OULUN AMMATTIKORKEAKOULU

Joonas Partanen

DATAN LEIMAAMINEN JA KÄSITTELY TUKIASEMAN TESTIAUTOMAATIOJÄRJESTELMÄSSÄ

DATAN LEIMAAMINEN JA KÄSITTELY TUKIASEMAN TESTIAUTOMAATIOJÄRJESTELMÄSSÄ

Joonas Partanen
Opinnäytetyö
Kevät 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Joonas Partanen
Opinnäytetyön nimi: Datan leimaaminen ja käsittely tukiaseman testiautomaatiojärjestelmässä
Työn ohjaajat: Jarkko Halttu (Nokia), Sami Laine (Oulun ammattikorkeakoulu)
Työn valmistumislukukausi ja -vuosi: Kevät 2023
Sivumäärä: 20

Tässä opinnäytetyössä käytiin läpi testiautomaatiojärjestelmän kokoonpano korkealla tasolla sekä esitettiin yksi ratkaisu datan elinkaaren hallintaan. Työn lähtökohta oli, että käyttäjä pystyisi määrittämään itse haluamansa datan säilytysajan leimaamalla tiedostot. Lisäksi opinnäytetyössä toteutettiin proof of concept -ohjelma, joka lukee tuota leimaa ja tekee datalle tarvittavat siirto- tai poistotoimenpiteet sen pohjalta.

Automatisoidussa tukiasemien testaamisessa syntyy valtavia määriä dataa erilaisien lokitiedostojen muodossa. Datan elinkaaren hallinta on täten tärkeä osa tehokasta työskentelytapaa.

Testiautomaatio mahdollistaa järjestelmien jatkuvan testaamisen, joten myös lokitiedostoja syntyy jatkuvasti lisää. Tietoa ei ole kuitenkaan järkevää tai kustannustehokasta säilöä loputtomiin, joten tarvittiin ratkaisu tärkeimpien tietojen merkitsemiseen ja siirtämiseen pidempiaikaiseen säilöön.

Asiasanat: automaatio, tukiasema, data

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author: Joonas Partanen
Title of thesis: Data classification and handling in automated base station testing
Supervisors: Jarkko Halttu (Nokia), Sami Laine (Oulun ammattikorkeakoulu)
Term and year when the thesis was submitted: Spring 2023
Number of pages: 20

In this thesis, we explored the structure of a test automation pipeline and proposed one solution for data lifecycle management. The basic idea was that the user should be able to determine how long the data should be preserved by labelling the files. We also created a proof-of-concept program which operates based on this labelling by either removing the data or by transferring it to longer-term storage.

Automated base station testing generates considerable amount of data in the form of various log files. Data lifecycle management is important part of smart and efficient way of working.

With the help of automation, testing can be continuous and so there is a constant stream of new log files and data. It is not sensible or cost effective to try to save all of this indefinitely, so there was a need for labelling all the data and a way of handling it appropriately.

Keywords: automation, base station, data

SISÄLLYS

SANASTO

5G	Viidennen sukupolven langaton tiedonsiirtotekniikka.
eNB	LTE-tukiasema.
GlusterFS	Avoimen lähdekoodin skaalautuva tiedostojärjestelmä.
gNG	5G-tukiasema.
KPI	Key Performance Indicator, tukiasemalta mitattava suorituskyvystä tai toiminnasta kertova arvo.
LTE	Long Term Evolution, neljännen sukupolven (4G) langaton tiedonsiirtotekniikka.
Raakadata	Käsittelemätön tekstimuotoinen lokitieto aikaleimoineen, joka tulee sellaisenaan esim. tukiasemalta.
Tukiasema	Kiinteästi asennettu kokonaisuus, jonka kautta verkon käyttäjät muodostavat data- tai puheyhteyden.
Testi	Tukiaseman suoritusta eri käyttötapauksissa mittaava testi.
Testiajo	Yksi automaation suorittama kokonaisuus, joka sisältää yhden tai useamman testin.
UI	User Interface, loppukäyttäjän näkymä.

1 JOHDANTO

Tämä opinnäytetyö tehtiin Nokia Solutions and Networks Oy toimeksiantona. Työn aiheena oli selvittää testiautomaatiota kehittävälle tiimille, miten suuren datamäärän leimaaminen, käsittely ja poistaminen tulisi hoitaa eri järjestelmissä. Testiautomaatiotiimillä ei ole tällä hetkellä käytössä valmista mekanismia, joka seuraisi, milloin data vanhentuu ja se tulisi poistaa. Joissain järjestelmissä on käytössä vanhimpien tiedostojen poisto automaattisesti tietyn ajan kuluessa, mutta ongelmaksi muodostuu se, jos joitain referenssinä käytettäviä tärkeitä tuloksia tai tiedostoja poistuu tuossa prosessissa. Tässä työssä tutkittava ratkaisu edesauttaa myös pilvipohjaisen tallennusratkaisun käyttöönottoa, joka ratkaisisi tarpeellisen tiedon saatavuuden myöhemminkin.

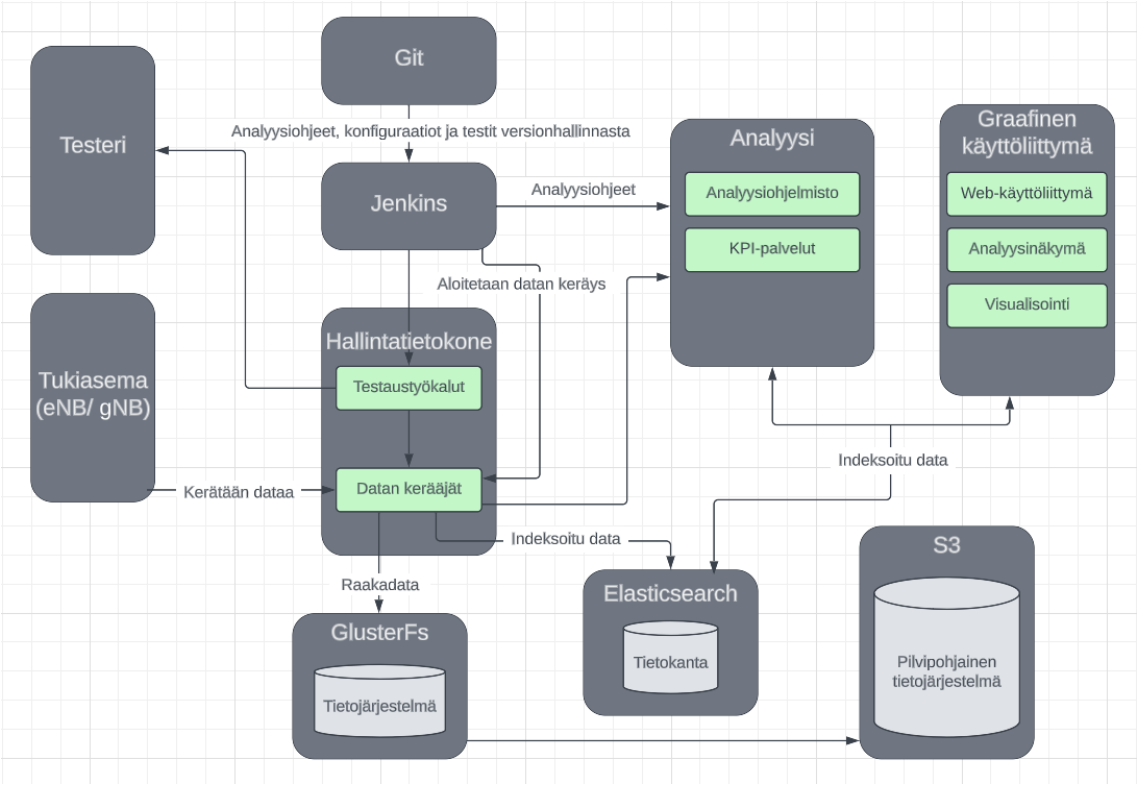
Työssä käsiteltiin kahdenlaista dataa: analysoimatonta, suoraa testattavasta järjestelmästä tulevaa raakadataa sekä analysoitua ja käsiteltyä dataa. Lähtökohtana oli, että kaikkeen dataan voisi testiajokohtaisesti merkitä ennalta määrättyjen sääntöjen pohjalta aikaleiman, joka kertoisi, milloin tiedot saa poistaa järjestelmistä. Kun datan leimaaminen saatiin valmiiksi, tutkittiin, millä metodeilla dataa voitaisiin käydä läpi ja poistaa tai siirtää pidempiaikaiseen tallennukseen järjestelmällisesti. Tätä varten varattiin oma virtuaalinen palvelin, joka suorittaa datan läpikäymistä, poistoa ja tarvittaessa siirtoa pilvitallennustilaan. Prosessista tehtiin proof of concept -tyyppinen ratkaisu, jonka voi tulevaisuudessa jalostaa lopulliseksi käyttöönotettavaksi palveluksi. Palvelun tai ohjelman toteuttamiseen oli monta mahdollisuutta, mutta tässä työssä käytettiin ohjelmointikielenä Pythonia sen monipuolisuuden ja yksinkertaisen syntaksin takia.

Työssä käsitelty data ei ole yrityksen toiminnan kannalta kriittistä (business critical) dataa, mutta datan elinkaaren hallinta on kuitenkin oleellinen osa myös testidatan kannalta. Datan elinkaari määräytyy sen mukaan, kuinka usein sitä tarvitaan. Yleensä data on hyödyllisintä heti sen luomisen jälkeen ja sitä tarvitaan tällöin usein. Tässä vaiheessa datan tulisi olla aina varmasti, helposti ja nopeasti saatavilla. Datan vanhetessa, esimerkiksi meidän tapauksessamme noin 2 viikon jälkeen sen luomisesta, siihen ei enää tarvitse olla välitöntä, nopeaa pääsyä, ja se voidaan siirtää hitaampaan tiedostojärjestelmään tai jopa tuhota. Lopuksi, kun datasta ei ole enää hyötyä tai se ei ole validia, se voidaan tuhota lopullisesti. (1.)

Kirjallisen osion aluksi käydään läpi, mistä yksi testiautomaation ohjaama tukiasema voi koostua ja mitä kaikkea käsite testilinja pitää sisällään. Perusteiden jälkeen perehdytään datan kulkuun järjestelmästä toiseen, lähtien tukiasematasolta ja päätyen aina loppusijoituspaikkaan. Jotta dataa voidaan liikutella ja muokata, on tärkeä tietää, millaisia tiedostoja tukiasemalta ja eri testereiltä tulee, kuinka paljon niitä on ja millaisessa muodossa ne on tallennettu. Ennen varsinaista selvitystä tulee olla edes korkean tason ymmärrys järjestelmän eri komponenteista. Lopuksi tutustutaan itse toteutuksen toimintaan ja mahdollisiin jatkokehitystoimiin.

2 TESTIAUTOMAATIOJÄRJESTELMÄ

Työn onnistumisen kannalta on tärkeä ymmärtää, mistä tarkasteltava testiautomaatiojärjestelmä koostuu. Kuvassa 1 on kuvattu järjestelmän kokonaisuus ja toiminta. Testiautomaation kokoonpano muodostuu yksinkertaistetusti muutamasta komponentista, joista oleellisin on testattava 5G- tai LTE-tukiasema, riippuen testattavasta teknologiasta. Näiden hallintaan on varattu tietokone, jolla suoritetaan tarvittavat testit ja kerätään testiajosta saatavat tulokset ja muu tallennettava data. Toinen tärkeä osa testilinjaa on testeri, jolla simuloidaan erilaisia käyttäjiä ja käyttäjämääriä sekä erilaista matkapuhelinverkossa tapahtuvaa liikennettä. Testerin toiminta ei ole tämän työn kannalta oleellista, kunhan tiedetään korkealla tasolla, mitä se tekee. Automaation kokonaisuuteen kuuluu lisäksi kolme pääasiallista tiedostojen tallennus- ja säilytyspaikkaa, joiden toiminta ja tarkoitus eroaa hieman toisistaan, sekä analyysijärjestelmä ja loppukäyttäjälle näkyvä graafinen web-käyttöliittymä visualisointeineen ja valmiiksi analysoituine tuloksineen. Seuraavaksi käydään läpi järjestelmän eri komponenttien toimintaa ja tarkoitusta hieman tarkemmin.



KUVA 1. Testiautomaatiojärjestelmä yksinkertaistettuna

2.1 Git-versionhallintajärjestelmä

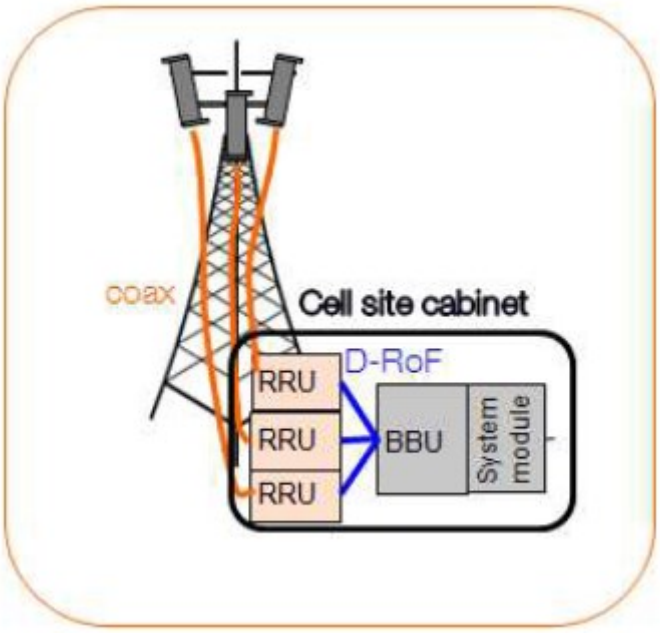
Git on avoimen lähdekoodin helppokäyttöinen versionhallintajärjestelmä (2). Testiautomaation tapauksessa siellä sijaitsee kaikki tarpeelliset tiedostot, kuten testiskriptit ja analyysisäännöt. Myös tukiaseman ja testerien konfigurointitiedostot on usein sijoitettu versionhallintaan. Tämä mahdollistaa sen, että jokainen testiajo suoritetaan teoriassa aina samoilla säännöillä ja testiskripteillä, ja näin ollen tulokset ovat vertailukelpoisia keskenään. Käyttäjän on myös helppo tehdä pieniä muutoksia ilman huolta järjestelmän sekoittamisesta tai rikkomisesta.

2.2 Jenkins-automaatiotyökalu

Avoimen lähdekoodin Jenkins on automaatiotyökalu, jota voidaan hyödyntää testiautomaatiossa monin tavoin (3). Nokian automatisoidussa testauksessa se hoitaa mm. tukiaseman ja testerin ohjelmistojen päivitykset, testiskriptien ja analyysisääntöjen muutoksien noutamisen versionhallinnasta sekä lokien keräämisen ohjaamisen, pakkaamisen sekä oikeaan paikkaan siirtämisen.

2.3 Tukiasema

Testattavien tukiasemien kokoonpano voi vaihdella paljonkin, mutta yleisesti niihin kaikkiin kuuluu kuitenkin seuraavat komponentit (kuva 2): systeemimoduuli, kantataajuusyksikkö tai -yksiköt (BBU, engl. baseband unit), oikeat tai simuloidut radiot (RRU, engl. Remote Radio Unit) sekä joissain tapauksissa myös oikeat antennit (4). Tukiaseman hallintatietokone on myös merkittävässä roolissa testiautomaation kannalta, koska sen avulla kerätään tukiasematason lokeja ja suoritetaan testeriä ohjaavia skenaarioita.



KUVA 2. Tukiaseman yksinkertaistettu kokoonpano (5)

2.4 Hallintatietokone

Hallintatietokoneella on tukiaseman hallinnan lisäksi tärkeä rooli datan keräämisessä. Se saa keräysohjeet ja käskyn aloittaa ja lopettaa lokien tallentamisen Jenkins-palvelulta. Hallintatietokone siis toimii systeemitason lokien ensimmäisenä, joskin hyvin hetkellisenä, tallennuspaikkana. Hallintakoneella lokit pakataan ja siirretään sekä pakattuina edelleen analysoitavaksi että raakadatana GlusterFS-tiedostojärjestelmään.

2.5 Tiedostojen tallennusjärjestelmät

Järjestelmään kuuluu kolme pääasiallista tiedostojen ja datan tallennus- ja säilytyspaikkaa. Näille kaikille yhteisiä ominaisuuksia ovat mm. skaalautuvuus ja lähtökohtaisesti suuri kapasiteetti. Kaksi näistä, GlusterFS ja Elasticsearch, on järjestelmän toiminnan kannalta tärkeitä, koska niitä käytetään nopeaan, heti saatavilla olevan datan säilyttämiseen ja analysoimiseen. Nämä ovat paikallisia, omassa verkossa olevia fyysisiä järjestelmiä, joiden tiedonsiirtonopeudet ovat huippuluokkaa verrattuna verkon yli tapahtuvaan tiedonsiirtoon. Kolmanteen, S3:een, viedään ns. tärkeää dataa, jota pitää säilyttää pidempiä aikoja. Se on pilvipalvelu, joka toimii verkossa, joten sen tiedonsiirtonopeudet ovat hieman rajallisemmat verrattuna paikallisiin järjestelmiin. Vaikka tämä tallennusjärjestelmä ei ole vielä käytössä, ratkaisee opinnäytetyö myös siihen liittyviä ongelmia.

KUVA 3. Havainnollistava kuva GlusterFS-tallennusjärjestelmän kansiorakenteesta

2.5.1 GlusterFS

GlusterFS on avoimen lähdekoodin tiedostojen tallennusjärjestelmä (6). Sen tiedostorakenne on hyvin perinteinen kansiorakenne, kuten kuvassa 3 on havainnollistettu. Tämä mahdollistaa tiedostojen lisäämisen, poistamisen, muuttamisen tai etsimisen suhteellisen helposti. Kun tiedostoja siirretään paikasta toiseen, rakenteen tulisi säilyä samana. Täällä oleva data on varmennettu paikallisesti, mutta datamäärät ovat valtavia eikä tietoa voida säilöä erityisen pitkiä aikoja.

2.5.2 Elasticsearch

Elasticsearch ei virallisesti ole tiedostojen tallennusalusta vaan haku- ja analysointityökalu. Kun tietoa viedään Elasticsearch-järjestelmään, se parsitaan, analysoidaan ja indeksoidaan, jotta käyttäjät voivat ajaa kyselyitä ja yhdistellä dataa saadakseen monimutkaisiakin yhteenvetoja tiedoista. Elasticsearchin dokumentaatio käyttää tästä prosessista nimitystä *Data ingestion.* (7.)

2.5.3 S3

S3 on Amazon Web Servicesin tarjoama pilvipohjainen tiedostojen tallennusratkaisu (8). Sen on tarkoitus toimia pitkään säilytettävien tiedostojen loppusijoituspaikkana, jotta paikallisena nopeampana tallennustilana toimivan GlusterFS:n tila saadaan tehokkaammin käyttöön ajankohtaisille tiedostoille. Tässä työssä toteutettu datan leimaaminen mahdollistaa myös tulevaisuudessa S3:n tilan tehokkaamman käytön. S3 on toteutettu Nokian omalla pilviratkaisulla eli dataa ei tallenneta ulkoisien toimijoiden järjestelmiin.

3 SUUNNITTELU JA TOTEUTUS

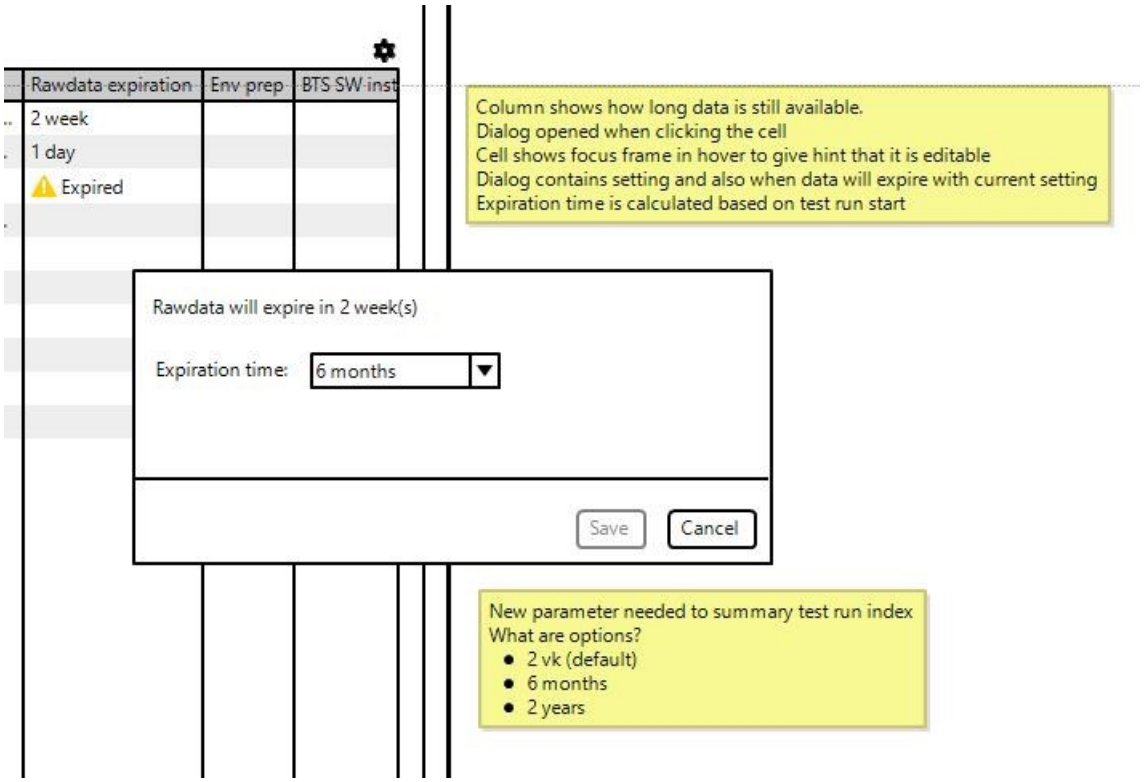
3.1 Lähtötilanne

Nykyisessä testiautomaatiojärjestelmässä ei ole käytössä automatisoituja sääntöjä tai mekanismeja, jotka mahdollistaisivat erilaiset tallennusajat tietyille tai valikoiduille tiedoille. Tällä hetkellä pidempään säilytetään ainoastaan analyysien tuloksia sekä dataa, jota web-käyttöliittymä käyttää esittämään tuloksia. GlusterFS:ssä säilöttävä raakadata on saatavilla tällä hetkellä vain tietyn ajan eikä sieltä viedä S3-palveluun oikeastaan mitään. Ongelmana pitkän ajan säilytyksessä on se, ettei järjestelmään ole määritetty sääntöjä tai tapaa, jolla tietoja vietäisiin S3:een. Tämä työ auttaa ratkaisemaan ongelman ainakin osittain.

3.2 Suunnittelu ja vaatimusten määrittely

Pienen suunnittelun jälkeen päädyttiin yksinkertaiseen mutta tässä tapauksessa varmasti toimivaan ratkaisuun. Tiedostorakenteeseen luotaisiin ajotasolle (kuva 3) tekstitiedosto esimerkiksi nimellä *expiration date* ja se sisältäisi päivän tarkkuudella ajankohdan, jolloin tiedostot saa poistaa. Tekstimuotoinen tiedosto valittiin, koska sitä olisi ehdottomasti helpointa säilyttää ja muokata eri järjestelmissä. Lisäksi jos myöhemmin ilmenee tarve, voidaan tiedosto nimetä uudelleen joksikin geneerisemmäksi ja määrittää sinne uusia parametrejä tai jopa muuttaa tiedoston formaattia esimerkiksi JSON-tyyppiseksi. Ensimmäinen määrittelypalaveri tuotti runsaasti lisäkysymyksiä, kuten mitkä komponentit web-ohjelman puolelta tarvitsevat muutosta, miten tieto poistopäivästä esitetään web-UI:ssa tai miten käyttäjä voi muuttaa vanhenemisen päivämäärää. Tekstitiedostomuotoinen leima vaikutti edelleen parhaalta tavalta sisällyttää poistoajankohta dataan.

Työn toteuttamista varten täytyi ensin kartoittaa, mitkä osat järjestelmästä siihen tarvitaan mukaan. Haluttiin selkeä ja helppo menetelmä luoda aikaleiman sisältävä tiedosto, joka olisi tarvittaessa myös loppukäyttäjän muokattavissa. Järjestelmän kompleksisuuden takia muutoksista kirjattiin määrittelyt ja pyydettiin testiautomaatiotiimin vastaavia henkilöitä toteuttamaan muutokset. Näin saataisiin kerralla kaikki tarvittavat komponentit, niin näkyvät kuin taustalla olevat, yhteensopiviksi. Vaatimusten määrittely tehtiin kahdessa osassa.



KUVA 4. Ensimmäinen suuntaa antava UI-suunnitelma

3.3 Vaatimusten määrittely

Ensimmäisessä määrittelyssä esitettiin vaatimuksia kokonaiskuvan osalta, jotta voitiin todeta, mitkä komponentit tarvitsevat muutoksia. Loppukäyttäjän näkökulmasta muutoksia tulisi vain käyttöliittymään ja datan saatavuuteen, mutta näitä varten myös web-ohjelman palvelinpuolta (backend) ja Elasticin toimintaa olisi muutettava. Pääasiallisia muutoskohteita oli kolme. Ensimmäiseksi web-UI:hin pitäisi lisätä sarake, joka kertoo datan säilymisestä sekä valintaikkuna, jossa käyttäjä voi määrittää säilytysajan (kuva 4). Toiseksi Elasticsearchiin olisi tehtävä muutos, joka osaa käsitellä uuden sarakkeen ja lopuksi tieto datan säilymisajasta tulisi saadaan Elasticista tiedostona GlusterFS:lle.

Kun datan säilytysajan kertova luokittelu on tehty ja tiedosto löytyy sille määrätystä hakemistosta, voitaisiin alkaa rakentamaan mekanismia niiden löytämiseen ja lukemiseen. Jotta toteutus saataisiin pidettyä mahdollisimman mutkattomana, todettiin Python tähän tehtävään parhaiten sopivaksi työkaluksi monipuolisuutensa ja yksinkertaisuutensa vuoksi. Tiedostojen lukemisen ja käsittelyn tulisi onnistua virtuaaliselta koneelta, johon Glusterin tiedostohakemisto on kiinnitetty verkkolevyn tavoin. Ohjelman toimintaperiaate on suoraviivainen (kuva 5). Alustana toimii kehitysvaiheessa

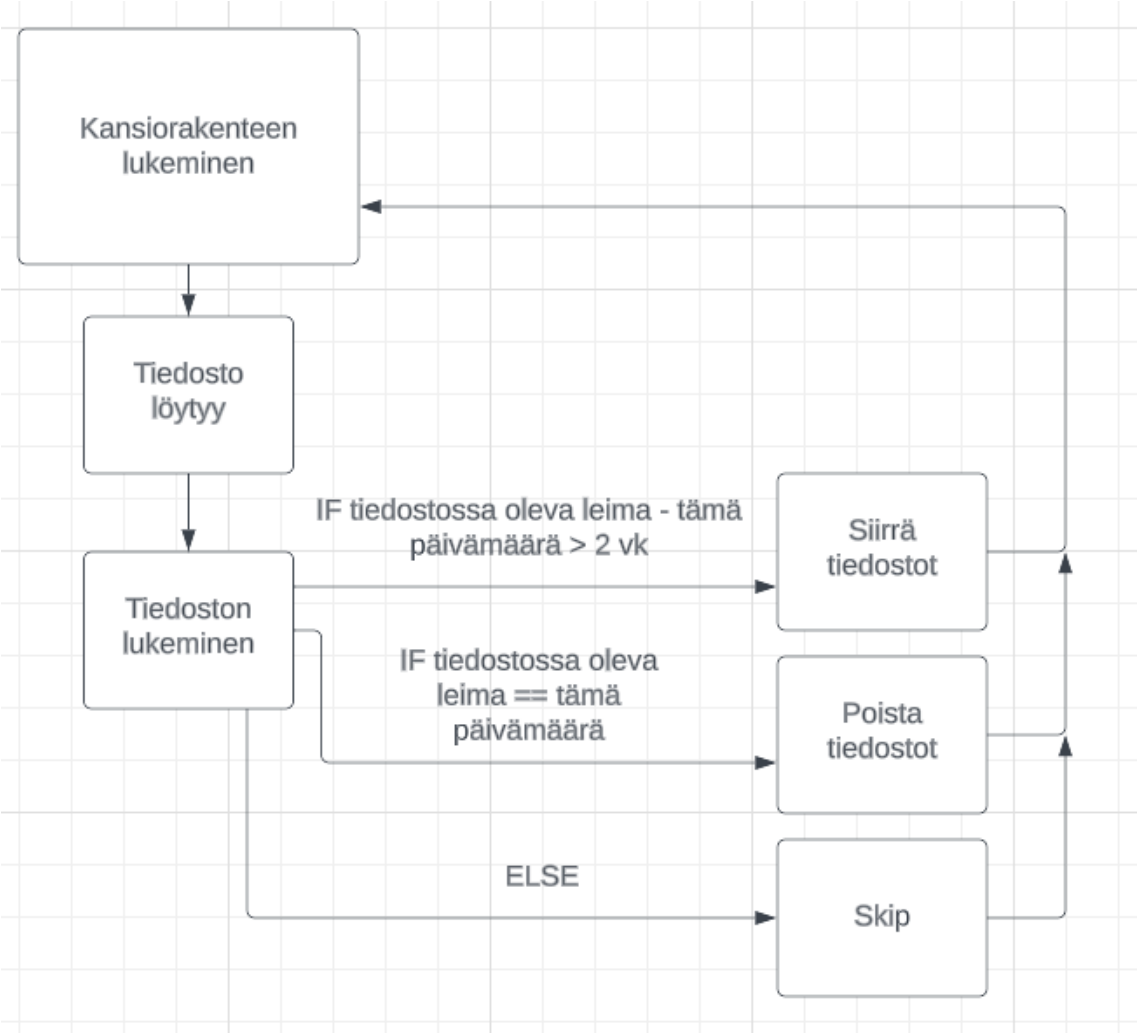
CentOS-jakelulla oleva Linux-virtuaalikone, koska sellainen oli valmiina. *Ripgrep*-komennon asennus oli ainoa muutos, jonka olemassa olevalle ympäristölle tarvitsisi tehdä.

3.4 Python ja sen moduulit

Toteutuksen ohjelmointikieleksi valikoitui siis Python sen monipuolisuuden, helposti ymmärrettävän syntaksin sekä suuren moduuli- ja pakettivalikoiman takia. Käytössä on myös virtuaalinen palvelin, eivätkä maksimaalinen tehokkuus tai resurssien käyttö aseta rajoitteita ohjelmalle. Oleellisimmat moduulit työssä ovat *threading*, *datetime*, *time* sekä *subprocess*. *Threading* moduulin avulla ohjelmasta saadaan monisäikeinen ja osia koodista voidaan suorittaa samanaikaisesti. Aiemmin mainittu *ripgrep* on Linux-komento, jonka avulla voidaan hakea tieto nopeasti ja tehokkaasti. Tämän komennon koodissa käyttämiseen tarvitaan *subprocess*-moduulia. Moduulit *datetime* ja *time* sisältävät nimensä mukaisesti aikaan ja päivämäärään liittyviä toiminnallisuuksia.

3.5 Ohjelman toiminta

Ohjelma on rakenteeltaan ja toiminnaltaan yksinkertainen (kuva 5). Se koostuu yhdestä luokasta, eri funktioista sekä pääfunktiosta. Jokaista toimintoa varten on oma funktio. Tämä helpottaa itse koodausta sekä parantaa koodin ymmärrettävyyttä, mikä on jatkokehitystä ajatellen tärkeää. Ensin etsitään tarvittavat aikaleiman sisältävät tiedostot *ripgrep*-komennon avulla. *Ripgrep*-komento palauttaa tiedoston sijainnin eli polun, joka lisätään listaan myöhempää käyttö varten. Kun kaikki tiedostot on löydetty, etsintäfunktio palauttaa valmiin listan, jossa on kaikkien aikaleiman sisältävien tiedostojen polut. Valmis lista annetaan parametrina tiedostojen lukufunktioon, joka tarkistaa, ettei tiedosto ole tyhjä, lukee tiedostojen sisältämän aikaleiman ja aloittaa päivämäärien tarkistuksen.



KUVA 5. Korkean tason kuvaus ohjelman toiminnasta

Tarkistusfunktio toimii seuraavilla säännöillä: Jos tiedostossa oleva aikaleima on yli 14 vuorokauden päässä, tiedosto siirretään pidempiaikaiseen tallennuspaikkaan. Jos aikaleima puolestaan on 14 vuorokauden sisällä nykyisestä päivämäärästä, tiedostoille ei tehdä mitään. Jos taas aikaleima on sama kuin sen hetkinen päivä tai aikaleima on menneisyydessä, tiedostot poistetaan. Vertailufunktiossa on myös tarkistus aikaleiman esitysmuotoa varten sekä vielä huomautus sille, jos aikaleiman päivämäärä on menneisyydessä. Ohjelma käynnistää tiedostojen hallintasäikeen, joka huolehtii samanaikaisesti tiedostoille tehtävistä toimista pääsäikeen lukiessa tiedostojen listaa eteenpäin. Näin saatiin aikaiseksi yksinkertainen, mutta tehokas, monisäikeinen ohjelma.

3.6 Huomioita kehitysvaiheesta

Vaikka varsinaisia järjestelmän tai palvelimen asettamia rajoitteita suorituskyvylle ei ollut, täytyi kehitystyössä kuitenkin ottaa huomioon tallennusjärjestelmän usean kymmenen teratavun koko.

Tiedostojen läpikäymistä testailtiin muutamalla metodilla. Ensimmäinen toteutus tehtiin erilliseltä virtuaalikoneelta, johon ei ollut kiinnitetty GlusterFS:ää verkkolevyn tavoin. Tässä versiossa tiedostorakenne käytiin läpi SFTP-yhteyden kautta, mutta se osoittautui melkein heti heikoksi vaihtoehdoksi. Huomioon otettava asia on myös alustan kellon ajan ja päivämäärän tarkistus, jotta aikaleimat ovat vertailukelpoisia keskenään.

4 YHTEENVETO

Tämä opinnäytetyön tarkoitus oli selvittää Nokian testiautomaatiotiimille menetelmiä datan leimaamiseen ja leimatun datan käsittelyyn. Järjestelmän monimutkaisuus yllätti aluksi, mutta työn kannalta oleellisimmat osat aukesivat melko nopeasti kattavan dokumentaation ja tiimin avulla. Tiimi lähti hyvin nopeasti työn aiheen esittelyn jälkeen ideoimaan, mikä kertoi siitä, että tällaiselle työlle oli aidosti tarvetta. Loppujen lopuksi työstä kehittyi jotain oikeasti hyödyllistä. Työn tuloksena syntynyt proof of concept -ohjelma antaa hyvän pohjan jatkokehitykselle, koska siitä selviää sekä prosessin toimintaperiaate ja yksi tapa toteuttaa haluttavat asiat.

Työn edetessä tuli jatkuvasti ilmi uusia ideoita, miten asioita voisi toteuttaa paremmin tai tehokkaammin. Lopullisessa tuotantoon tulevassa versiossa täytyy miettiä, onko järkevää käydä päivittäin läpi samaa listaa vai voisiko osan jo luetuista tiedoista tallentaa esimerkiksi tietokantaan ja toimia sen perusteella. Tässä on kuitenkin riskinä se, että käyttäjä käy muuttamassa säilytysaikaa ja se jää harvemmalla suoritussyklillä huomaamatta. Kun ratkaisu saadaan tuotantokelpoiseksi, myös web-ohjelman toimintaan täytyy tehdä muutoksia. Kun tiedostoja siirretään eri tallennustilojen välillä, esimerkiksi linkit ja polut täytyy myös päivittää, jotta loppukäyttäjät myös löytävät säilötyt tiedot.

Vaikka toteutus jäikin proof of concept -tasolle, työssä esitetiin keinoja ja ratkaisuja testidatan elinkaaren hallintaan. Yleensä data on hyödyllistä ja relevanttia kun se on luotu, mutta joissain tapauksissa käyttäjällä voi olla tarve säilyttää lokeja ja tuloksia pidempiä aikoja esimerkiksi pitkän ajan vertailua varten.

LÄHTEET

1. Petrocelli, Tom 2005. Data Protection and Information Lifecycle Management. Julkaisupaikka: Pearson. Hakupäivä 01.04.2023. https://www.oreilly.com/library/view/data-protection-and/0131927574/
2. Git. Hakupäivä 25.05.2023. https://git-scm.com/
3. Jenkins User Documentation. Hakupäivä 25.05.2023. https://www.jenkins.io/doc/
4. Remy, Jean-Gabriel & Letamendia, Charlotte 2014. LTE Standards. Wiley-ISTE
5. Chanclou, Philippe et al. 2013. Optical fiber solution for mobile fronthaul to achieve Cloud Radio Access Network. Hakupäivä 27.03.2023. https://www.researchgate.net/publication/260334428_Optical_fiber_solution_for_mobile_fronthaul_to_achieve_Cloud_Radio_Access_Network
6. GlusterFS Documentation. Hakupäivä 27.05.2023. https://docs.gluster.org/en/latest/
7. What is Elasticsearch? Hakupäivä 27.03.2023. https://www.elastic.co/what-is/elasticsearch
8. Amazon S3. Hakupäivä 5.6.2023. https://aws.amazon.com/s3/