud redaktori Sublime Text pluginana, kasutades selles olevat API-t. API kujutab endast meetodite kogumit, millega on võimalik teostada redaktoris selle spetsiifilisi tegevusi. Kogu liides jookseb eraldi lõimes, seega on välistatud võimalus, et

rakendus peab API järele ootama või vastupidi ning samuti ei liidesel võimalik redaktorit ennast kokku jooksutada. [5]

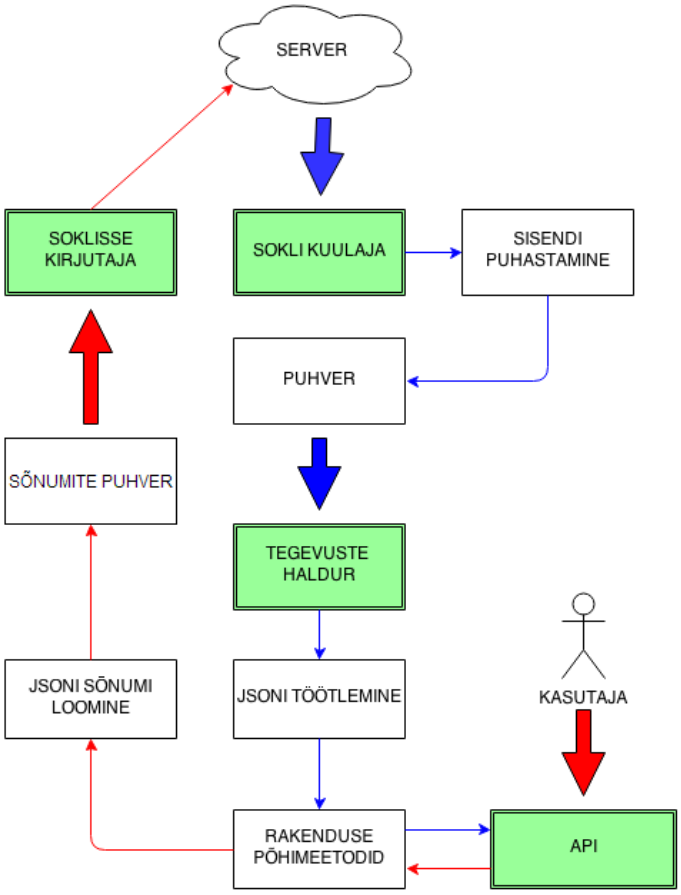
Selles projektis arendatavas rakenduses kasutatakse APIt muudatuste teada saamiseks ja muudatuste sisestamiseks.

Muudatuste lugemisel saadakse teada, kuidas kursor muutus ning selle järgi on võimalik teada saada kui palju teksti ja kuhu kirjutati. Samuti püütakse kinni tegevused nagu faili avamine, faili sulgemine ja faili salvestamine.

Muudatuste tegemisel kasutatakse mooduleid, mis võimaldavad faili sisu muuta, lisades või kustutades sealt teksti ning kasutatakse ka faili salvestamist.

2.5 Pythoni klient

Rakenduse kliendi poolne osa on kirjutatud pythonis, kasutades versiooni 3.3, mis on redaktoriga juba kaasas.

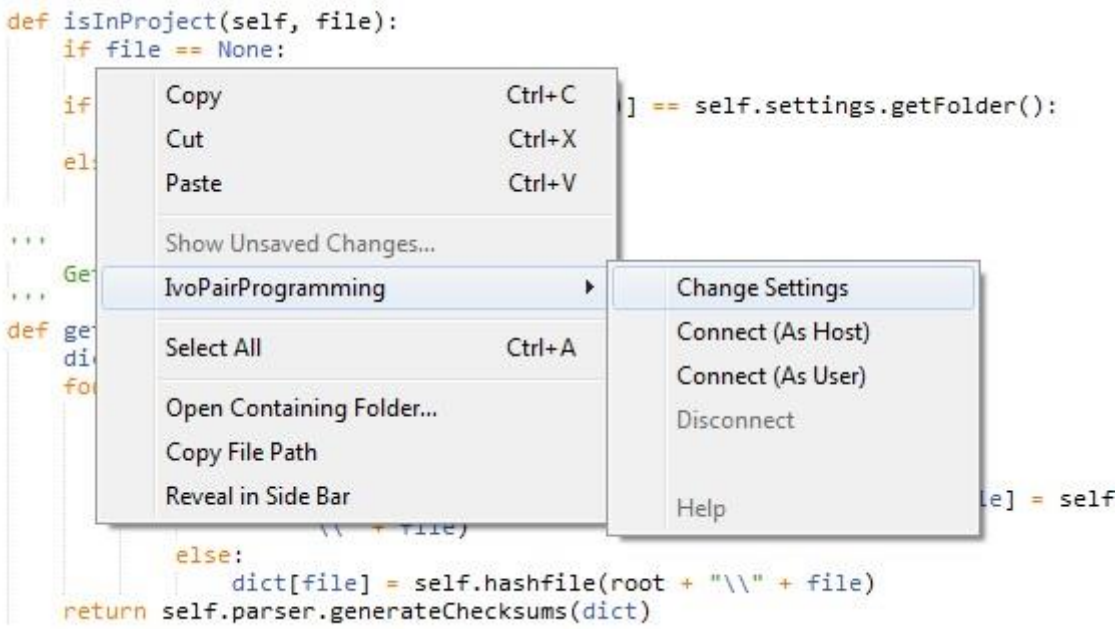


Joonis 2. Kliendi disain.

Joonisel 2 on kujutatud rakenduse kliendi disain. Joonisel on toodud rakenduses jooksvad lõimed välja rohelistena ning nende vahelised suhtlused on toodud välja eri värvi nooltega. Sinised nooled näitavad serverist sisse tulnud andmete voo liikumist ja punased välja mineva oma.

Programm on jaotatud neljaks alamosaks vastavalt funktsionaalsusele. Kaks neist on seotud serveriga suhtlemisega. Üks loeb serverist tulnud andmeid ja pärast sisendi puhastamist paneb puhvrisse ja teine loeb väljaminevate sõnumite puhvrist sõnumeid ja saadab serverisse. Põhitöö teevad ära tegevuste haldur ja API. Kõikide sissetulnud sõnumitega tegeleb tegevuste haldur, kes töötleb sõnumis olevaid andmeid ja täidab vastavat funktsionaalsust nagu sõnumis ettenähtud oli ning enamikel juhtudel kasutab ka API funktsioone. API lõimed on kasutusel kasutaja tegevuste kättesaamiseks, näiteks teada saamine, millal faili muudeti. Kui API-l on sisend olemas, siis kasutab ta sama funktsionaalsust nagu tegevuste haldurgi, aga kõik väljundid teisendatakse JSON-iks ja saadetakse serverile.

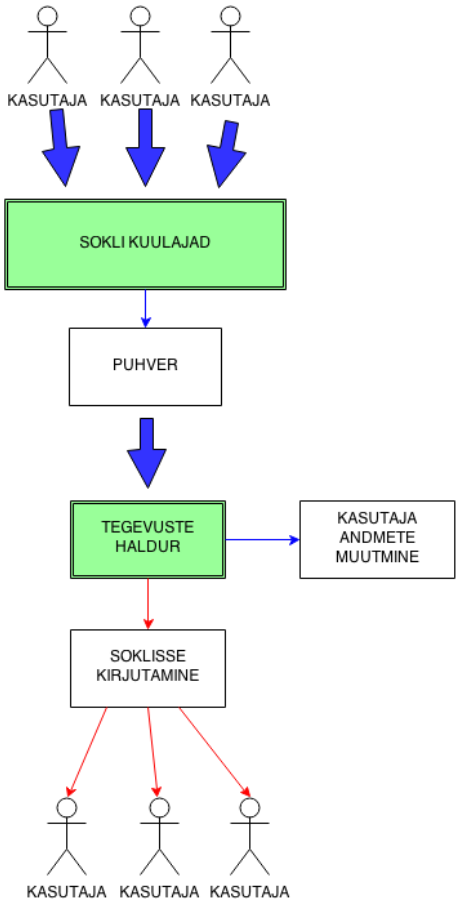
Rakendus jookseb tekstiredaktori sees, kasutades seal olevat *plugin_host* funktsionaalsust. Kogu kasutajaliides on samuti ehitatud redaktori peale, kasutades selle kontekstimenüüd ja võimalust teksti välimust muuta. Samuti edastatakse mõningat informatsiooni rakenduse olekuriba peal.



Joonis 3. Rakenduses kasutatav kontekstimenüü.

2.6 Java server

Rakenduse serveripoolne osa on valmistatud javas. Java on üks kahest maailma populaarseimast keelest ning oma olemuselt sobib serveri valmistamiseks hästi [6]. Pythoni puuduseks serveriks olemisel on see, et see on interpreteeritav keel ja seega kasutab GILi, mis lubab korraga joosta ainult ühel lõimel ehk kuigi rakendus on mitmelõimeline, jookseb see ikkagi ainult ühel protsessoril ühe protsessi kaupa. Kuna aga serveril on vaja tegeleda mitmete ühendustega korraga, siis võib see pudelikaelaks osutuda.



Joonis 4. Serveri disain.

Oma olemuselt on server väike programm, mis kuulab porti ja teeb vastavalt sisenditele tegevusi. Programm on jaotatud mitmeteks tükkideks vastavalt funktsionaalsusele. Põhilõim võtab vastu uusi kasutajaid ning määrab neid gruppidesse. Joonisel 4 on toodud välja programmi käitumine kasutajatega, kes on juba teatud grupiga liitunud. Joonisel on toodud rakenduses jooksvad lõimed välja rohelistena ning nende vahelised suhtlused on toodud välja eri värvi nooltega. Sinised nooled näitavad kasutajatelt sisse tulnud andmete voo liikumist ja punased välja mineva oma.

Igal kasutaja jaoks on loodud oma objekt, mis jookseb eraldi lõimena ja võtab vastu soklist tulnud andmeid ja paneb nad ühiskasutatavasse puhvrisse. Sealt edasi tegeleb nendega iga grupi jaoks loodud tegevuste haldur, kes vastavalt sõnumi sisule saadab selle edasi või muudab kasutaja objekti andmeid.

Server ise faile ei tee ega hoiusta. Kõik sissetulevad sõnumid, peale avatud failide loetelu sõnumi ja ühinemissõnumi, saadetakse teistele edasi.

Serveris hoiustatav info kasutaja kohta:

- **Kasutajanimi** – Luuakse kasutaja ühinemisel. Seda kasutatakse juhul kui on vaja edasta informatsiooni konkreetsele kasutajale.
- **Kasutaja roll –** Määrab ära kasutaja rolli grupis.
- **Kasutajal avatud olevate failide nimed –** Kasutatakse muudatuste edastamisel, kui kasutajal faili lahti pole, siis saadetakse muudatusi selle faili kohta ainult siis kui see salvestatakse või kasutaja liides ise küsib.

2.7 Serveri ja kliendi vahel suhtlus

Antud peatükis selgitatakse kuidas ja missugust tehnoloogiat kasutades rakenduse kliendid ja server omavahel suhtlevad.

2.7.1 Andmete edastamine TCP-ga

Server ja klient loovad omavahel ühenduse kasutades TCP-d. TCP ehk edastusohje protokoll on transpordi kihi protokoll, mida kasutades saavad ühenduses olevad kasutajad omavahel andmeid saata pideva voona [1].

TCP-l on mitmeid häid omadusi. Kõige tähtsam on selles rakenduses see, et kõik paketid jõuavad alati kohale. Lisaks andmete kohale jõudmisele on garanteeritud ka andmete õigus. Pakettide saatmisel on igal ühel kontrollsummad, mille järgi saab teada kas informatsioon oli moondunud, mille järel küsitakse see uuesti, et saada lõpuks kätte õige informatsioon. TCP-l on omadus ka andmeside kiirust muuta, mis suudab tasakaalustada mõlema poole ühendusi. [7]

Ühenduse loomine käib kolmepoolse kinnitusega, kus ühenduse algataja saadab selle rakenduse puhul teate serverile, et soovib ühendust luua, mille peale server saadab talle tagasi teate, et ühendus on loodud ning kasutaja annab serverile teada, et ta sai selle teate kätte. Pärast neid kolme sammu saab nende kahe osapoole vahel andmeid saata. [7]

2.7.2 Sõnumi formaat JSON-is

Kliendi ja serveri platvormilise erinevuse tõttu ei saa nad üksteisele programmeerimiskeele spetsiifilisi objekte saata. Selleks kasutavad nad JSON formaati.

JSON on andmevahetusformaat, mille idee seisneb selles, et inimestel oleks seda kerge lugeda ja kirjutada ning masinatel kerge töödelda ja luua. JSONil on kaks võimalikku formaati üks on võti-väärtus paaridega objekt ja teine on massiiv, kus sees on väärtused. Väärtusteks võivad olla stringid, numbrid, objektid, massiivid, tõeväärtused ja null. [8]

Selles rakenduses kasutatakse objekti formaati. Igal sõnumil on olemas päis ja keha. Sõnumi päiseks on sõnumi liik, mille järgi saab sõnumi vastuvõtja teada, missugusel viisil sõnumit töödelda. Sõnumi kehaks on alamobjekt, mis on igat liiki sõnumil erinev.

Server vaatab sõnumi saamisel päise järgi, et kas see oli talle saadetud või kas ta peab selle edasi saatma. Vastavalt sõnumi sisule otsustab server, kellele ta sõnumi edastab.

Kliendi poolel on tehnoloogiliselt sõnumi vastuvõtmine erinev, sest pythonil ei kuulata otse soklit, vaid võetakse andmeid puhvrist teatud osade kaupa. Sellest tulenevad sõnumite lugemisel probleemid. Esimene probleem tekib siis kui sõnum on pikem kui ettenähtud lugemishulk ehk sõnum tuleb ise kokku panna mitmest jupist. Teine probleem tekib siis kui puhvris on mitu sõnumit ja need loetakse sealt korraga välja, siis parser ei suuda kahte JSONi objekti korraga töödelda. Sellisel juhul need sõnumid tükeldatakse algoritmiga üksikuteks sõnumiteks tagasi, millest juba programm aru saab.

Rakenduses on loodud erinevad mallid iga tüüpi sõnumi kohta. Mallides kasutatakse pythonisse ja javasse sisse ehitatud funktsionaalsust JSONi loomiseks. Kliendi pool näeb välja lahendus selline, et tehakse vajalikud toimingud, siis antakse edastatava info vajalikud osad JSONi genereerijale, mis paneb sõnumi korrektsesse formaati ning pärast seda saab selle teise kasutaja poole teele panna.

Faile saadetakse samamoodi sõnumitena, aga kuna JSON ei toeta kahendkoodi formaati, teisendatakse failide sisu base64 formaati.

2.8 Failide teisendamine Base64 kodeeringusse

Base64 on kodeering, millega kuvatakse kahendkoodis andmeid teksti kujul. Seda kasutatakse tihti juhtudel kus on vaja hoiustada kahendkoodis andmeid aga nende ühest kohast teise ülekandmisel kasutatav protokoll on disainitud kasutama tekstipõhiseid andmeid.

Base64 aluseks on kooditabel (**vt** Tabel 2. Bas64 kooditabel kodeeringu määramiseks. **lk** 36), kus indeksiteks on kuue bitised numbrid ja neile on määratud vastavad väärtused [9].

Antud rakenduses kasutatakse teksti kodeeringuna UTF-8, mis tähendab, et ühe tähe pikkus võib olla üks kuni neli baiti. Kõik tähed teisendatakse kahendkujule ja liidetakse kokku üheks suureks stringiks. Edasi jagatakse suur string kuue bitilisteks juppideks, millele saab tabeli järgi väärtuse anda. Kui baitide arve ei jagu kolmega ehk viimases 24bitises blokis on ainult üks või kaks baiti, siis lisatakse juurde nullbaidid ning kodeerimisel teisendatakse puudu olevad kuue bitilised lõigud „=“ märkideks. [9]

Allolevas tabelis on näidatud, kuidas sõnast IVO saab pärast base64 rakendamist SVZP. Indeksi järgi kodeeringu leidmiseks on kasutatud kooditabelit (**vt** Tabel 2. Bas64 kooditabel kodeeringu määramiseks. **lk** 36),.

Tabel 1. Sõna viimine base64 kodeeringusse.

	I								V								O							
Bitid	0	1	0	0	1	0	0	1	0	1	0	1	0	1	1	0	0	1	0	0	1	1	1	1
Indeks	18								21				25				15							
Kodeering	S								V				Z				P							

Oluline on märkida veel ka seda, et iga kolme originaal baidi asemel tekib kodeerimisel neli baiti ehk andmemaht kasvab 33%. Uue andmemahu suuruseks on $n * \left(\frac{4}{3}\right)$, kus n on originaalteksti baitide arv.

2.9 SHA-1 algoritmiga kontrollsummade leidmine

SHA-1 on ühesuunaline räsifunktsioon, mille rakendamisel tekstile luuakse kindla pikkusega vaste ehk sõnumireferaat, mille pikkuseks selle algoritmi puhul on 160 bitti. Selline algoritm võimaldab määrata teksti terviklikkust. Iga muudatus tekstis loob väga suure tõenäosusega teistsuguse sõnumireferaadi. [9]

Rakenduses kasutatakse räsifunktsiooni failidele kontrollsumma leidmiseks, mida kasutatakse failide samasuse võrdlemiseks. Ühenduse loomisel on vaja kontrollida missugused failid on kasutajal ajakohased ja millised mitte. Selle jaoks arvutatakse kõikidel failidel kontrollsummad, mis saadetakse kasutajale kellel on kõik õiged failid olemas. Kontrollsummade võrdlemisel saab teada, millised failid on erinevad ja millised ei ole. Seejärel saab aegunud või puuduolevad failid kasutajale saata.

2.10 Sätete fail rakenduse seadistamiseks

Rakendusel on olemas sätete fail, kust loetakse rakenduse käivitamisel vajalikku infot. Failis on viis parameetrit, mida saab seadistada vastavalt vajadustele.

- **Projekti juurkataloog.** Kataloogi asukoht süsteemis. Kõiki faile, mis asuvad selles ja selle alamkataloogides on võimalik jagada ning teiste kasutajate poolt tulnud failid kirjutatakse sinna.
- **Serveri url/ip.** Serveri aadress, millega rakendus üritab ühendust luua.
- **Serveri port.** Pordi number millel rakendus jookseb serveris.
- **Grupi nimi.** Võõrustaja poolt valitud nimi. Nimi võib olla täiesti juhuslik. Ainsaks piiranguks on, et ühe serveri peal saavad korraga olla ainult unikaalsete nimedega grupid.
- **Grupi parool.** Grupiga liitumiseks vajalik parool, mille määrab võõrustaja.

Oma olemuselt on fail JSON formaadis, kus info esitatakse võtme-väärtus paarina. Kuid kuna on lisatud kasutajat abistavad kommentaarid, siis otse selle faili sisu töödelda ei saa. Vajalikku info kätte saamiseks eemaldatakse algoritmiga faili lugemisel ebasobivad read ning alles jäänud sisust saab juba parser aru.

Sätete fail ongi ainus seadistus, mis kasutajal tuleb teha ühenduse loomiseks. Faili saab avada kontekstimenüüst samuti avatakse fail juhul kui mingisuguse parameetri sisu ei vasta ühendamisel nõuetele, näiteks kui kasutaja eksis parooliga.

```
{
    // Project folder
    "RootFolder":"C:\\Users\\ivo\\Desktop\\testing\\",

    // Server ip/url 217.146.76.178
    "Host":"217.146.76.178",

    // Server port
    "Port":1129,

    // Group name
    "Group":"PairProgramming",

    // Group password
    "GroupPassword":"0000"
}
```

Joonis 5. Sätete fail.

3. Rakenduse käivitamine ja ühenduste loomine

Selles peatükis käsitletakse rakenduse käivitamistsüklit milleks on rakenduse avamine, ühendamine serveriga, projekti sünkroniseerimine.

Rakendust saab käivitada redaktoris kontekstimenüü kaudu. Käivitamisel saab valida kahe rolli vahel:

- **Võõrustaja.** Seda rolli kasutab kasutaja, kes soovib alustada uut sessiooni. Tema ühinemisel luuakse serveris uus grupp, millega kõik teised saavad liituda. Kõik teised liitujad küsivalt liitumisel tema faile.
- **Kasutaja.** Seda rolli kasutab kasutaja, kes ühineb sessiooniga. Pärast ühinemist sünkroniseeritakse tema failid võõrustaja omaga.

3.1 Serveriga ühenduse võtmine ja keskkonna seadistus võõrustajana

Võõrustaja seadistab sätete faili, seades grupile nime ja parooli. Seejärel saab võõrustaja serveriga ühendust võtta saates serverile vajalikud andmed, server seejärel kontrollib kas antud nimega grupp juba eksisteerib või mitte.

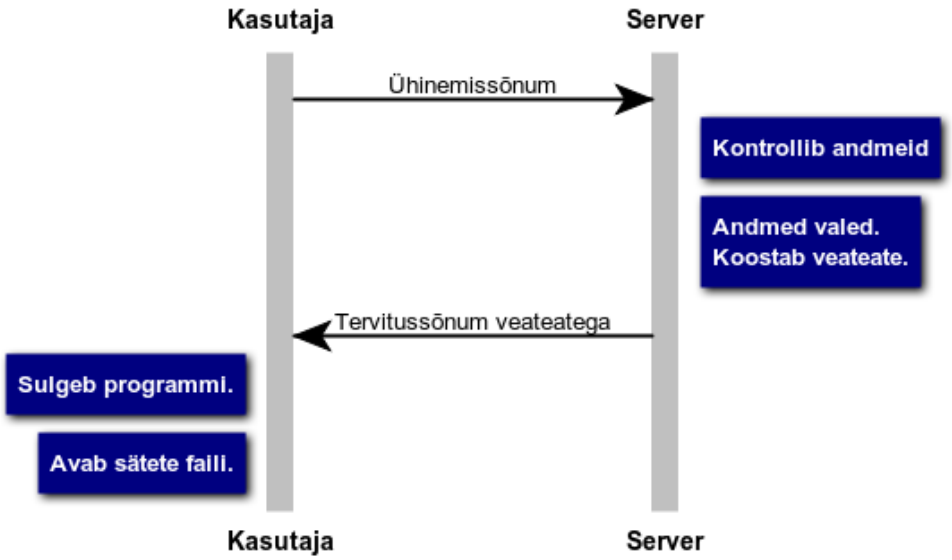
Grupi eksisteerimisel saadetakse vastus vastava veakoodi ja -teatega ning suletakse serveri poolt sokkel. Vastuse kätte saades kuvatakse kasutajale vastav veateade ning suletakse kõik programmi tööga kaasnenud ühendused ja alamlõimed ning avatakse sätete fail selle muutmiseks.

Kui valitud nimega gruppi pole veel loodud, siis server loob selle, pannes tööle alamlõimed, mis tegelevad sõnumite saatmistega ning haldamistega ja soklite kuulamisega. Pärast seda saadetakse võõrustajale tagasi tervitussõnum. Sõnum kätte saades kogutakse kokku tekstiredaktoris olevad avatud failid ja kontrollitakse, millised neist kuuluvad jagatavasse projekti ning seejärel saadetakse nende failinimede list serverile, mis seda hilisemateks tegevusteks hoiustab.

3.2 Serveriga ühenduse võtmine kasutajana

Kasutaja seadistab sätete faili, kirjutades sinna grupi nime ja sellele vastava parooli. Seejärel saadetakse sõnum serverile kes kontrollib andmete vastavust. Kahel juhul tagastatakse tagasi

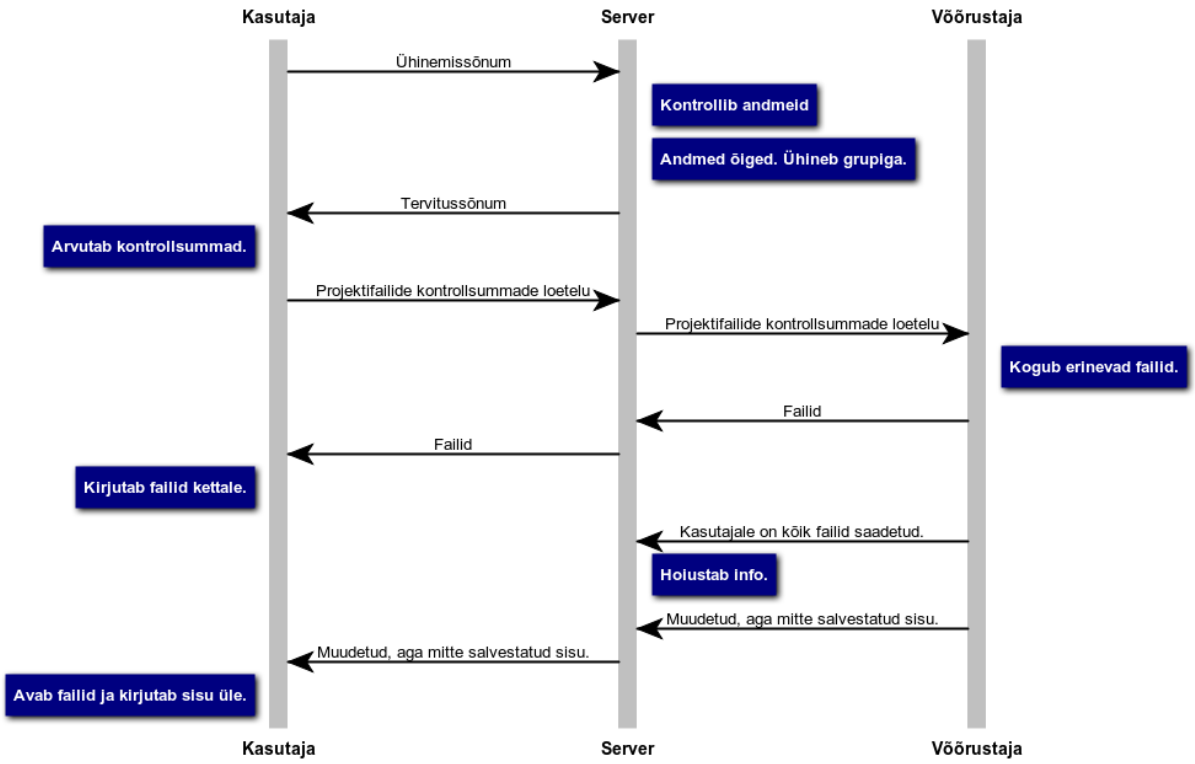
veateatega sõnum. Esimeseks juhuks on see kui gruppi ei eksisteeri ning teiseks juhuks on olukord kus grupp küll on olemas aga paroolid ei ühti. Mõlemal juhul suletakse kõik ühendused ja muud loodud lõimed ning kuvatakse vastav veateade ja avatakse sätete fail selle muutmiseks.



Joonis 6. Kasutaja liitumine vigaste andmetega.

Kui grupp eksisteerib ja ka parool ühtib, siis tagastatakse kasutajale tervitussõnum. Sõnumi kätte saades arvutatakse iga projektis oleva faili kontrollsummad SHA-1 algoritmiga ning saadetakse need serverile. Server saadab sõnumi edasi võõrustajale, kus võõrustaja arvutab enda failide kontrollisummad. Seejärel võrreldakse neid. Kontrollsummade erinemisel loetakse sisse faili sisu ning konverteeritakse need base64 formaati ning saadetakse vastavate abiparameetritega serverile, mis saadab selle edasi kasutajale, kus see dekodeeritakse ning fail kirjutatakse projekti kataloogi sama kataloogi struktuuriga nagu oli võõrustajal. Kui võõrustajal on kõik failid saadetud, siis saadetakse serverile sellest teavitus, et vajadusel kui võõrustajal peaks ühendus katkema saab see kasutaja ise võõrustajaks hakata.

Lisaks saadetakse teiste kasutajate poolt ka edasi failide sisud, mida on muudetud, aga kettale salvestatud veel pole. Tehnoloogiliselt on see sarnane failide enda saatmisega, et sisu teisendatakse base64 süsteemi ja lisatakse paar abiparameetrit nagu failinimi ja kasutajanimi. Kui kasutaja saab kätte sellise sõnumi, siis redaktoris avatakse see fail ja selle sisu kirjutatakse üle teiselt kasutajalt tulnud sõnumi sisuga.



Joonis 7. Kasutaja liitumine korrektsete andmetega.

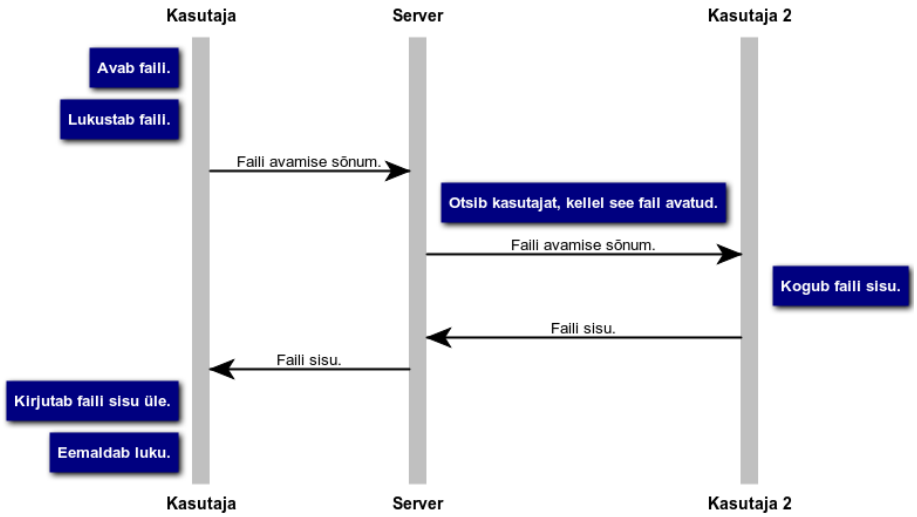
4. Reaalajas failide sünkroniseerimine ja info vahetamine

Selles peatükis käsitletakse rakenduse põhitöö tsüklit, kus töödeldakse juba otseseid kasutajate sisendeid, milleks on failide tegemine, avamine, muutmine, salvestamine.

4.1 Failide avamisel sisu uuendamine

Failide avamisel tekib vajadus kontrollida, kas selle sisu on ajakohane, sest suletud failide puhul uuendatakse selle sisu ainult salvestamisel. Selle probleemi lahenduseks hoiustab server kasutajate lahti olevaid faile ning kasutaja saadab lihtsalt sõnumi serverile, et soovib avatud faili uuendada ning server valib seejärel kasutaja, kellel samuti see fail avatud on ja küsib tema käest selle sisu, mille kasutaja siis saadab.

Saadud uue faili sisu kirjutatakse editoris olevaga üle, kettale seda ei salvestata, kuna see polnud ka teistel salvestatud. Oluline on siin ka veel see, et nii kaua kui oodatakse serverilt vastust, et kas uuendatakse või pole vaja, on faili sisu lukus ehk kasutaja ei saa sinna kirjutada. Niipea kui tuleb teade faili kohta, siis see ka vabastatakse.



Joonis 8. Faili sisu uuendamine, selle avamisel.

Serveri poolel ei ole loodud optimeerimist ehk failide ja kasutajate järjekorda pärast info lugemist ei muudeta, seega võetakse info esimeselt kasutajalt kellel see fail lahti on ehk kui see kasutaja oma faili ei sulge, siis uuendatakse alati tema faili järgi kuigi see fail võib ka mõnel teisel kasutajal lahti olla.

4.2 Failide muudatuste sünkroniseerimine

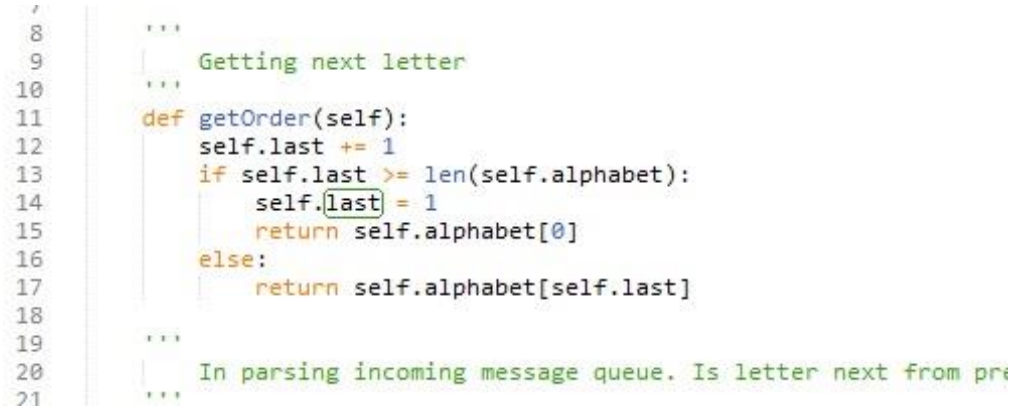
Faili muutmisena loetakse igasugust antud redaktori vaates avatud oleva teksti muutumist. Vaate muutumise kohta saab infot API kuulaja, milles järel otsitakse üles muudetud koht, mis siis vastavate abiparameetritega serverisse edastatakse.

Muudatuste asukoht otseselt ei ole teada, rakendus peab ise seda tuletama. Seda tehakse kursorite asukohtade järgi. Teades enne muudatuse tegemist asunud kursorite asukohti ja pärast muudatuse tegemist uusi asukohti, siis saab sealt teha järeldused, millal midagi kustutati või lisati. Näiteks on enne muudatuse tegemist kursorid kohtadel 0 ja 5 ehk teisisõnu olid faili viis esimest tähte selekteeritud kui nüüd API saab teate, et faili on muudetud loetakse uuesti kursori asukohad ja saadakse teada, et mõlemad kursorid asuvad kohal 0. Siit saab järeldada, et need viis tähte kustutati. Seega edastatakse sõnum, et kohtade 0 ja 5 vahel olev tekst tuleb kustutada ehk API liidese kaudu tühja stringiga asendada.

Rakenduses tuvastatakse alljärgnevaid muudatuste liiki:

- **Teksti lisamine.** Kasutaja lisas faili uut teksti.
- **Teksti kustutamine.** Kasutaja kustutab teatud osa tekstist.
- **Sulgude või ülakomade lisamine.** Kasutaja sisestab algussulu või ülakoma. Redaktor lisab automaatselt lõpetava sulu või teise ülakoma ning jätab kursori nende vahele.
- **Selekteerimisel sulgude või ülakomade sisestamine.** Kasutaja selekteeris teksti ning sisestas kas ülakomad või sulud, sellisel juhul redaktor paneb need selekteeritud teksti ümber.
- **Selekteerimisel selle teksti asendamine või kustutamine.** Kasutaja selekteeris teatud osa tekstist ja asendas selle mingi muu tekstiga või kustutas selle.
- **Undo või redo.** Kasutaja võtab oma tegevuse tagasi või taastab hilisema seisu

Erinevate kasutajate muudatused tuuakse esile kasutajaliideses neid üksteisest eristades, joonistades iga kasutaja viimasele tehtud muudatusele kastikese ümber.

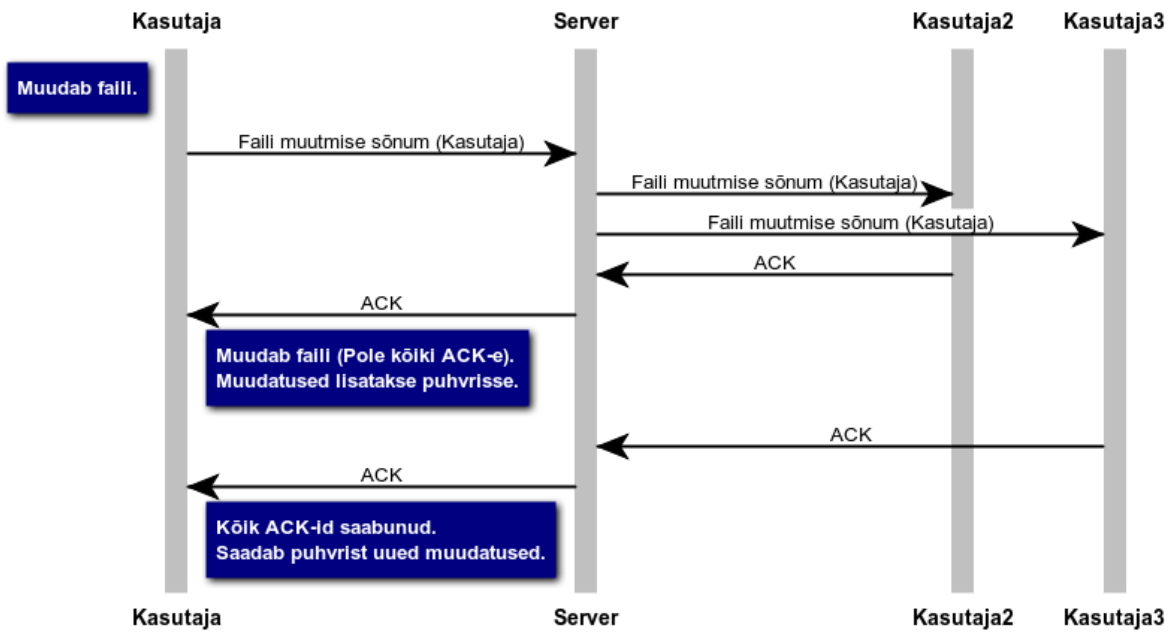


Joonis 9. Teise kasutaja kirjutatud teksti visuaalne tähistamine.

4.2.1 Mitme kasutaja poolt korraga muudatuste teostamine

Eraldi probleemid tekivad siis kui mitu kasutajat peaksid korraga samaaegselt kirjutama, sest sellisel juhul saadakse muudatuse koht ja muudatus kätte, aga kuna teine kasutaja kirjutas ise ka vahepeal, siis peab teksti sisestamise positsiooni muutma. Selle lahendamiseks loodi algoritm, mis kasutab redaktori poolset osalist lukustamist.

Faili muutes edastatakse kasutaja muudatus teistele, kellel antud fail avatud oli ning seejärel jäädakse ootama ACKi, et nad on selle sõnumi kätte saanud. Enne kui kõik kasutajad pole ACKi tagastanud uusi muudatusi kasutaja poolt ei edastata. Need kogutakse puhvrisse ning kui kõikidelt on ACKid saabunud saadetakse terve puhvri sisu korraga. Selline lukustamine aitab vältida olukorda kus ühe kasutaja muudatustest tekib N kasutaja korral N võimalikku erinevat olekut, mis võis tekkida kui kasutaja hoidis klaviatuuril suvalise tähe nuppu all. Hetkel aga on iga kasutaja kohta võimalik tekkida ainult kaks olekut: kas uus muudatus on saabunud või ei ole.



Joonis 10. Faili muudatuste sünkroniseerimine.

Sõnumi vastuvõtmisel kasutatakse selle peal algoritmi, mis muudab teise kasutaja poolt tulnud sõnumi kursorite koordinaadid korrektseteks. Selleks hoiab iga kasutaja teiste kasutajate viimast muudatust meeles, et vajadusel saaks virtuaalselt muudatuse tagasi võtta, et teada mis positsioonile peaks teksti sisestama.

Näiteks kui kasutajad A, B ja C kirjutavad kõik korraga ühe tähe, siis vaatleme kuidas käitub kasutaja C sõnumite vastuvõtmisel. Esimesena jõuab temani näiteks kasutaja A muudatus, aga kuna kasutajal A polnud veel C muudatust, korrigeerib kasutaja C sõnumis olevaid kursorite asukohti võttes virtuaalselt oma tehtud muudatuse tagasi ning seejärel sisestab kasutaja A muudatuse ning saadab talle ACKi tagasi. Järgmisena saabub kasutaja B sõnum. Temal ei olnud sõnumit kirjutades ei A muudatust ega C muudatust, seega võetakse mõlemad virtuaalselt tagasi ja saadakse korrektsed koordinaadid sõnumile B ning see järel saadetakse talle tagasi ACK. Kasutajal C on nüüd kõik sisestused olemas, teiste kasutajatega käitutakse täpselt samamoodi.

Lahenduse leidis ka probleem, mis tekkis kui mitu kasutajat kirjutasid täpselt samasse kohta või muutmise kohad kattusid. Üldalgoritmi põhjal tekkis olukord, kus mõlemad võtsid oma muudatuse virtuaalselt tagasi ning sisestasid siis teiselt kasutajalt tulnud teksti ning sellega sisestati alati serverilt tulnud tekst enne enda oma ja seda igal kasutajal ehk kui kõik kasutajad kirjutaksid samasse kohta samal ajal, siis tekiks neil kõigil erinev faili sisu. Selle lahendamiseks kasutati asjaolu, et server annab igale kasutajale liitumisel nime mis koosneb tema grupiga

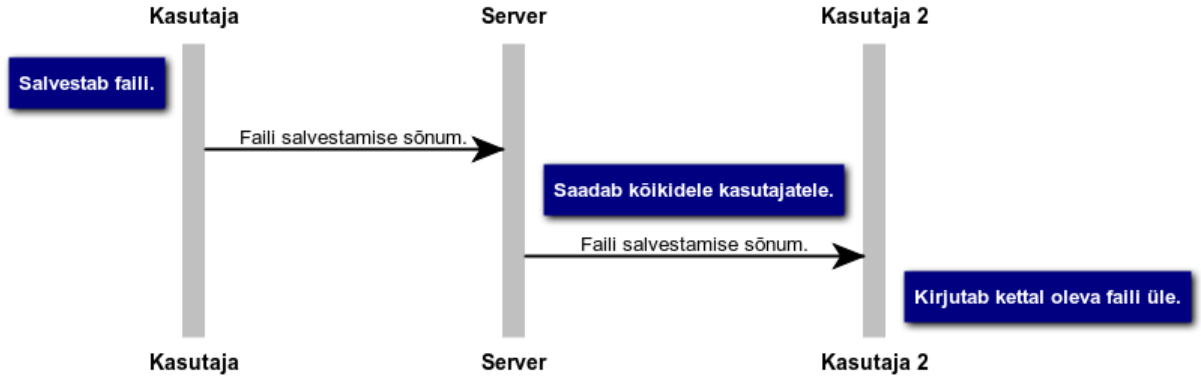
liitumise järjekorranumbrist. Selle numbri järgi moodustatakse järjekord, kus eespool on väiksema numbriga kasutajad ning sellega suudetakse koordinaatide kattumisel kasutajate vahel kõigil tekst samas järjekorras kirja panna.

4.3 Failide salvestamine

Failide salvestamise tsükkel saab alguse sellest kui üks kasutaja salvestab redaktoris. Salvestamise ajal vaadatakse üle tekst, mida salvestatakse ning koostatakse vastav sõnum.

Faili sisu loetakse salvestamisel sisse ja teisendatakse base64 süsteemi, see protsess on vajalik selle jaoks, et kuna faili salvestamisel tuleb see kõigile saata, aga kuna kõigil ei pruugi olla failid avatud, siis ei saa neil kasutada redaktori sisest salvestamist.

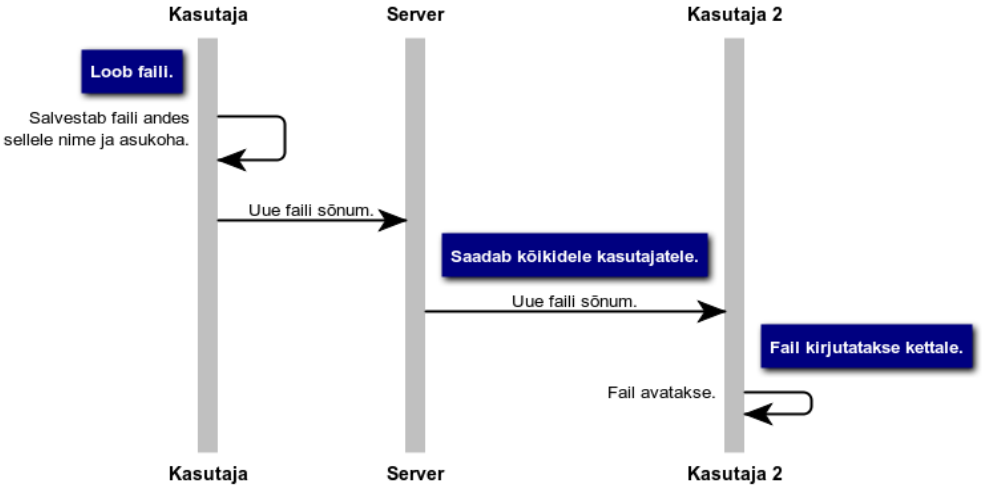
Salvestamisel kirjutatakse kasutajal üle kettal oleva faili sisu..



Joonis 11. Faili salvestamine.

4.4 Uute failide loomine

Uute failide loomine käib läbi redaktori enda. Faili jagatakse teistega seejärel kui faili esimest korda salvestatakse, sest ilma selleta ei saa olla kindel, et fail sellesse projekti kuulub. Faili jagamine käib samade reeglite alusel nagu salvestaminegi ainsaks erinevuseks on, et uut faili luues avatakse see fail kõikide teiste kasutajate redaktoris. Faili avamine on lihtsalt teada andmiseks teistele kasutajatele selle loomiseks. Fail avatakse kõrval sakis ja see ei mõjuta hetkel kasutajal lahti olevasse faili kirjutamist.



Joonis 12. Uue faili loomine.

5. Rakenduse sulgemine

Rakenduse sulgemise all mõeldakse olukorda, kus ühendus serveriga katkeb. Selliseid olukordi on kolm:

- **Tekstiredaktori sulgemine**. Kasutaja sulgeb tekstiredaktori ning operatsioonisüsteem sulgeb kõik redaktori poolt käima pandud alamtegevused ja programmid.
- **Plugina sulgemine**. Kasutajal on võimalus redaktori kontekstimenüüst plugina töö lõpetada.
- **Internetiühenduse kadumine**. Kasutajal kaob ära interneti ühendus, mille tõttu soklid lõpetavad oma töö.

5.1 Rakenduse sulgemine kasutajana

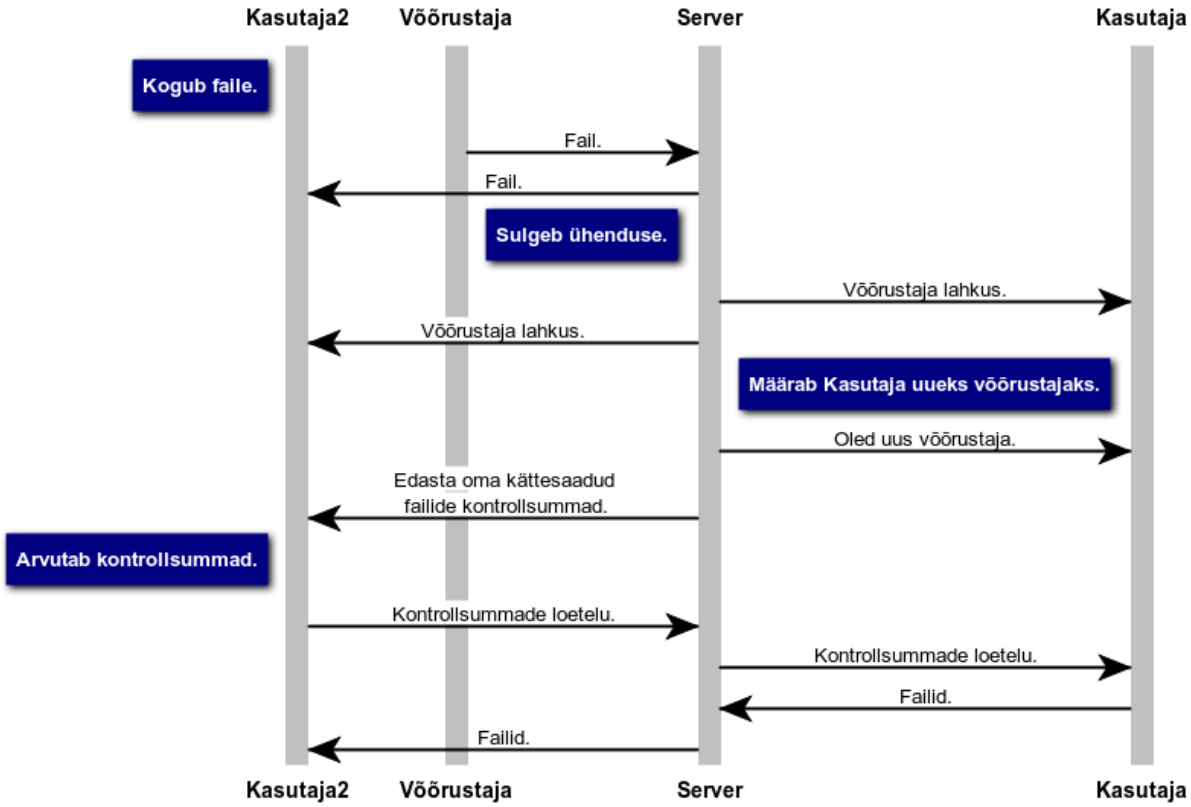
Kasutaja sulgeb oma ühenduse ühel eeltoodud viisil. Server annab sellest kõikidele grupi liikmetele teada. Juhul kui lahkuja puhul oli tegemist uue kasutajaga, kellele saadeti alles faile, siis lõpetatakse ka failide saatmine. Vaatamata sellele, et kõike faile ei jõutud endale võtta ei kustutata siiski olemasolevaid faile, sest juhul kui sulgemise põhjuseks oli ühenduse kadumine, pole vajadust uuesti samu faile saata.

5.2 Rakenduse sulgemine võõrustajana

Võõrustaja rakenduse sulgemine redaktori poole pealt on sarnane kasutaja omaga. Serveril aga tuleb rohkem tööd teha, sest on vaja leida uus võõrustaja. Uus võõrustaja valitakse kasutajate hulgast kellel on kõik projekti failid olemas. Uuele võõrustajale antakse ka teada, et tema on selleks valitud. Juhul kui sellist kasutajat ei leita, siis server lõpetab grupi töö ning kasutajate rakendustes sulgub programm.

Ühenduse katkemisel võib tekkida probleem kui samal ajal on mõni uus liituja, kes alles võtab faile võõrustajalt, sellisel juhul tekib olukord, kus kasutaja ei saa kõike faile projektist. Selle vältimiseks antakse sellele kasutaja rakendusele, sellest teada ning kasutaja vaatab üle, missugused failid ta kätte sai ja arvutab kontrollsummad neist ja saadab need edasi serverile.

Server edastab need juba uueks võõrustajaks määratud kasutajale, kes käitub päringuga samamoodi nagu tavaliselt failide saatmisel.



Joonis 13. Võõrustaja lahkumine failide edastamise ajal.

6. Kokkuvõte

Töö eesmärgiks oli luua rakendus, mis lubab mitmel kasutajal reaalajas faile muuta ja neid hallata. Rakenduse serveri komponent pidi olema lihtne ning ei tohtinud faile serveris hoiustada. Sünkroniseerimine pidi käima lõppkasutajate kaudu ning serveri ülesandeks oli kasutajaid ühendada omavahel. Rakenduse seadistamine pidi olema piisavalt lihtne ja kiire.

Töö eesmärgid täideti täielikult. Töö tulemusel valmis liides tekstiredaktorile Sublime Text 3, läbi mille on võimalik reaalajas mitmekesi faili muuta ning neid hallata. Kogu sünkroniseerimise protsess toimub lõppkasutajate poolel, kasutades algoritme, mis suudavad sõnumites sisaldavaid aegunud andmeid muuta ajakohasteks. Rakenduse seadistamiseks on loodud JSON formaadis fail kus saab muuta välja väärtuseid vastavalt vajadusele. Loodi ka lahendus erinevate kasutajate muudatuste eristamiseks. Kõikide tehnoloogiate juures selgitati, miks ja milleks neid kasutati.

Rakenduse võimalikke edasiarenduse võimalusi on mitmeid. Üheks variandiks on serveri komponent täielikult ära kaotada ja minna üle P2P lahendusele. Kuna praegu juba on serveri osakaal küllaltki väike, siis väga suuri muutusi kliendi poole pealt tegema ei pea. Küll tuleb, aga mõelda välja loogika, kuidas erinevate kasutajate IP aadresse saada ja vahetada omavahel ning kuidas NATi taha pääseda. Teiseks võimaluseks on kasutada ära redaktori omadust koodi selle siseselt käima lasta ja lisada juurde sinna tehisintellekt. Tehisintellekti töö oleks leida üles teiste kasutajate poolt hetkel kirjutatav pooleli olev kood ja seda ignoreerida. Seda selleks, et oleks võimalik koodi tööle panna ilma vigadeta, mis selgelt esinevad kui teatud kasutajatel on meetod pooleli või vigaselt kirjutatud kood.

SUMMARY

The aim of this thesis was to develop an application, what can be used for real-time file management. Server component of the application had to be simple and it was not allowed to store files. Synchronization had to be managed through end user and server task was to connect users. Application setup was supposed to be simple and fast.

The aims of this theses were completed. As a result of this work a plugin was developed for Sublime Text 3 with what was possible to manage files and change them in real-time with multiple users. Whole synchronization procces was managed by user part of the application, where algorithms were made which could update outdated data. Application setup is made through JSON file, where are parameters which user can modify to their needs. Also an GUI solution was made to mark remote users changes to the files.

There are many possible future developments. First version would be to lose the server component and change it for P2P technology. Because already currently server plays a little part in the application, the change should not be very big for the client. Improvements must be made to find users IPs and to share them between each other. Also solution must be made for dynamic IPs and how to get behind NAT. Second version would use editor capability to execute code without leaving from it and to develop an artificial intelligence. Artificial intelligence purpous would be to find part of the codes which are originated from other users and causes an exceptions and to disable these, so that user can compile or interpreter correctly.

Kasutatud kirjandus

[1] „e-Teatmik:IT ja sidetehnika seletav sõnaraamat,“ [Võrgumaterjal]. Available: http://vallaste.ee/. [Kasutatud 17 5 2014].

[2] „Sublime Blog Sublime Text 3 Beta,“ [Võrgumaterjal]. Available: http://www.sublimetext.com/blog/articles/sublime-text-3-beta. [Kasutatud 17 5 2014].

[3] „Build Systems (Batch Processing) Sublime Text Unofficial Documentation,“ [Võrgumaterjal]. Available: http://docs.sublimetext.info/en/latest/file_processing/build_systems.html. [Kasutatud 17 5 2014].

[4] „Floobits:Plans,“ [Võrgumaterjal]. Available: https://floobits.com/plans. [Kasutatud 17 5 2014].

[5] „API Reference - Sublime Text 3 Documentation,“ [Võrgumaterjal]. Available: http://www.sublimetext.com/docs/3/api_reference.html. [Kasutatud 17 5 2014].

[6] „TIEBO Software: Tiebo index,“ [Võrgumaterjal]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Kasutatud 17 5 2014].

[7] „TCP (Transmission Control Protocol) (Linktionary Term),“ [Võrgumaterjal]. Available: http://www.linktionary.com/t/tcp.html. [Kasutatud 17 5 2014].

[8] „JSON,“ [Võrgumaterjal]. Available: http://json.org/. [Kasutatud 17 5 2014].

[9] „RFC 4648 - The Base16, Base32 and Base64 Data Encodings,“ [Võrgumaterjal]. Available: https://tools.ietf.org/html/rfc4648. [Kasutatud 17 5 2014].

[10] N. I. o. S. a. Technology, „Secure Hash Standard (SHS),“ National Institute of Standards and Technology, Gaithersburg, 2012.

[11] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

Lisa 1

Tabel 2. Bas64 kooditabel kodeeringu määramiseks.

Indeks	Väärtus	Indeks	Väärtus	Indeks	Väärtus	Indeks	Väärtus	Indeks	Väärtus
0	A	13	N	26	a	39	n	52	0
1	B	14	O	27	b	40	o	53	1
2	C	15	P	28	c	41	p	54	2
3	D	19	Q	29	d	42	q	55	3
4	E	17	R	30	e	43	r	56	4
5	F	18	S	31	f	44	s	57	5
6	G	19	T	32	g	45	t	58	6
7	H	20	U	33	h	46	u	59	7
8	I	21	V	34	i	47	v	60	8
9	J	22	W	35	j	48	w	61	9
10	K	23	X	36	k	49	x	62	+
11	L	24	Y	37	l	50	y	63	/
12	M	25	Z	38	m	51	z		