Joona Oikarinen

Analysaattorin diagnostiikan kehittäminen käyttäjäystävällisemmäksi

Insinööri (AMK)
Tieto- ja viestintätekniikka
Kevät 2020

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Joona Oikarinen

Työn nimi: Analysaattorin diagnostiikan kehittäminen käyttäjäystävällisemmäksi

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: ohjelmistokehitys, tiedonhallinta, lokitiedostot, tietokanta, käyttöliittymä, Linux, rajapinta, WebSocket

Opinnäytetyö tehtiin toimeksiantona paikalliselle yritykselle. Työn tavoitteena oli kehittää käyttäjäystävällinen kokonaisratkaisu analysaattorin diagnosoinnin helpottamiseksi. Kokonaisratkaisun tarkoituksena oli jäsentää lokitiedostoista oleelliset tiedot, tallentaa ne ja esittää ne käyttäjän haluamalla tavalla. Työ koostui kahdesta eri ohjelmistosta. Ensimmäinen ohjelmisto oli palvelinosuus, joka huolehtii oleelliset tiedot lokeista ja järjestelmästä tietokantaan. Samalla se tarjoaa tietokannasta rajapinnan web-käyttöliittymälle, jolla tiedot esitetään. Toinen ohjelmisto on web-käyttöliittymä, johon tässä työssä tehtiin laajennus. Laajennuksesta käyttäjä pystyy tekemään tarkkoja hakuja lokitiedostoista erilaisten filttereiden avulla ja säätämään asetuksia.

Työssä kerrotaan ohjelmistojen toteuttamisesta sekä niiden kehitykseen käytetyistä teknologioista. Työssä keskityttiin mahdollisimman monipuolisen, mutta helppokäyttöisen käyttöliittymän toteuttamiseen. Kehityksessä piti ottaa huomioon lokitiedostojen standardoimattomuus, rajapintojen yhteensopivuus ja rajoitetut resurssit ohjelman suoritusaikana.

Palvelinosuuden toteuttamisessa käytettiin C++-ohjelmointikieltä ja Qt-kehitysympäristöä. Web-käyttöliittymän osalta käytettiin perinteisiä web-ohjelmointikieliä. Valitut ohjelmointikielet olivat HTML, CSS, JS ja PHP. Kaikki edellä mainitut teknologiat valittiin, koska ne olivat toimeksiantaja yrityksellä ennestään laajassa käytössä. Kommunikointiin ohjelmistojen välillä käytettiin WebSocket-protokollaa. Tätä kommunikointiprotokollaa käytettiin, koska se mahdollistaa yrityksen muiden ohjelmistojen integroimisen tähän järjestelmään jatkokehitysvaiheessa.

Kehitetty palvelinosuus pystyy keräämään analysaattorin standardoimattomat lokitiedostot ja muuta käyttöjärjestelmädataa halutussa muodossa. Tämä prosessi tapahtuu kustomoitujen skriptien avulla. Palvelinosuudella on rajapinnat tietokantaan sekä web-käyttöliittymälle. Rajapinta tietokantaan mahdollistaa datan tallentamisen ja hakemisen halutussa muodossa SQL-kyselyjen avulla. Rajapinta web-käyttöliittymälle mahdollistaa datan lähettämisen ja vastaanottamisen selaimeen, jonka kautta data kulkee käyttäjälle. Vaikka ohjelmistokokonaisuudessa on monta ohjelmistorajapintaa, ne saatiin toimimaan ongelmitta kokonaisuutena.

Ohjelmistokokonaisuus vastasi toimeksiantajayrityksen määrittämiä vaatimuksia. Ohjelmisto mahdollistaa oikein konfiguroidun analysaattorin kokonaisvaltaisen tarkastelun yhdeltä sivulta. Lisäksi mahdollisissa vikatilanteissa ongelmaa voidaan etsiä uusilla etsintätyökaluilla. Kokonaisuus rakennettiin niin, että sitä voidaan laajasti jatkokehittää.

KAMK • University of Applied Sciences

Abstract

Author: Joona Oikarinen

Title of the Publication: Development of user-friendly analyzer's diagnostics

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: software development, data handling, log files, database, user interface, Linux, interface, WebSocket

Thesis was done as a commission for a local company. Goal of the thesis was to develop an overall solution for user-friendly analyzer's diagnostics. Solution included parsing important log data, saving it into the database and visualizing it to user via web user interface. Overall solution constructed from two different software. Server software handled interfaces to scripts, database and web user interface. The second part of the overall solution was an expansion for the company's already existing web user interface. The expansion offers the user an ability to search log files and adjust the software settings.

The thesis tells about the software development and the technologies used in the process. The focus of the thesis was to create an easy-to-use, but still a comprehensive interface. There were some limiting factors to take into account. Basically, there was no agreed standard for the log files. This created a need for a dynamic parsing system. Other factor were interface compatibility and limited resources on target PC's execution time.

Used programming language for developing the server software was C++ and used integrated development environment was Qt. Web user interface used traditional web development languages, such as HTML, CSS, JS and PHP. The abovesaid technologies were already in wide use of the commissioner. That is the reason these technologies were chosen. Software used WebSocket protocol as communication interface. WebSocket communication protocol was chosen because it enables further development. The commissioner has other software using this protocol, so this allows later software integration.

The developed server software is able to parse un-standardized log files and other system data. This process is implemented by using customized script. The software has interfaces to the database and to the web user interface. The database interface enables saving and retrieving data using SQL-queries. The web user interface enables sending and receiving data to a browser. All user control comes from the browser and it's the endpoint of the software's application programming interface. Even the overall solution includes many different interfaces, in the end they work seamlessly.

The overall solution met the requirements of the commissioner. When configuration is set correctly, the software allows a complete view of the system on one page. On fault cases software allows user to search precisely for the problem from the logs. Precision comes from multiple search filters. The solution was built to allow easy and broad further development.

Sisällys

Lyhenneluettelo

CSS	Cascading Styling Sheet. WWW-sivuille kehitetty tyyliohje.
DBMS	Database Management System. Relaatiotietokannan hallintatyökalu.
HTML	Hypertext Markup Language. WWW-sivujen käyttämä standardoitu merkintäkieli.
HTTP	Hypertext Transfer Protocol. WWW-palvelimien ja selainten välissä käytettävä tiedonsiirtoprotokolla.
IBM	International Business Machines Corporation. Yhdysvaltalainen teknologia yritys.
IETF	The Internet Engineering Task Force. Organisaatio, joka vastaa internet-protokollien standardoinnista.
IP	Internet Procotol. Protokolla, joka huolehtii tietoliikennepakettien välittämisestä Internetissä.
JS	JavaScript. Skriptikieli.
JSON	JavaScript Object Notation. Tiedostomuoto tiedonvälitykseen.
SQL	Structured Query Language. Relaatiotietokannan kyselykieli.
TCP	Transmission Control Protocol. Tietoliikenneprotokolla, joka toimii luotettavana yhteytenä tietoverkoissa.

1 Johdanto

Työni aihe tuli suoraan yritykseltä, jolla on tarve kehittää tuotantokäytössä oleva analysaattorin diagnosointijärjestelmä paremmaksi. Yrityksellä on tällä hetkellä käytössä automaattisesti ylläpidettävä tapahtumarekisteri, johon kaivataan parannusta. Nykytilanteessa mahdollisia vikatilanteita tarkastellaan manuaalisesti erilaisten lokitiedostojen perusteella, mikä on aikaa vievää ja tarvitsee aina asiantuntijan.

Työn tarkoituksena on tuottaa ohjelmistojärjestelmä, joka ottaa syötteenä lokitiedostoja, jäsentää niistä oleellisen tiedon, tallentaa oleelliset tiedot tietokantaan sekä näyttää käyttäjän haluamat tiedot web-käyttöliittymällä. Tämä johtaa tilanteeseen, jossa ongelman selvittämisestä tulee nopeampaa, eikä halutun tiedon tarkastelu vaadi samanlaista asiantuntijuutta järjestelmää kohtaan. Käyttöliittymä tulee ainakin aluksi yrityksen sisäiseen käyttöön. Ohjelmiston mahdollinen käyttäjäkunta päätetään myöhemmin pidettävässä palaverissa.

Käyttöliittymän avulla käyttäjä pystyy löytämään mahdollisen ongelman ja ongelman kannalta oleellisen tiedon nopeammin. Työ mahdollistaa lokitiedostojen tarkastelun esimerkiksi tehtaan työntekijän toimesta ja tietyt ongelmatilanteet voidaan löytää jo asiakkaan toimesta. Jos etäyhteys silti tarvitaan, voidaan käyttöliittymän kautta hakea tarvittavia tietoja nopeammin kuin manuaalisella komentorivillä tapahtuvalla tarkastelulla.

Työn toteuttamisessa vastaan tulevia haasteita ovat lokitiedostojen standardisoimattomuus eli oleellisen tiedon jäsentäminen muun tiedon seasta voi olla ongelmallista. Haasteena on myös ohjelmiston eri osien rajapintojen suunnittelu yhteensopiviksi. Lisäksi ohjelmisto saa viedä prosessointiaikaa vain, kun sitä on vapaana, eli muiden ohjelmistojen pitää pystyä saamaan prioriteetti tämän ohjelmiston yli.

Työssä etsitään ratkaisua erilaisten lokitietojen ja systeemitietojen keräämiseen, jäsentämiseen ja visualisoimiseen. Käyttöliittymästä on tarkoitus tehdä yksinkertainen, jotta sen käyttö- ja käyttöönottokynnys on mahdollisimman matala.

Työ liittyy suoraan yrityksen aikaisempiin projekteihin ja onkin eräänlainen laajennus olemassa olevan ohjelmiston rinnalle. Lokitiedostot tulostetaan satunnaisessa formaatissa jo olemassa olevasta ohjelmistosta ja web-käyttöliittymä tulee laajennuksena jo valmiiksi olemassa olevan web-käyttöliittymään.

2 Lokitiedostoista yleisesti

Tässä luvussa esitellään, mitä lokitiedostot ovat, miksi lokitiedostoja käytetään sekä mihin ja miten niitä käytetään. Lokitieto, myöhemmin loki, tarkoittaa tallennetta jostakin tapahtumasta. Tallenteesta käy ilmi aikajärjestys, tapahtuma ja mahdolliset aiheuttajat. Lokituksen ansiosta voidaan tarkkailla ohjelmiston sisäisiä toimintoja jälkeenpäin. Lokituksella tarkoitetaan lokitietojen tallennusta. Tämä mahdollistaa kaiken ohjelmiston käytössä olevan tiedon tallentamisen.

Lokit ovat hyödyllisiä useissa tilanteissa, sillä normaalitilanteessa niiden avulla voidaan tarkastaa ohjelmiston virheetön toimivuus. Virhetilanteissa taas voidaan tutkia, mikä on pielessä ja lähteä korjaamaan virheen aiheuttamaa juurisyytä. Näiden lisäksi lokeista on mahdollista selvittää tekijä, toiminto, ajanhetki, tapahtumia, väärinkäyttö- ja tietomurtoyrityksiä. Oikein suunniteltu ja toteutettu lokiympäristö tallentaa täydellisen tapahtumaketjun ohjelmiston toiminnasta. [1, s. 13.] [2.]

Nykypäivänä melkein kaikki tietotekniikkaan liittyvät laitteet keräävät tapahtumalokia. Tapahtumalokiin tallennetaan tietoa käyttäjän tekemistä toiminnoista kyseisessä järjestelmässä, josta yhtenä esimerkkinä sisään- ja uloskirjautumiset. [2.]

Lokeilla on tietynlainen elinkaari, joka määritetään etukäteen lokitusta suunnitellessa. Elinkaari koostuu lokien keräämisestä, analysoinnista, säilyttämisestä, luovuttamisesta ja poistamisesta tai arkistoinnista. Keräämisellä tarkoitetaan lokien tallentamista. Lokien analysointi voi tapahtua manuaalisesti tai automaattisesti. Datamäärän kasvaessa manuaalinen tarkastelu muuttuu työlääksi ja erillisiä tarkistelutyökaluja tarvitaan tietyn tiedon löytämiseksi. Säilyttämisellä tarkoitetaan lokien tallessapitoa. Luovuttaminen tarkoittaa tiedon antamista kolmannelle osapuolelle. Poistaminen tarkoittaa tiedon hävittämistä. Arkistoinnilla tarkoitetaan lokien pidennettyä säilyttämistä. Usein pitkäaikaissäilytys tapahtuu suoritettavasta ohjelmistosta erillään olevalla tallennusvälineellä. [1, s. 14.] [1, s. 58.]

Lokien avulla voidaan suorittaa ohjelmiston hallinnan ja valvonnan kannalta tärkeitä tehtäviä. Tällaisia tehtäviä ovat poikkeamatilanteen havainnointi, selvitys, toipuminen ja ylläpitoprosessin kehittäminen, jatkuvan kuormituksen seuranta, todisteiden keruu, tapahtumien kiistämättömyyden varmistaminen, käyttäjien, rekisteröityjen ja ylläpitäjien oikeusturvasta huolehtiminen. [1, s. 15–16.]

Kokonaisuus lokien ympärillä ei ole ilmaista ylläpitää, vaan se vaatii laitteisto- ja henkilöstöresursseja. Tämä on otettava huomioon suunnitelmissa ja budjetoinnissa. Lokien käsittelyn pitää perustua ennalta määritettyyn tarpeeseen ja toimintatapaan. Tuloksista aiheutuvat toimenpiteet pitää myös olla ennalta määrätty. Kaikkien käyttäjien oikeusturva ja tietosuoja pitää olla huomioitu kaikessa käsittelyssä. [1, s. 19.]

Lokiympäristössä on määritettävä roolit, vastuut, toimintatavat ja prosessit. Hyvä suunnittelu, dokumentointi ja toteutus poistaa turhia käytännön ongelmia ja haasteita. Edes vahingossa tapahtuvaa väärinkäyttöä ei pysty tapahtumaan, jos ohjelmisto ei mahdollista sitä. Pelisäännöt tulee olla selvillä niin käyttäjillä kuin ylläpitäjilläkin. [1, s. 43.]

2.1 Erilaisia lokitiedostoja

Seuraavaksi käydään läpi erilaisia lokityyppejä. Lokit on mahdollista luokitella monin eri tavoin, mutta esitellään neljä erilaista luokittelua: ylläpitoloki, käyttöloki, muutosloki ja virheloki. Lokien määrittely tiettyihin luokkiin on hankalaa, koska niiden rajat ovat häilyvät ja ne voisivatkin olla useassa luokassa samanaikaisesti. Luokittelu onkin täysin riippuvainen lokin tietosisällöstä. [1, s. 29.]

Helpon saatavuuden takaamiseksi lokitiedostojen jaottelu on tehtävä suunnitelmallisesti. Lokit ovat olemassa sitä varten, että niitä voidaan tarvittaessa tarkastella. Helppo saatavuus on avainasemassa tarkastelun nopeuttamiseksi. Vähäistä lokitusta ei kannata hajauttaa liian laajalle, eikä toisaalta raskasta lokitusta tule tallentaa yhteen paikkaan. [3.]

Ylläpitoloki sisältää tietoja käyttöoikeuksien muutoksista, rekistereiden virhetilanteiden hallinnasta ja järjestelmän muutoksista. Käyttöoikeudet määrittävät tietyn käyttäjän tai ryhmän oikeudet järjestelmässä. Tämä loki on olemassa helpottamaan järjestelmän ylläpitäjän työtä. Esimerkiksi jälkeen päin voi käydä tarkistamassa, minkälainen käyttäjä tietylle henkilölle on luotu ja millaiset oikeudet käyttäjällä on. [1, s. 30.]

Käyttöloki sisältää tietoja eritellysti onnistuneista ja epäonnistuneista kirjautumisyrityksistä, käyttöoikeuksien vaihdosta, tietokannan hauista tuloksineen, tulostuksesta ja tallennuksesta. Tämä loki voi paljastaa käyttäjätietojen väärinkäytön. Automaattitoimenpiteenä voi olla esimerkiksi käyttäjän automaattinen jäädyttäminen, minkä uudelleen avaaminen tarvitsee ylläpidon manuaalisen aktivoinnin. [1, s. 30.]

Muutoslokin tulee sisältää tiedot muutoksista tietosisältöön, järjestelmäparametreihin ja konfiguraatiotiedostoihin. Tämä loki on hyödyllinen, jos huomataan tietyn toiminnon menneen pieleen tietyllä aikavälillä. Jälkikäteen voidaan tarkistaa, mitä asetusmuutoksia on tehty, että ohjelmisto on alkanut toimia väärin ja palauttaa vanhat arvot. [1, s. 30.]

Virhelokin tulee sisältää tietoja erilaisten järjestelmien, tapahtumien ja rekisterien virheistä sekä epäjatkuvuuksista. Lokin avulla voidaan virhetilanteen ilmaantuessa tarkistaa, mikä meni vikaan ja korjata ongelma. [1, s. 30.]

2.2 Käsittelyoikeudet

Lokitiedoston suunnitteluvaiheessa pitää miettiä, miksi lokeja halutaan kerätä ja analysoida. Ylimääräisen tiedon tallennus lisää kustannuksia ja hankaloittaa analysointia. Lokien käsittelyn tarve perustuu henkilön toimenkuvaan ja tehtävään. [1, s. 45–46.]

Käsittelyoikeus lokeihin perustuu lakiin. Lokeja on paljon erilaisia, ja niihin kohdistuva tarve ja käsittelyoikeus riippuu siitä, millaista tietoa lokit sisältävät. Kaikkien henkilöiden ei tarvitse nähdä kaikkia ohjelmiston lokeja. Käsittelyoikeudet rajataan rooleittain, ja eri rooleilla on eri käyttöoikeudet. Jos kyseessä on sähköiseen viestintään liittyviä tunnistamistietoja, täytyy noudattaa tietosuojalakia. Käsittely on silloin sallittua vain seuraavissa tilanteissa: palvelujen toteuttamiseksi ja käyttämiseksi, laskutusta, markkinointia tai teknistä kehittämistä varten, maksupalvelun väärinkäyttötapauksissa, teknisen vian havaitsemiseksi tai tietoturvallisuuden huolehtimiseksi. [1, s. 46.]

Uuden järjestelmän kehitysvaiheessa on hyvä miettiä, kuinka paljon lokitusta oikeasti tarvitaan. Liian raskas lokitus vie turhaan suoritusaikaa ja muistia. Toisaalta liian kevyeen lokitukseen ei välttämättä jää kaikki tarpeellinen tieto, mitä voitaisiin tarvita myöhemmin. Lokitus on täysin järjestelmäkohtaista, joissain järjestelmissä lokitetaan kaikki tapahtumat ja toiminnot, kun toisessa ei tarvita minkäänlaista lokitusta. [3.]

Kaikkea ei aina haluta eikä kannata lokittaa. Jos lokitetaan henkilötietoja EU-alueella, pitää siitä tehdä tietosuojaseloste ja noudattaa EU:n tietosuoja-asetusta. On mietittävä tarpeellisuus suhteessa käyttötarkoituksiin. Pääsääntöisesti ei kannata tallentaa henkilötunnuksia, arkaluonteisia tietoja, luottokorttinumeroita, salasanoja, käyttöavaimia, valtuutustietoja tai henkilöiden välistä viestiliikennettä. [2.]

Suunnitteluvaiheessa kannattaa ottaa huomioon, tarvitseeko tietty loki erityiskäsittelyä esimerkiksi säilömisessä. Arkistoidut lokitiedot täytyy periaatteessa poistaa tai anonymisoida, jos käsittelylle ei ole enää tarvetta [2]. Anonymisointi tarkoittaa henkilötietojen muokkausta siten, ettei henkilö ole enää tunnistettavissa siitä [4].

2.3 Analysointi

Lokitiedostojen analysointi on hallinnan ja käsittelyn vaativin ja usein myös tärkein tehtävä. Analysointi tarvitsee hyvät hallintatyökalut ja automatisoidun ympäristön, jotta se veisi vähemmän aikaa. Analysointi koetaan usein tehottomana ja hitaana toimenpiteenä. Analysointia nopeuttaa ohjelmiston tunteminen ja lokien perusasioiden ymmärtäminen, koska käyttäjä huomaa tehokkaammin epäsäännölliset lokitiedot. [1, s. 47.]

Mittavissa ympäristöissä lokien manuaalinen analysointi on erittäin työlästä, jopa mahdotonta. Tällaisten ohjelmistojen ympärille voidaan rakentaa automaattista suodattamista tekeviä työkaluja. Edellytyksenä kuitenkin pitää tunnistaa hyödylliset lokit hyödyttömistä. Ylimääräisten lokitietojen suodatuksen jälkeen manuaalisesta analysoinnista tulee huomattavasti tehokkaampaa. Automaattiset työkalut voidaan myös yhdistää laukaisemaan toimintoja, esimerkkinä järjestelmänvalvojien hälytys. [1, s. 47.]

3 Kehitystyökalut ja tekniikat

Tässä luvussa käsitellään opinnäytetyön toteutuksessa käytettyjä kehitystyökaluja ja tekniikoita. Tarkoitus on avata, mitä käytettiin ja mikä niiden hyöty on, sekä kertoa valintaperusteista.

3.1 Qt

Ohjelmistokehittäjä Qt Group on suomalainen julkinen osakeyhtiö. Qt Group Oyj syntyi vuonna 2016 Digia Oyj:n jakautumisen yhteydessä [5]. Qt-teknologia on käytössä yli 70 toimijalla, joista tärkeimmät asiakasryhmät ovat teollisuusautomaatio, lääketeollisuuden laitteet ja autoteollisuus. Teknologiaa käyttäviä ohjelmistokehittäjiä on globaalisti yli miljoona. Qt:n työkalut ohjelmistokehitykseen on rakennettu avointa lähdekoodia hyödyntäen. Itseasiassa osan Qt-teknologiakehityksestä tuottaa itsenäisesti toimiva laaja avoimen koodin kehittäjäyhteisö. [6.]

Vuonna 2019 tilikauden lopussa Qt:n henkilöstömäärä oli 340. Tästä osuudesta Suomessa oli noin yksi neljäsosa. Liikevaihto koostuu kaksi kolmasosaa lisenssimyynnistä ja konsultoinnista, yksi kolmasosa tulee ylläpitotuotoista. [7.]

Opinnäytetyössä kehitysympäristönä toimi Qt Creator. Kehitysympäristön valinta perustui toimeksiantajayrityksen yleiseen Creatorin käyttöön. Creator on Qt Groupin yksi päätuote ohjelmistokirjaston rinnalla. Creator toimii Windows-, Linux- ja macOS-käyttöjärjestelmillä. Sillä luotavat ohjelmat ja ohjelmistot menevät monille eri työpöytäsovelluksille, sulautettuihin järjestelmiin sekä suosituimmille mobiililaitteille. [8.]

Creatorissa on paljon ominaisuuksia. Uuden projektin aloitukseen on luotu erilaisia mallipohjia, joilla kehittäjä pääsee nopeasti alkuun. Projektinhallinta pitää lähdekoodit, konfiguraatiotiedostot ja muut tarvittavat resurssit tallessa. Projektinhallintaan voidaan liittää suoraan versionhallintatyökalu, jonka avulla versioiden väliset erot pysyvät tallessa. [9.]

Käyttöliittymän suunnitteluun Qt on luonut Qt Designer -nimisen ohjelman. Designerilla voidaan luoda nopeasti graafisia käyttöliittymiä. Designer toimii raahaa ja pudota -tyylisen sivuvalikon ja objektien konfiguroinnin avulla, eikä tämä vaihe sisällä koodaamista. [10.]

Kun ohjelmistonkehitys on saatu käynnistettyä ja graafisen käyttöliittymän alustava paikalleen asettelu on tehty, päästään itse ohjelmointiosioon. Creatoriin on määriteltävissä käyttäjän haluamat asetukset aina fontin säädöstä automaattiseen tekstin arvaukseen ja kaikkeen siltä väliltä. [11.] Lisäksi on mahdollista liittää käyttäjän haluamia lisäosia, joilla saadaan laajennettua uusia tarvittavia ominaisuuksia, esimerkkinä koodin staattiseen analysointiin käytettävä CppCheck ja koodin dynaamiseen analysointiin oleva Valgrind [11] [12] [13]. Staattinen analyysi tarkoittaa lähdekoodin tutkimista suorittamatta sitä, kun taas dynaaminen tarkastelu tapahtuu ohjelman suoritusaikana [14, s. 7].

Creatorissa on määriteltävissä erilaisia rakennussarjoja, joilla ohjelmisto saadaan käännettyä tietylle alustalle. Alustan lisäksi rakennussarjaan kuuluu kääntäjä, virheidenetsintätyökalu ja ennalta määrätty Qt:n kirjastoversio. [15.] Käännetty ohjelmisto voidaan suorittaa suoraan työpöytäsovelluksessa, emuloidulla laitteella tai kehitystietokoneeseen kiinnitetyllä laitteella [16]. Emulointi tarkoittaa toisen laitteiston jäljittelemistä samojen tulosten saamiseksi kuin alkuperäisellä laitteella [17].

3.2 WebSocket

WebSocket on tietokoneiden viestintäprotokolla, joka tarjoaa kaksisuuntaisen keskusteluväylän TCP-yhteyden avulla. TCP:stä tarkemmin luvussa 3.2.1. WebSocket-protokolla on standardisoitu vuonna 2011 IETF:n toimesta, joka vastaa yleisesti internet-protokollien standardoinnista [18]. WebSocketin tarkoitus on luoda pistoketyyppinen ratkaisu selainpohjaisten sovellusten tarpeelle käyttää kaksisuuntaista keskusteluväylää selaimen ja palvelimen välillä. Tämä myös poistaa tarpeen monelle HTTP-yhteydelle, jonka aikaisemmin kehitetyt teknologiat ovat vaatineet. [19.]

3.2.1 TCP

TCP on tietoliikenneprotokolla, joka on yksi TCP/IP-protokollaperheen jäsen. TCP:n päätarkoitus on luoda luotettavia yhteyksiä tietokoneiden välille ja toimia yhteytenä tietoverkoissa. Se ei odota luotettavuutta alemmilta protokollilta, joten sen täytyy taata se itse. [20, s. 150.] [21.]

WebSocket pohjautuu TCP:hen. Sovelluksen näkökulmasta TCP lähettää jatkuvan datavirran verkkoon. Sovelluksen itse ei tarvitse pilkkoa dataa erilaisiin osiin tai paketteihin. TCP tekee tämän

lajittelemalla tavut TCP-lohkoihin, jotka menevät IP-kerroksen kautta lähettäväksi määränpäähän. TCP päättää itse, kuinka data lohkotaan ja milloin data lähetetään. Jokaisen lähetettävän lohkon ensimmäiselle tavulle määrätään sarjanumero, ja jokaisen tavun saapumisesta vastaanotetaan vahvistus. Jos data ei saavu määränpäähän tietyn ajan kuluessa, se lähetetään uudelleen. Sarjanumeroidun datan ansiosta pilkottu data saadaan kasattua samaan järjestykseen vastaanottavassa päässä; puuttuvat ja kopiopaketit huomataan. Edellä mainittujen omaisuuksien ansiosta yhteys on todella luotettava. [20, s. 151–152.]

Kun vastaanottava osapuoli kuittaa paketin vastaanotetuksi, se myös ilmoittaa, kuinka paljon kapasiteettia on vastaanottaa lisää dataa ongelmitta. Mahdollisia ongelmia ovat datan ylikirjoitus ja sisäisten rekisterien ylivuotaminen. Tämä virtauksenhallinta edelleen nostaa yhteyden luotettavuutta. TCP on kaksisuuntainen järjestelmä, mikä tarkoittaa, että molemmat osapuolet toimivat sekä lähettäjinä että vastaanottajina. Datan lähettäminen voi tapahtua samanaikaisesti. [20, s. 151.]

3.2.2 WebSocketin hyödyt

Historiallisesti välitöntä keskustelua vaativat selainpohjaiset ohjelmistot ovat vaatineet HTTP-kyselyjen hyväksikäyttämistä erillisillä päivityskyselyillä normaalin päivityksen rinnalla. Tästä aiheutuvia ongelmia ovat pakotus palvelimelle käyttää montaa erillistä TCP-yhteyttä per asiakas, pakettien iso koko ja selaimen vaatimus tehdä ylimääräistä työtä yhteyksien seuraamisessa. Pakettien iso koko johtuu jokaisen viestin sisältämistä HTTP-protokollan otsikkotiedoista. [19, s. 3.]

WebSocket on luotu ratkaisemaan edellä mainittuja ongelmia yksinkertaisemmalla ratkaisulla, joka käyttää vain yhtä TCP-yhteyttä. Se soveltuu moniin eri selainpohjaisiin ohjelmiin, joita voisivat olla esimerkiksi pelit, osakemerkinnät, usean käyttäjän sovellukset ja reaaliaikaiset palvelimen paljastavat käyttöliittymät. WebSocket on suunniteltu syrjäyttämään aikaisemmat kaksisuuntaiset kommunikaatioteknologiat, jotka käyttävät HTTP:tä kuljetuskerroksena. Tarkoituksena on saada mahdollisimman paljon hyötyä nykyisestä infrastruktuurista, kuten välityspalvelimista, suodatuksesta ja tunnistautumisesta. [19, s. 3.]

3.2.3 Toimintaperiaate

Protokolla koostuu kahdesta eri osasta, jotka ovat kättelyosuus ja dataliikenneosuus. Kättelyvaihe alkaa selaimen lähettämästä avauspyynnöstä, jonka täytyy olla yhteensopiva HTTP-pohjaisen palvelimen ja sen välitysohjelmiston kanssa. Avauspyynnössä pyydetään palvelimelta protokollan vaihtoa WebSocketiin. [19, s. 4–5.] Avauspyyntö on ainut relaatio HTTP-protokollaan. Tämän jälkeen kaikki juttelu tapahtuu suoraan TCP:n kautta [19, s. 10].

Kun yhteys on onnistuneesti luotu, molemmat osapuolet voivat lähettää dataa milloin tahansa. Tässä vaiheessa viestejä voidaan lähetellä vapaasti edestakaisin. Viestit ovat protokollassa tarkkaan määriteltyjä kehyksiä, joita voidaan lähettää yksi tai useampi kerralla. On myös tilanteita, joissa kehys joudutaan pilkkomaan osiin lähetyksen ajaksi. Jokainen kehys sisältää vain yhden tyyppistä dataa. Tämä data voi olla tekstiä, binääridataa tai ennalta määriteltyjä ohjauskehyksiä. Protokollan kirjoitushetkellä määriteltyjä datatyyppejä on varattu kuusi ja reserviin on jätetty kymmenen tyyppiä myöhempää laajennusta varten. [19, s. 4–5.]

Yhteyden sulkeminen on huomattavasti yksinkertaisempaa kuin avaaminen. Toinen osapuolista lähettää yhteyden sulkemispyynnön, jonka seurauksena toinen osapuoli lähettää lopetukseen liittyvän vahvistusviestin. Kun lopetusviesti on vastaanotettu, voidaan yhteys sulkea turvallisesti, sillä tiedetään, ettei toinen osapuoli ole lähettämässä enää muuta dataa. [19, s. 8.]

3.3 Web-kehitystyökalut

Opinnäytetyössä käytettiin useita eri web-ohjelmointiin liittyviä ohjelmointikieliä. Ensimmäisenä mainittakoon HTML, jolla luodaan rakenne verkkosivuille. HTML:n yhteydessä käytetään yleensä laajennuksia, joilla saadaan lisätoiminnallisuutta käyttöön. Työssä käytettiin muotoilukielenä yleisesti käytössä olevaa CSS:ää ja toiminnallisuutta lisäävää skriptikieltä JavaScriptiä. [22.]

Työssä oli myös käytössä PHP, joka on avoimeen lähdekoodin perustuva yleiskäyttöinen skriptikieli. Vaikka kyseessä onkin yleiskäyttöinen skriptikieli, on se erityisen sopiva web-kehityksessä ja se voidaan upottaa HTML-tekstin kanssa samaan tiedostoon. [23.] Työssä palvelimella oli käytössä Apache HTTP Server. Apache on ilmainen, tehokas ja joustava avoimeen lähdekoodiin perustuva HTTP-palvelinohjelmisto [24].

3.4 Linux ja skriptit

Työssä käytettiin Linux-pohjaista käyttöjärjestelmää. Linux on Linus Torvaldsin kirjoittama käyttöjärjestelmä, joka on kopioitu Unix-käyttöjärjestelmästä [25]. Käyttöjärjestelmä on suuri kokonaisuus pienemmistä ohjelmistokomponenteista rakennettuja toiminnallisuuksia, jotka hyödyntävät käytettävää laitteistoa [26].

Perinteiset käännettävät ohjelmointikielet, kuten C, C++ ja Lisp, on suunniteltu rakentamaan datarakenteita ja algoritmeja tyhjästä. Tyypillisesti tämänkaltaiset ohjelmointikielet on voimakkaasti kirjoitettu eli datatyypit ovat määritelty tietyntyyppisiksi. Tällaisia tyyppejä ovat esimerkiksi kokonaisluvut, liukuluvut, merkit ja merkkijonot. Näillä ohjelmointikielillä on tarkoitus tehdä ohjelmia, jotka toimivat ympäristöissä, missä ei oleteta olevan valmiiksi asennettuja komponentteja. [27, s. 23–24.]

Skriptikielet, kuten Perl, Python, Rexx, Tcl, Visual Basic ja Unix shell, esittävät hyvin erilaista ohjelmointityyliä verrattuna edellä mainittuihin ohjelmointikieliin. Skriptikielet olettavat, että käytettävään järjestelmään on asennettu joukko eri kielillä kirjoitettuja hyödyllisiä komponentteja. Niillä ei ole tarkoitus kirjoittaa uutta ohjelmaa tyhjästä, vaan yhdistää komponentteja keskenään. Skriptikielillä on tarkoitus laajentaa olemassa olevien komponenttien ominaisuuksia. On harvinaista, että skriptissä toteutetaan monimutkaisia algoritmeja tai datarakennelmia. Tyypillisesti monimutkaisuus tehdään käyttäjälle näkymättömissä komponenttien sisällä. [27, s. 24–25.]

Työssä kehitettävä pääohjelma suunnitellaan siten, ettei ole väliä, millä skriptikielellä ohjelmaan ladattavat skriptit kirjoitetaan. Kunhan skriptejä suorittava ohjelmisto on asennettu oikein ja skriptien alkuun asetettava polku on oikein. Esimerkkiskriptikieliä, joilla skriptejä voidaan kirjoittaa: Bash, Perl, PHP, Python.

3.5 SQL

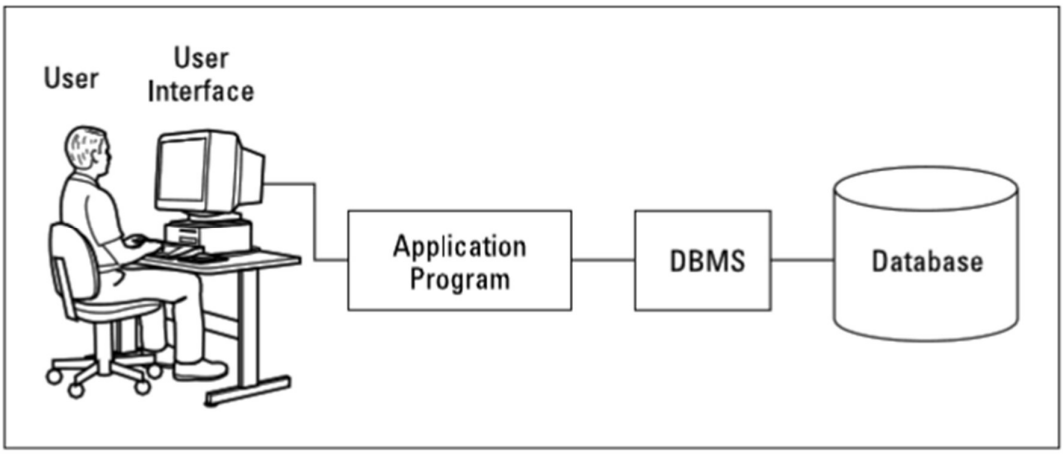
SQL tarkoittaa rakenteellista kyselykieltä. Sen tarkoitus on mahdollistaa tietokantojen luominen, datan lisääminen ja muokkaus kantaan sekä datan kysely kannasta. IBM kehitti kielen 1970-luvulla, ja siitä on kehittynyt teollisuusstandardi. [28, s. 5.]

Yleisesti datan tallennuksessa halutaan, että datan tallennus on nopeaa ja helppoa, koska sitä tehdään usein. Datan tallennuspaikan tulee olla luotettava, jottei data häviä. Datan hakeminen

pitää olla nopeaa siitä huolimatta, miten paljon tietoa on tallennettu. Lisäksi tahdotaan, että haluttu data voidaan erotella helposti muusta datasta, mitä juuri tällä hetkellä ei tarvita. [28, s. 6.]

Tietokannan merkitys terminä on laajentunut viime aikoina. Tässä dokumentissa se määritellään itseselitteiseksi keräelmäksi tallenteita. Tietokannassa on dataa ja metadataa, joka on kuvaelma datan rakenteesta. Tämä oman rakenteen kertova metadata tekee tietokannasta itseselitteisen. Tallenne on esitys fyysisestä tai ajatuksellisesta objektista. Esimerkiksi tallenne voisi olla yrityksen asiakastieto, josta käy ilmi nimi, osoite ja puhelinnumero. [28, s. 7.]

Tietokantaa ohjataan tietokannan hallinnointisysteemillä, myöhemmin DBMS. DBMS sisältää joukon ohjelmia, joilla voi määritellä, ylläpitää ja prosessoida tietokantoja ja niihin liittyneitä ohjelmia. Markkinoilla on monia DBMS-ohjelmistoja, ja niitä on tarjolla eri käyttötarkoituksiin. [28, s. 8.] Suosituimpia DBMS-ohjelmistoja ovat Oracle, MySQL, Microsoft SQL Server, PostgreSQL ja MongoDB [29]. Kuvassa 1 havainnollistus loppukäyttäjän reitistä tietokantaan.



Kuva 1. Havainnollistus loppukäyttäjän reitistä tietokantaan [28, s. 9].

Kuvaa purettaessa pienempiin palasiin voidaan havaita projektissa käytettävät osuudet. Käyttöliittymänä (engl. user interface) ja sovellusohjelmistona (engl. application program) toimivat opinnäytetyössä tehtävät ohjelmistot. DBMS on ohjelmisto, jolla ohjataan tietokantaa (engl. database).

4 Ohjelmiston toteutus

Ohjelmiston toteutuksessa selvitettiin lähtötilanne ja vaatimukset. Tämän jälkeen tehtiin suunnitelma, jota lähdettiin toteuttamaan. Ohjelmistokehitysprosessin aikana tehtiin jatkuvaa testausta.

4.1 Lähtötilanne

Ennen opinnäytetyön aloittamista analysaattori tulosti kiintolevylle standardisoimatonta lokidataa. Kun prosessissa ilmeni ongelmia, kehittäjä otti etäyhteyden kyseiseen tietokoneeseen ja lähti selaamaan lokitiedostoja yksitellen ja etsimään mahdollisesta viasta syntyviä lokitietoja. Ongelmaksi tässä koitui hidas yksittäisten tiedostojen selaaminen sekä vaatimus Linuxin käytön osaamisesta. Opinnäytetyössä tehtävä pääohjelma vastaa tähän tarpeeseen.

Web-käyttöliittymän osalta valmiiksi toimiva ympäristö on jo olemassa. Tätä ympäristöä laajennetaan työn kannalta tarpeellisin tavoin, jossa tullaan hyödyntämään WebSocket-protokollaa kommunikoinnissa. Tavoitteena on tuoda lokeista esille mielenkiintoisimmat ja mahdollisia poikkeamia sisältävät tiedot sekä samalla poistaa Linuxin osaamisen käyttövaatimus. Lisäksi tavoite on lisätä työn tehokkuutta, madaltaa kynnystä systeemin kunnon tutkimiseen ja avata mahdollisuuksia automaation parempaan hyödyntämiseen tulevaisuudessa.

4.2 Vaatimukset

Vaatimusmäärittelyn toiminnallisessa puolessa mukana oli teknisiä asiantuntijoita, jotka ovat loppukäyttäjiä. He eivät ole kehittäjiä, vaan heidän tuomat vaatimukset pyrittiin huomioimaan mahdollisimman hyvin ja pyrittiin rakentamaan ohjelmiston arkkitehtuuri ja toiminnallisuus vastaamaan tätä tarvetta.

Kehitettävällä ohjelmistolla täytyy pystyä keräämään satunnaisessa asettelussa olevat lokitiedostot. Kerätyistä lokitiedostoista pitää pystyä karsimaan ja jäsentämään tarvitun muotoista dataa, joka voidaan puskea SQL-tietokantaan myöhempää säilytystä varten. SQL-rajapinnan täytyy olla tarpeeksi ketterä, jotta erilaisia SQL-kyselyitä voidaan lisätä käyttöliittymän vaatimin tavoin. Ohjelmisto kehitetään C++-ohjelmointikielellä, ja kohdekäyttöjärjestelmä on Linux.

Datan kerääminen järjestelmästä tapahtuu erilaisin skriptein. Nämä skriptit voidaan kirjoittaa millä tahansa skriptikielellä, mikäli polut ja skriptikielten vaatimat binääritiedostot on oikein asennettu. Skriptejä voidaan lisätä ja poistaa sen mukaan, millainen tarkkailu järjestelmälle halutaan asettaa. Pääohjelma suorittaa näitä skriptejä erillisen konfiguraatiotiedoston mukaisesti. Lisäksi täytyy ottaa huomioon järjestelmän yleinen prosessorikuorma. Jos kuormaa on paljon ja muut tärkeät prosessit voisivat jäädä ilman tarvitsemaansa prosessointiaikaa, täytyy lokien käsittelyohjelman suoritus keskeyttää resurssien vapautumisen ajaksi.

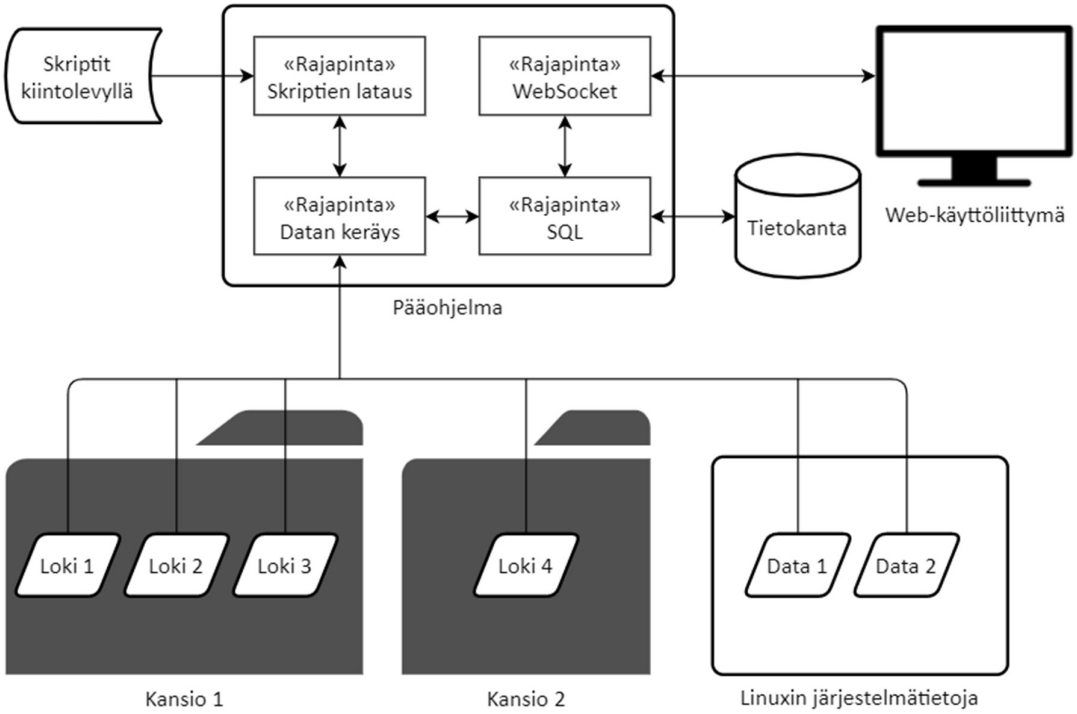
Web-käyttöliittymästä täytyy tehdä helppokäyttöinen, jotta sen käyttäminen olisi mahdollisimman vaivatonta kohdekäyttäjäkunnalle. Käyttöliittymästä täytyy nähdä haluttujen asetuksien mukaiset tiedot vaivattomasti, ja sieltä pitää myös pystyä muokkaamaan tiedonkeräyskonfiguraatiota. Käyttäjä saa yhteyden käyttöliittymään selaimen avulla sisäverkosta. Linuxilla HTTP-palvelinohjelmana on Apache.

Keskustelu näiden kahden ohjelmiston välillä tapahtuu WebSocket-kommunikaatioprotokollan välityksellä. Data lähetetään JSON-muodossa, joka mahdollistaa standardifunktioiden käyttämisen käyttöliittymän lähdekoodissa. JSON on tekstisyntaksi, joka helpottaa tietorakenteiden vaihtoa ohjelmistojen välillä [30].

4.3 Suunnittelu

Vaatimusmäärittelyn jälkeen voidaan aloittaa ohjelmistojen suunnittelu. Opinnäytetyössä tehtävä pääohjelma ei pohjaudu mihinkään valmiiseen ohjelmaan tai kirjastoon, joten toteuttaminen on hyvin vapaamuotoista. Tarvittavat rajapinnat pitää tietysti tehdä vaatimusten mukaisesti ja

ajatella mahdollisia myöhempiä muutoksia ohjelmistoon. Kuvassa 2 pääohjelman suunnitelma rajapinnoista.



Kuva 2. Suunnitelma pääohjelman rajapinnoista.

Web-käyttöliittymän tekeminen on suoraviivaisempaa, sillä valmis ohjelmistoarkkitehtuuri on suunniteltu ja toteutettu jo aikaisemmin yrityksen toimesta. Tässä työssä tehtävä osuus jääkin laajennuksen tasolle. Laajennukseen täytyy tehdä kuitenkin ohjelmistoa sekä näkyville käyttöliittymään että näkymättömiin dataliikenteen ja -muokkauksen osalta.

Tietokannan valinnassa piti ottaa huomioon ohjelmointia ja käytettävyyttä koskevia asioita. Ohjelmoinnin kannalta valmis ohjelmointikirjasto Qt-ympäristöstä nopeuttaa huomattavasti tietokannan käyttöönottoa. Käytettävyyden kannalta haluttiin tietokanta, joka vastaa parhaiten opinnäytetyön vaatimuksiin. Data pääohjelmalta on aina tietyssä muodossa, eikä vaatimuksissa ole horisontaalista skaalautuvuutta tietokannan suhteen, joten SQL-tietokanta on parempi vaihtoehto kuin NoSQL-tietokanta. Lisäksi vaatimusmäärittelyssä tietokantana oli SQL-tietokanta.

4.4 Pääohjelman toteutus

Pääohjelmaa lähdettiin toteuttamaan datan keräämisen osuudesta. Koska datan keräys haluttiin tapahtuvan skripteillä, ensimmäisenä lähdettiin kehittämään ominaisuus, joka mahdollisti ohjelman ulkopuolisten skriptien suorittamisen. Skriptien suoritus haluttiin ajoittaa tapahtumaan tietyllä toistolla, joten sille täytyi jotenkin määritellä aika.

Tämä johti tarpeeseen konfigurointitiedostolle, jonne tulevia skriptejä pystyttiin määrittelemään. Tiedostoon tuli skriptin ajan lisäksi sijaintipolku, toiminto ja nimi. Ajalla pystyttiin määrittelemään, kuinka usein kyseinen skripti suoritetaan. Toiminto vaikutti pääohjelman päätöksen tekoon datan tallentamisen osalta. Nimen tehtävä oli nimetä skriptin suorittaminen pääohjelmalle, jotta lokitiedoston tarkistelu olisi mahdollisimman helppoa mahdollisten virhetilanteiden ilmentyessä.

Tässä vaiheessa tehtiin muutamia erilaisia testausskriptejä eri skriptikielillä, jotta toiminnallisuus voitiin todeta toimivaksi. Testiskripteillä tehtiin yksinkertaisia asioita, kuten tekstin kirjoittamista tiedostoon, olemassa olevan tekstitiedoston suodatusta ja ohjelmien suoritusaikojen kyselyä. Python- ja Bash-skriptin määrittely konfiguraatiotiedostoon kuvassa 3.

```
<script>
    <name>Python print</name>
    <path>./scripts/pythonPrint.py</path>
    <timer>100</timer>
    <type>undefined</type>
</script>
<script>
    <name>Log handler CPU load</name>
    <path>./scripts/cpuLoad.sh</path>
    <timer>100</timer>
    <type>commonscript</type>
</script>
```

Kuva 3. Python- ja Bash-skriptin määrittely konfiguraatiotiedostossa.

Kun skriptit saatiin ladattua ja suoritettua pääohjelmasta käsin, oli tehtävälistalla seuraavaksi virheidenkäsittelyn toteutus. Virheidenkäsittely tarvittiin, jottei pääohjelman suoritus kaadu virheiden ilmaantuessa ja mahdollisesta virhetilanteesta haluttiin lokitiedostoon merkintä tapahtumasta. Mahdollisia virhetilanteita voi syntyä väärin määritellystä skriptistä tai skriptin ajossa tapahtuvasta virheestä.

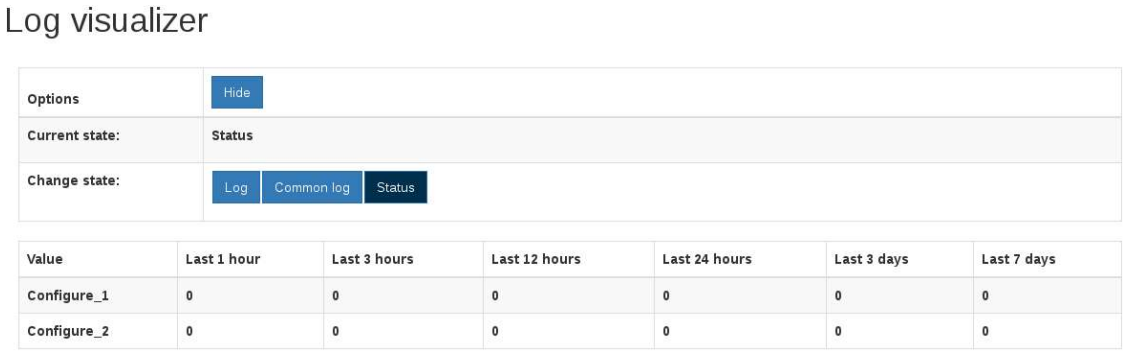
Kun skriptien suoritusosuus oli saatu toteutettua, täytyi data pystyä varastoimaan. Data haluttiin varastoida SQL-tietokantaan. Seuraavaksi pitikin kehittää rajapinta tietokannalle. Rajapinta toteutettiin Qt:n valmiilla tietokantakirjastolla. Ensimmäisessä testausvaiheessa tarvittiin jo itse tietokanta, jota vasten ohjelmaa voitiin testata. Aluksi tietokannassa oli vain yksi testaustaulukko. Tietokanta sai monta kehityskierrosta uusien tarpeiden ilmaantuessa ohjelmiston kehitysvaiheessa.

Datankeräyksen toimiessa testatusti voitiin edetä pääohjelman viimeisen ominaisuuden kehittämiseen, joka oli WebSocket-rajapinta. Rajapinta täytyi tehdä samanaikaisesti web-osuuden kanssa, sillä keskusteluväylää ei olisi muuten voinut testata. Ensin haluttiin luoda pelkkä yhteys palvelimen (pääohjelma) ja asiakkaan (selain) välille. Kun yhteys saatiin onnistuneesti luotua, sitä voitiin lähteä laajentamaan työn vaatimalla tavalla.

Keskusteluun suunniteltiin ja toteutettiin oma viestin kasaus- ja purkamisjärjestelmä. Viestit olivat vaatimusmäärittelyn mukaisesti JSON-muotoisia. Kummankin keskustelupään purkamisvaiheessa viestin alusta luettiin etutunniste, jonka avulla viesti saatiin ohjattua haluttuun prosessointiosioon. Prosessoinnin seurauksena saatiin tehtyä halutut toiminnot, joita esimerkkinä ovat viestin lähettäminen vastapuolelle tai käyttöliittymässä käyttäjälle näkyvä muutos.

4.5 Selainosuuden toteutus

Käyttöliittymä koostuu kolmesta eri osuudesta. Ensimmäisessä osiossa on etsimis- ja tarkastelutyökalu pelkille lokitiedostoille ja toisessa muualta systeemistä tulevalle datalle. Kolmannessa osuudessa on matriisityyppinen taulukko, josta käyttäjä näkee nopeasti ennalta määritetyn kokonaistilanteen järjestelmästä. Kuvassa 4 esitellään matriisityyppinen taulukko.



Kuva 4. Matriisityyppinen taulukko, johon voi konfiguroida haluamansa arvot helposti nähtäväksi.

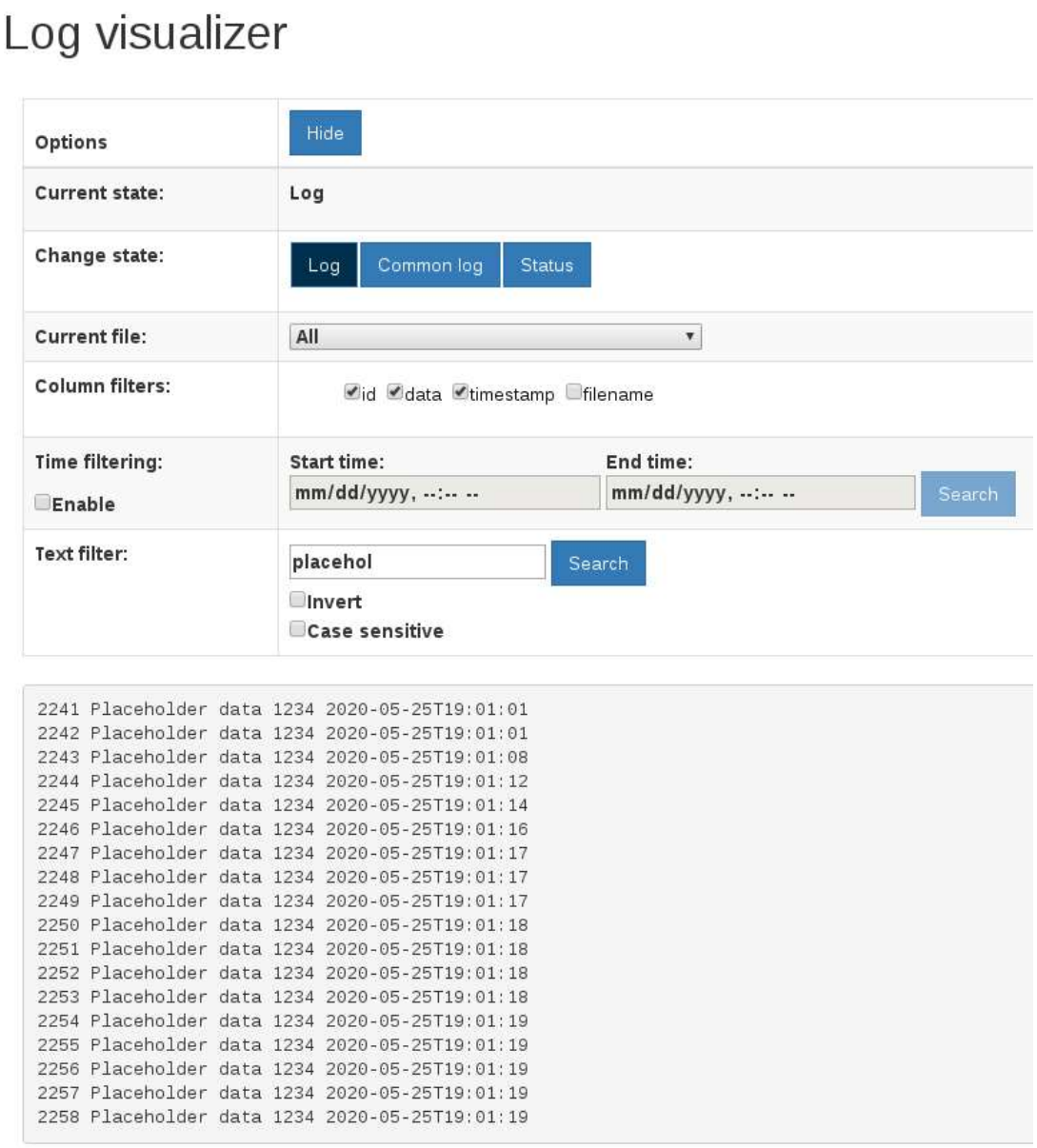
Selainosuuden aloittaminen tapahtui taustalla tapahtuvan WebSocket-dataliikenteen toteutuksella. Aluksi tehtiin jo olemassa olevaan web-käyttöliittymään kaksi uutta sisällöltään tyhjää sivua, jotka seurasivat tiettyä tyylittelyä. Tyhjiin sivuihin lähdettiin toteuttamaan WebSocket-yhteyden ottoa pääohjelmaan. Ongelmien selvitys ja testaus saatiin toteutettua suoraan selaimen konsolista. Onnistuneen yhteyden luonnin jälkeen lähdettiin toteuttamaan logiikkaa datan vastaanottamiselle ja lähettämiselle.

Kun dataliikenne pääohjelmaan oli tehty ja keskustelulogiikka oli kunnossa, lähdettiin tekemään käyttöliittymää käyttäjälle. Tässä vaiheessa oli tärkeää miettiä, mitä kaikkia ominaisuuksia loppukäyttäjä tarvitsee nopeiden ja täsmällisten hakujen tekemiseksi. Käyttöliittymään haluttiin ominaisuus selata kaikkia tiedostoja samanaikaisesti tai ainoastaan yhtä yksittäistä tiedostoa. Tekstikenttä vapaansanan hakuun, mahdollisuus merkkikokoriippuvaisuuteen, hakeminen tietystä aikaikkunasta sekä käänteinen haku. Käyttöliittymää lähdettiin kehittämään ominaisuus kerralla ja aina yhden valmistuttua siirryttiin seuraavaan, kunnes kaikki olivat tehtynä.

Kahteen ensimmäiseen osuuteen web-käyttöliittymän käyttäjä ei voi vaikuttaa, vaan asetukset ovat järjestelmän sisäisessä konfiguraatiotiedostossa, jonne käyttäjällä ei ole pääsyä. Sinne ei ole tehty käyttäjärajapintaa, koska se vaatii myös skriptien tekemistä. Käyttäjälle ei haluta antaa suoraa rajapintaa, joka mahdollistaisi skriptien suorittamisen suoraan käyttöliittymästä. Nämä skriptit tullaan tekemään toimeksiantoyrityksen ohjelmistokehittäjien toimesta erilaisiksi erilaisiin laitekokonaisuuksiin. Web-käyttöliittymän kolmanteen osuuteen käyttäjä voi vaikuttaa. Asetuksien muokkaamiselle on tehty oma sivu, josta näkee kerralla kaikki tarkasteltavat hakusanat. Näitä asetuksia käyttäjä voi muokata helposti suoraan asetussivulta, joka esitellään kuvassa 5.

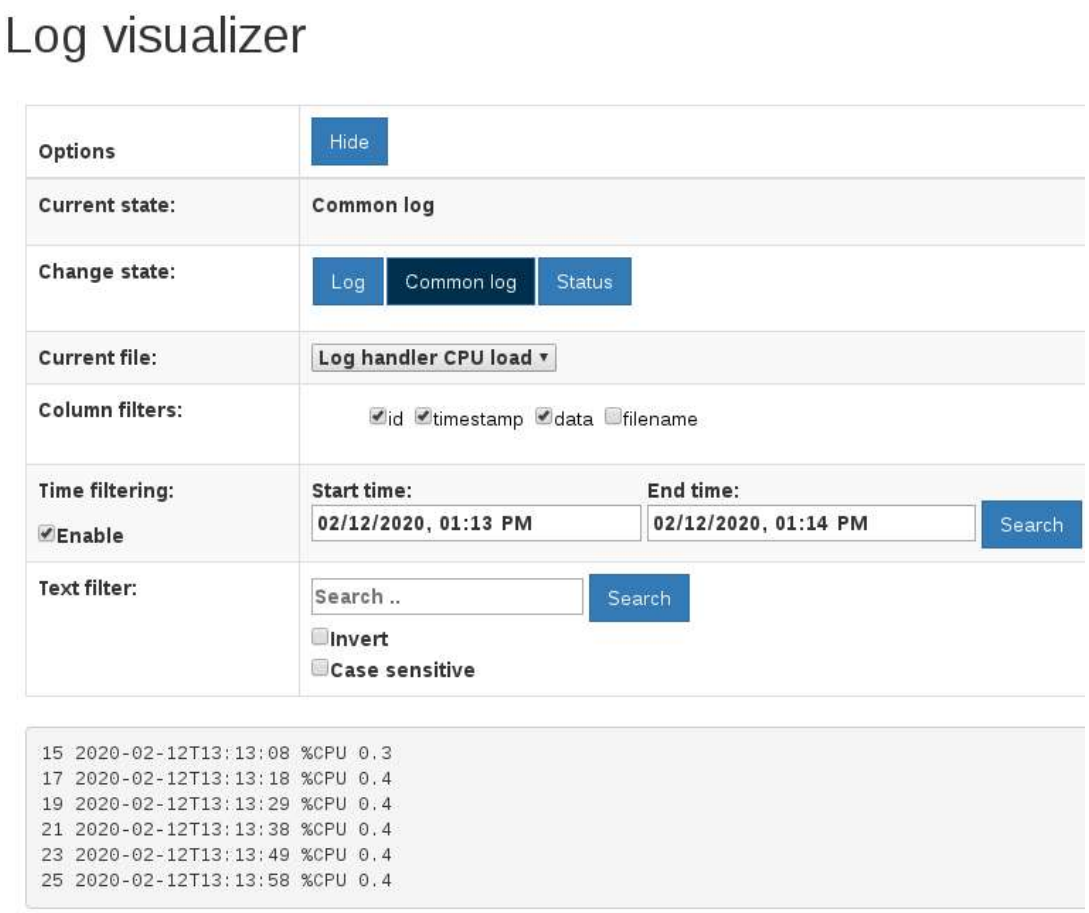
Kuva 5. Matriisinäkymään liittyvä asetuksien muokkaussivu.

Kuvassa 6 lokien tarkastelu- ja etsintätyökalu. Kuvan yläosassa on erilaisia valintapainikkeita, joilla voi suodattaa kuvan alaossassa näkyvää hakutulosta. Tällä saadaan tehtyä tarkkoja hakuja isosta määrästä lokidataa.



Kuva 6. Lokien tarkastelu- ja etsintätyökalu.

Kuvassa 7 esitellään vastaanvan tyyppinen työkalu, jolla etsitään muita järjestelmätietoja. Työkalulla voidaan suodattaa järjestelmätietoja tiedoston, ajan ja tekstinhaun perusteella. Lisäksi tekstihaussa on tarkentavia ominaisuuksia, joilla voidaaan tehdä käänteinen haku tai hakea kirjasimen koon perusteella.



Kuva 7. Järjestelmätietojen tarkastelu- ja etsintätyökalu sekä esimerkki tiedon hakemisesta suodattimilla.

4.6 Testaus

Ohjelmistokehityksessä halutaan tehdä mahdollisimman paljon testausta korkealaatuisen lopputuloksen saavuttamiseksi. Ongelmat halutaan huomata mahdollisimman varhaisessa vaiheessa, sillä niiden korjaaminen on tällöin helpompaa. Testauksen tärkeys korostuu tuotantoon menevissä ohjelmistoissa. Jos ongelmat pääsevät kehitysvaiheesta tuotantoon loppukäyttäjälle, niiden korjaus joutuu käymään uudestaan pitkän kehitystyön läpi.

Ohjelmiston kehitys ja testaus tapahtui virtuaaliympäristössä. Virtuaaliympäristöön oli viety tarvittavat ohjelmistot ja asetettu tietyt konfiguraatiot, jotta tilanne vastaisi mahdollisimman realistisia olosuhteita oikeaan linjastoon nähden. Virtuaaliympäristö ei täysin vastaa kohdelaitteistoa, mutta on tarpeeksi lähellä, jotta ohjelmistokokonaisuuden toimivuus voidaan todistaa.

Ohjelmistoa testattiin manuaalisesti aina seuraavan ominaisuuden valmistuttua kehityksen aikana. Testauksessa todettiin uuden ominaisuuden virheetön toimivuus ja samalla tarkistettiin, ettei vanhoihin ominaisuuksiin tullut odottamattomia muutoksia. Ohjelmiston lopullinen testaaminen oli huomattavasti helpompaa, sillä ohjelmistoa testattiin säännöllisesti koko kehitysprosessin ajan. Lopullisessa testauksessa varmistettiin kaikkien ominaisuuksien toimivan halutulla tavalla ja varmistettiin kokonaisvaltainen ohjelmiston toimivuus.

Kun testauksessa ilmeni ongelmia, ne korjattiin välittömästi ennen työn etenemistä. Testauksessa vastaan tulleita ongelmia olivat esimerkiksi palvelimen ja asiakkaan välillä olevan kommunikointiväylän luominen ja datan muotoileminen tietokantaan. Ongelmien esiintyessä lähdettiin tutkimaan ongelman juurisyytä. Juurisyyn löydyttyä se joko korjattiin suoraan tai kehitettiin kiertotie ongelman välttämiseksi. Testausta jatkettiin niin kauan, kunnes kaikki ongelmat olivat poistuneet.

Lopullisessa testauksessa ei havaittu uusia ongelmia. Tämä oli jatkuvien välitestauksien ansiota. Web-käyttöliittymältä pystyi konfiguroimaan suunnitellut asetukset. Lokitiedostojen tutkimistyökalulla pystyi onnistuneesti käyttämään kaikkia ominaisuuksia. Lopputestauksen jälkeen kuitenkin tiedostettiin mahdollisten piilevien ongelmien olemassaolo. Näitä piileviä ongelmia voi olla välillä vaikea huomata, eikä niitä saada havaituksi kuin laajoilla pitkäaikaisilla testauksilla.

5 Jatkokehitys

Ohjelmistojen jatkokehittäminen on tärkeää, sillä ohjelmistoihin on voinut jäädä puutteita. Näiden puutteiden paikkaamiseksi toimitetaan mahdollisia päivityspaketteja. Ohjelmistot voivat myös vaatia jatkuvaa päivitystä, jotta ne pysyvät ajan tasalla. Ajan kuluessa voidaan havaita uusia tarpeita, jotka voidaan ratkaista ohjelmistokehityksellä. Ohjelmiston toimittaja voi löytää uusia innovaatioita, jotka kehittävät ohjelmiston toimintaa. Tällaiset kehittävät päivitykset voivat tarjota esimerkiksi vakaamman, nopeamman, turvallisemman ja kustannustehokkaamman ratkaisun.

Asiaan perehtyminen sekä tietyn aiheen ympärillä työskentely tuottaa yleensä uusia ideoita ja paljastaa mahdollisia muita kehitystarpeita, jotka eivät olleet alkuperäisessä suunnittelulaajuudessa. Näitä ideoita on hyvä kirjata muistiin tulevaisuuden varalle, vaikkei niitä projektin puitteissa ehditty toteuttaa.

Ohjelmisto on tällä hetkellä prototyyppivaiheessa. Se on vielä irrallinen ohjelmisto, joka pitää yhtenäistää muun ohjelmiston kanssa. Tämä tarkoittaa sitä, että ohjelmistoa vielä kehitetään tai testataan paikallisessa ympäristössä eikä se ole tuotantokäytössä. Prototyyppiohjelmiston pitää olla tarpeeksi vakuuttava ja vakaa, että se voidaan ottaa tuotantokäyttöön. Vakuuttava tarkoittaa määrättyjen ohjelmistotehtävien suorittamista. Vakaalla tarkoitetaan ohjelman pitkää yhtämittaista virheetöntä suorittamista, eikä se saa olla herkkä ulkoisille tekijöille. Tämän jälkeen ohjelmistoa voidaan lähteä jalostamaan tuotantokäyttöön.

Jalostaminen tuotantokäyttöön tarkoittaa laajemman testaamisen toteuttamista oikealla laitteistolla. Jos pitkäaikaisessa testauksessa havaitaan ongelmia, ne tulee korjata. Kun riittävän laaja testaus on toteutettu, ohjelmisto voidaan laittaa kokeiluna tietyn ohjelmistopaketin mukana oikealle tehtaalle ajoon. Vasta kun ohjelmisto otetaan tuotantokäyttöön, siitä on todellista hyötyä sekä toimittajalle että asiakkaalle.

Jatkokehitysideoina ohjelmiston rinnalle voisi toteuttaa erillisen työkalun, jolla voisi tarkistella tietokantaan kerättyä dataa ilman alkuperäistä ohjelmistoa. Työkalun avulla voisi avata tallennetun tietokannan, ilman asennettua tuotanto-ohjelmistoa. Lisäksi ohjelmiston voisi integroida yrityksen toiseen ohjelmistokokonaisuuteen, jotta saisi hyödynnettyä siihen aikaisemmin luotuja työkaluja. Tällaiset työkalut voisivat tarjota esimerkiksi automaattisia toimintoja. Haittana integraatiosta on ohjelmiston monimutkaistuminen.

Tulevaisuudessa ohjelmistolle voisi myös luoda automaattitestejä, jotka paljastavat jatkokehityksessä tapahtuvat ohjelmistovirheet. Nämä virheet voidaan havaita odottamalla tiettyä tulosta erinäisistä toiminnoista aina, riippumatta mitä ohjelmistomuutoksia on tehty.

6 Yhteenveto

Työn tavoitteena oli luoda ohjelmistokokonaisuus, joka koostuu web-käyttöliittymän laajennuksesta, palvelinosuuden suunnittelusta, toteuttamisesta ja testaamisesta. Lisäksi kokonaisuuteen kuului erillisen tietokannan toteuttaminen. Työstä tahdottiin yhtenäinen kokonaisratkaisu, jonka esitteleminen loppukäyttäjälle on konkreettisempaa kuin esimerkiksi pelkän tietokanta-arkkitehtuurin esittely. Opinnäytetyölle asetetut päävaatimukset täyttyivät. Käyttöliittymän olisi voinut viimeistellä hienommaksi ja testausta olisi voinut laajentaa, mutta käytettävä työaika piti rajata.

Ohjelmisto antaa käyttäjälle yhdellä silmäyksellä nopean tilannekatsauksen järjestelmästä. Järjestelmän konfigurointi voidaan tehdä räätälöidysti haluttaessa seurata eri arvoja. Ongelman ilmaantuessa lokitiedoston etsintätyökalulla voidaan etsiä tarkasti mahdollinen ongelman syy. Ohjelmisto tarjoaa helppokäyttöisen rajapinnan selaimen avulla, eikä käyttäjän tarvitse osata käyttää Linuxia. Kokonaisuutena sisältö vastaa hyvin aiemmin esiteltyä vaatimusmäärittelyä.

Kehitystyö oli pääpiirteittäin suoraviivaista, ja se eteni ilman suurempia ongelmia. Tämä johtui siitä, että kehitystyökalut ja ohjelmointikielet olivat entuudestaan tuttuja. Kaikki käytetyt teknologiat eivät olleet alun perin tuttuja, mutta aikaisempi kokemus vastaavanlaisista menetelmistä auttoi omaksumaan ne nopeasti kehitystyön ohessa.

Työn tekemisen mieleisin osuus oli palvelinosuuden kehittämisen kokonaisuus. Tämän osuuden tekeminen oli mielenkiintoista, koska se oli tarpeeksi haastavaa, mutta eteni silti jouhevasti. Työ tehtiin suurilta osin hyvissä ajoin, mutta viimeistely jäi roikkumaan, mikä johti siihen, että osa asioista kerkesi jo unohtua ja tapahtui turhaa kertaamista. Opinnäytetyön tekeminen oli opettavainen ja työläs kokemus. Huomasin, että suomenkielisen dokumentoinnin tekeminen on työläämpää kuin odotin.

Ohjelmisto on tällä hetkellä prototyyppivaiheessa, ja siinä on potentiaalia tuotantokäyttöön. Potentiaalia syntyy ohjelmiston tarpeen vuoksi, yksinkertaisuudesta loppukäyttäjälle ja laajoista käyttömahdollisuuksista. Se kuitenkin tarvitsee vielä muutaman kehitysiteraation ja laajan testaamiskierroksen ennen sitä.

Lähteet

(1) Valtiovarainministeriö – Lokiohje. Saatavilla: https://www.vahtiohje.fi/web/guest/3/2009-lokiohje. Viitattu 15.5.2020.

(2) Kyberturvallisuuskeskus. Saatavilla: https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/ohjeet-ja-oppaat/nain-keraat-ja-kaytat-lokitietoja. Viitattu 9.4.2020.

(3) National Cyber Security Centre. Saatavilla: https://www.ncsc.gov.uk/guidance/introduction-logging-security-purposes. Viitattu 9.4.2020.

(4) Tietosuojavaltuutetun toimisto. Saatavilla: https://tietosuoja.fi/pseudonymisointi-anonymisointi. Viitattu 9.4.2020.

(5) Digia. Saatavilla: https://digia.com/sijoittajat/jakautuminen/. Viitattu 10.4.2020.

(6) Qt. Saatavilla: https://investors.qt.io/fi/qt-as-an-investment/. Viitattu 10.4.2020.

(7) Qt Group Oyj vuosikertomus 2019. Saatavilla: https://mb.cision.com/Public/14183/3035371/8e31ac65e9583363.pdf. Viitattu 10.4.2020.

(8) Qt Documentation. Saatavilla: https://doc.qt.io/qtcreator/index.html. Viitattu 10.4.2020.

(9) Qt Documentation. Saatavilla: https://doc.qt.io/qtcreator/creator-project-managing.html. Viitattu 10.4.2020.

(10) Qt Documentation. Saatavilla: https://doc.qt.io/qtcreator/creator-using-qt-designer.html. Viitattu 10.4.2020.

(11) Qt Documentation. Saatavilla: https://doc.qt.io/qtcreator/creator-editor-functions.html. Viitattu 10.4.2020.

(12) CppCheck. Saatavilla: http://cppcheck.sourceforge.net/. Viitattu 10.4.2020.

(13) Valgrind. Saatavilla: https://valgrind.org/. Viitattu 10.4.2020.

(14) Zaidman A. Scalability Solutions for Program Comprehension Through Dynamic Analysis. 2006.

(15) Qt Documentation. Saatavilla: https://doc.qt.io/qtcreator/creator-building-targets.html. Viitattu 10.4.2020.

(16) Qt Documentation. Saatavilla: https://doc.qt.io/qtcreator/creator-running-targets.html. Viitattu 10.4.2020.

(17) Yleinen suomalainen asiasanasto. Saatavilla: https://finto.fi/ysa/fi/page/Y164068. Viitattu 10.4.2020.

(18) The Internet Engineering Task Force. Saatavilla: https://www.ietf.org/about/who/. Viitattu: 11.4.2020.

(19) The Internet Engineering Task Force. Saatavilla: https://tools.ietf.org/html/rfc6455. Viitattu: 11.4.2020.

(20) Parziale L., Britt D., Davis C., Forrester J., Liu W., Matthews C. & Rosselot N. TCP/IP Tutorial and Technical Overview. IBM Corporation, 1998.

(21) The Internet Engineering Task Force. Saatavilla: https://tools.ietf.org/html/rfc1180. Viitattu 11.4.2020.

(22) Mozilla web docs. Saatavilla: https://developer.mozilla.org/en-US/docs/Web/HTML. Viitattu 11.4.2020.

(23) PHP. Saatavilla: https://www.php.net/manual/en/intro-whatis.php. Viitattu 11.4.2020.

(24) Apache. Saatavilla: https://github.com/apache/httpd. Viitattu 11.4.2020.

(25) The Linux Kernel Archives. Saatavilla: https://www.kernel.org/linux.html. Viitattu 11.4.2020.

(26) Ubuntu. Saatavilla: http://ubuntu.fi/mika-kayttojarjestelma-on/. Viitattu 12.4.2020.

(27) Ousterhout J. Scripting: Higher-Level Programming for the 21st Century. IEEE; 1998.

(28) Taylor A. SQL for dummies. Hoboken For Dummies, 2013.

(29) DB-Engines – DB-Engines Ranking. Saatavilla: https://db-engines.com/en/ranking, viitattu 8.5.2020

(30) Mozilla web docs – Working with JSON. Saatavilla: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON. Viitattu 24.5.2020