Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Ilari Sandborg

Tiedon toistuvan analysoinnin automatisointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Automaatiotekniikka

Insinöörityö

04.05.2020

Tekijä Otsikko Sivumäärä Aika	Ilari Sandborg Tiedon toistuvan analysoinnin automatisointi 27 sivua + 2 liitettä 04.05.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	sähkö- ja automaatiotekniikan tutkinto-ohjelma
Ammatillinen pääaine	automaatiotekniikka
Ohjaajat	lehtori Timo Kasurinen General Manager Antti Marttala

Työn tilaajana toimi Wioss Witron Onsite Services Gmbh. Tilaajan organisaatiossa automaatiojärjestelmän tuottamaa tietoa hyödynnettiin laitteiston tilan ja tuotantoprosessin analysointiin. Useita raportteja ja muita tietolähteitä yhdistelevät, toistuvat tarkistukset ja analysoinnit veivät kuitenkin paljon resursseja ja olivat herkkiä tulkintavirheille. Tämän insinöörityön tavoitteena oli etsiä ja toteuttaa tekninen ratkaisu, jonka pohjalta tilaaja voi tulevaisuudessa automatisoida tiedon keräämisen ja analysoinnin.

Tietoturvallisuus vaatimusten asettamien rajoitusten vuoksi valmiin kaupallisen ohjelmiston käyttäminen osoittautui mahdottomaksi ja kaikki tavoitteen saavuttamiseksi tarvittava ohjelmisto toteutettiin itse. Työtä varten tehtiin kaksi erillistä ohjelmaa. Ensimmäinen ohjelma havaitsee raportointijärjestelmän muodostamat raportit, suodattaa niistä halutut tiedot ja siirtää tiedot perustettavaan SQL-tietokantaan. Toinen ohjelma analysoi tiedot määrättyjen sääntöjen mukaan ja esittää ne tilaajan selainpohjaiseen järjestelmään integroidussa osassa.

Lopputuotteena tilaajalle toteutettiin ohjelmisto, joka kerää tarvittavat tiedot automaatiojärjestelmästä, tallentaa ne tietokantaan sekä analysoi ja esittää ne käyttäjälle. Esimerkkinä toteutettiin laitteiden häiriötiheyden muutoksia tarkkaileva analyysi.

Ohjelmisto on ollut testikäytössä tilaajalla useiden kuukausien ajan ja todettu luotettavaksi ja toimivaksi. Työssä toteutetun ohjelmiston ja esimerkin pohjalta tilaajan on helppo jatkaa analysointiprosessien automatisointia.

Avainsanat	ohjelmointi, analysointi, tietokanta, automatisointi

Author Title Number of Pages Date	Ilari Sandborg Automation of Repeating Data Analyses 27 pages + 2 appendices 4 May 2020
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Electrical and Automation Engineering
Professional Major	Automation Engineering
Instructors	Timo Kasurinen, Senior Lecturer Antti Marttala, General Manager

Thesis project commissioner is Wioss Witron Onsite Services Gmbh. Thesis project commissioner's organization uses the data produced by the automation system for analysis and evaluation of equipment state and production process. The analyses and routine checks involving sourcing of data from multiple sources are draining resources and are prone to errors. The goal of this thesis work was to find and produce a technical solution, that would serve as the foundation for the client's future data automations.

Because of the technical limitations set by the cyber security rules, a commercial software solution was ruled out. The software used to achieve the goal had to be custom programmed for this project. Two different programs were made for this thesis. The first program detects new report files generated by the reporting system, filters the data and stores the data in a new SQL-database. The second program pulls the data from the database, analyses the data and displays the results in a browser-based application that has been integrated to the thesis project commissioner's existing system.

Result for the project was a software package that will collect the necessary information, store them in a database and analyses and displays the results to the user. As an example, an analyze of error frequency changes in the automation system was created.

The program has been in use by a test group for roughly 5 months. There has been no major issues and the software has been stable and reliable. The thesis commissioner is now easily able to use the program to automate rest of the repeating analyses.

Keywords	Programming, data analyzing, database, automation.

Sisällys

1 Johdanto

Työn tilaajana on saksalainen yritys Wioss Witron on Site Services GmbH. Yrityksen liiketoimintana on tuottaa asiakasyrityksilleen automaation kunnossapito-, valvonta- ja asiantuntijapalveluita. Suomessa yritys työllistää vakituisesti yli 160 työntekijää. Yritys on Witron logistik + informatik GmbH:n omistama tytäryhtiö. Witron logistik + informatik GmbH on erikoistunut suunnittelemaan ja rakentamaan täysin automatisoituja logistiikkakeskuksia. Heidän tyypillisiä asiakkaitansa ovat suuret päivittäistavarakaupan toimijat Euroopassa, Pohjois-Amerikassa sekä Australiassa. [1].

Työn kohteena olevan automaatiojärjestelmän omistaja on Suomalainen Inex Partners Oy. Inex Partners Oy on SOK:n tytäryhtiö, joka vastaa koko s-ryhmän logistiikasta. Inex Partnersin rakennuttamat jakelukeskukset sijaitsevat Sipoossa ja sieltä toimitetaan päivittäis- ja käyttötavaroita ympäri Suomea. Kahdessa vierekkäin sijaitsevassa keskuksessa on pinta-alaa noin 270 000 m^2 ja kuljetuksia hoitaa noin 1000 ajoneuvoyhdistelmää joka päivä [2]. Jakelukeskuksen toiminta on pitkälle automatisoitua, ja automaatiojärjestelmän on toimittanut Witron logistik + informatik GmbH. Automaatiojärjestelmän kunnossapidosta vastaa Wioss Witron on Site Services GmbH.

Useimmat automaatiojärjestelmät tuottavat toimiessaan paljon erilaista tietoa. Tilaajayrityksessä järjestelmän tuottamaa tietoa hyödynnetään läpi organisaation: tuotannonsuunnittelussa, tuotannonseurannassa, laitteiston kunnonvalvonnassa sekä huoltosuunnittelussa. Tilaajan automaatiojärjestelmään kuuluu oleellisena osana raportointijärjestelmä, joka esittää käyttäjille tietoa erilaisten raporttien muodossa. Raportointijärjestelmä toimii erittäin hyvin ja sen sisältämä tieto on paikkansapitävää. Käyttäjän kannalta ongelmaksi on kuitenkin muodostunut se, että työtehtäväänsä varten tarvitsemaa tietoa täytyy joka kerta erikseen pyytää raportointijärjestelmästä. Esimerkiksi yksinkertaista laitteiston kunnonvalvontaa tehdessään, käyttäjä lataa tilastotietoa vikojen määristä laitteissa, vertailee saamiaan tietoja aiempiin tietoihin ja näiden tietojen perusteella tekee päätöksen kunnossapidollisten jatkotoimien tarpeellisuudesta.

Insinöörityön tarkoituksena on selvittää, voidaanko raportointijärjestelmän tarjoamaa tietoa saada esille, analysoida sekä välittää työntekijöille, täysin automaattisesti. Työn tavoitteena on löytää ja luoda tekninen ratkaisumalli jota soveltaen tilaaja organisaatiossa voidaan automatisoida toistuvia tarkastustehtäviä.

2 Toimintaympäristö

Jakelukeskuksen toiminnan sydämenä on laaja automaatiojärjestelmä. Laitteiston rungon muodostaa kattava kuljetin verkosto. Kuljettimien tehtävä on siirtää kuormalavat, rullakot, laatikot tai tarjottimet tuotantoprosessin vaiheesta toiseen. Näistä kaikista eri pakkaustyypeistä käytetään termiä kuljetusyksikkö. Kuljettimet jaetaan kolmeen päätyyppiin: ketjukuljettimiin, mattokuljettimiin ja rullakuljettimiin. Ketjukuljettimessa kuljetusyksikköä liikutetaan pyörivien metalliketjujen avulla, mattokuljettimessa kumisen korkeakitkaisen maton avulla ja rullakuljettimella pyörivien teräsrullien avulla. Jokainen kuljetusyksikkö on yksilöity uniikilla tunnuksella, joka on merkitty kuljetusyksikköön viivakoodilla. Kaikkien kuljetusyksiköiden liikettä kuljetinverkostossa seurataan valokenno antureiden ja viivakoodinlukijoiden avulla.

Varastoitavat tavarat varastoidaan hyllyihin käyttäen mekaanisia hissejä. Hissi noutaa tavaran kuljettimelta ja varastoi sen varastohyllyyn. Varastopaikat hyllyissä tunnistetaan X-, Y- ja Z-koordinaattien avulla. X-arvo kertoo hyllypaikan sijainnin vaaka-akselilla ja Y-arvo kertoo sijainnin pystyakselilla. Koska hyllyissä voi olla peräkkäisiä varastopaikkoja, kertoo Z-arvo varastopaikan sijainnin syvyysakselilla. Hissejä on käytössä kaikille eri kuljetusyksikkötyypille ja kaikki tyypit vaativat omanlaisensa hissin sekä varastointihyllyn.

Hissien ja kuljettimien lisäksi kuljetusyksiköiden siirtämiseen käytetään myös kerroshissejä, sekä siirtovaunuja. Kerroshissi on mastoon kiinnitetty kuljetin, jota voidaan siirtää pystysuuntaisesti kerrosten välillä. Siirtovaunut on kiskoilla liikkuvia kuljettimia, joiden avulla voidaan siirtää kuljetusyksiköitä vaakatasossa kuljettimelta toiselle.

Saapuva tavara puretaan automaattisen lavanpurkajan avulla yksittäisiksi tuotepakkauksiksi. Lavan purkaminen tapahtuu alipaineen ja puristuksen avulla niin, että lavasta nostetaan kerroksittain tuotepakkaukset kuljettimelle. Tuotteet varastoidaan hissien avulla odottamaan toimitusvuoroaan.

Tuotteet kerätään myymälöiden tilausten perusteella. Tuotteiden keräystä varten on olemassa erilaisia keräyskoneita. Case Order Machine (COM), kerää muodoltaan vaihtelevia paketteja kuljetusyksikköön matemaattisesti lasketun mallin mukaisesti. Ensin pakkaukset asetellaan oikeaan asentoon kääntimien avulla ja sen jälkeen asetinlaite siirtää tuotepakkauksen haluttuun kohtaan kuljetusyksikköä. Automatic Case Stacking (ACS), pinoaa hedelmä- ja vihanneslaatikoita pinoiksi, jotka siirretään kuljetusyksikön päälle. Case Piece Picking (CPP), puoliautomaattinen keräysasema, jossa keräilijän pysyessä paikallaan, järjestelmä toimittaa hänelle tuotepakkauksia. Keräilijä asettelee työpisteelle saapuvat tuotteet kuljetusyksiköihin. Automaatiojärjestelmä vastaa tyhjän kuljetusyksikön toimittamisesta työasemalle ja valmiin keräyksen pois kuljettamisesta. Dynamic Picking System (DPS), puoliautomaattinen keräysasema, jossa keräilijä poimii pientuotteita hyllyissä olevista keräyslaatikoista ja asettaa tuotteet asiakkaalle toimitettavaan laatikkoon. Automaattihissi toimittaa keräyspaikoille jatkuvasti uusia keräyslaatikoita ja vie valmiita asiakaslaatikoita pois.

Automaattisesti toimiva kiristekalvokone käärii kuljetusyksikön ympärille kiristekalvon, joka auttaa pitämään kuljetusyksikön pystyssä asiakkaalle saakka. Kiristekalvon jälkeen kuljetusyksikköön asetetaan tarra, joka kertoo kuljetusyksikön toimitusasiakkaan ja toimitusosoitteen. Tarrat asetetaan kuljetusyksiköihin automaattitulostimen avulla. Tulostimessa on asetinlaite, joka liimaa tulostetut lähetystarrat tiukasti kuljetusyksikön päätyyn ja sivuun.

2.1 Laitteiston ohjaus

Kaikkia järjestelmän laitteita ohjataan ohjelmoitavilla logiikkaohjaimilla. Logiikkaohjaimia on järjestelmässä useita, jotka vaihtavat tietoa toistensa, sekä taustajärjestelmän kanssa, tietoverkon välityksellä. Yhden logiikkaohjaimen (Programmable logic controller, PLC) ohjaamaa järjestelmän osaa kutsutaan nimellä ”Group Control (GC)”. Jokainen GC on puolestaan jaettu pienempiin loogisiin kokonaisuuksiin, joita kutsutaan nimellä ”Local

Area Control (LAC)”. LAC:n sisällä laitteet erotetaan toisistaan elementti numerolla. Jokaisella GC:lla, LAC:lla ja elementillä on kaksinumeroinen tunnistenumero. Tunnistenumeroiden avulla voidaan jokainen järjestelmän laite yksilöidä. Yksittäisen laitteen tunniste muodostuu seuraavasti: ”XX_YY.ZZ”, jossa XX tarkoittaa GC numeroa, YY LAC numeroa ja ZZ elementin numeroa. Tämän numerosarjan avulla laitoksessa työskentelevät henkilöt tunnistavat laitteet ja erottavat ne toisistaan.

Ohjelmoitavien logiikoiden ja samalla koko laitteiston toimintaa ohjataan hallintaohjelmiston avulla. Hallintaohjelmiston muodostaa ohjelmistokokonaisuus nimeltään ”warehouse management system (WMS)”. WMS-järjestelmä on jaettu kolmeen tasoon.

- Logistic Bus System (LBS) kommunikoi asiakkaan järjestelmän kanssa. Vastaanottaa ja lähettää tietoa esimerkiksi tuotteista, tilauksista ja ostoista.
- Keräysmoduulit toimivat tuotannonohjauksena. Keräystoiminnot laitoksessa on jaettu alueisiin käsiteltävien tuotteiden asettamien vaatimusten mukaan. Yksi keräysalue muodostaa yhden keräysmoduulin.
- Material Flow Control (MFC) reitittää kuljetusyksiköitä koko laitoksessa.

Automaatiojärjestelmän tilan tarkkailua varten on olemassa visualisointi, johon on piirretty järjestelmän kaikki osat värillisinä komponentteina, jotka on järjestetty niin että ne muodostavat laitteiden fyysistä asettelua vastaavan kuvan. Komponenttien värit kertovat kunkin laitteen tilan reaaliajassa. Visualisointi mahdollistaa laitteiston tilan seurannan ja vikatilanteiden nopean paikallistamisen. Yleisimmät värit visualisoinnissa ovat seuraavat:

- Vihreä, laite on automaattitilassa ja toimii normaalisti.
- Keltainen, laite on toiminnassa mutta siinä on havaittu jokin tapahtuma, josta on muodostettu varoitus.
- Punainen, laite on pysähtynyt ja on vikatilassa.
- Sininen, laite on käsiajo tilassa.

2.2 Tiedon saatavuus ja analysoinnin haasteet

Kaikki järjestelmän toimintaan tarvittava tieto sekä myös järjestelmän toimiessaan tuottama tieto, on tallennettu tietokantaan. Tietokantaan on tallennettu esimerkiksi tiedot tilauksista, tuotteista, asiakkaista, varastohyllyjen sisällöstä, säilytyslämpötiloista, tuotteen muodosta sekä tuhansia muita tietoja. Tietokantaan tallennetaan myös laitteiston toimintaan liittyvää tietoa. Tällaisia tietoja ovat esimerkiksi laitteen tilamuutokset, toimintahäiriöt ja suorituskykytiedot. Tietokannassa olevan tiedon tarkastelua varten on WMS ohjelmistoon lisätty komponentti nimeltä Management information system (MIS). MIS tarjoaa käyttäjälle valikoiman raportteja. Raportteja on olemassa n. 1500. Ne on suunniteltu niin, että niiden sisältämät tiedot liittyisivät toisiinsa sekä muodostavat kokonaisuuden, josta saa selville vastauksia yleisimpiin kysymyksiin. Tästä esimerkkinä on raportti, joka näyttää yhden keräysmoduulin keräämien tuotepakkausten kappalemäärät tunneittain. Tämän raportin tarkoitus on siis kertoa käyttäjälle kyseisen keräysmoduulin suorituskyvystä.

Mikäli keräysmoduulin suorituskyvyssä havaitaan poikkeama normaalitasosta, on luonnollista selvittää alueella mahdollisesti vaikuttavat häiriötilanteet. Häiriöviestien tarkastelua varten löytyy myös raportti MIS ohjelmasta. Ehkä tilanteessa olikin vain kyse pienestä tilausmäärästä? Tätäkin varten löytyy raportti, josta asia voidaan tarkastaa. Raporttien tietoa yhdistäen käyttäjä saakin muodostettua varsin kattavan kuvan automaation toiminnasta. Toisaalta tämän kaltainen raportointi jättää monia asioita käyttäjän vastuulle. Käyttäjällä tulee olla tieto siitä, että mitä raportteja on saatavilla, mitä tietoa ne sisältävät ja mitkä tiedot ovat yhdistelykelpoisia. Käyttäjän täytyy myös tietää kaikki ne asiat, jotka voivat vaikuttaa tutkittavaan asiaan tai ongelmaan. Mikäli käyttäjältä puuttuu jokin näistä tiedoista, ei voida varmistua siitä, että käyttäjä on saanut kaikki tarvittavat tiedot ongelman ratkaisemiseksi. Tämän vuoksi on olemassa riski, että päätöksiä tehdään vajailla tiedoilla väärin perustein.

Toinen selkeä ongelma raportteihin perustuvassa tiedossa on, että useiden raporttien lataaminen ja tulkitseminen on työlästä. Tämä korostuu erityisesti rutiininomaisissa toistuvissa tarkastuksissa. Samojen raporttien lataaminen ja saman tiedon tarkastaminen

päivittäin, tai jopa tunneittain, kuluttaa merkittävästi käyttäjän aikaa ja aiheuttaa tarpeetonta kuormitusta työntekijälle. On myös selkeä riski siihen, että käyttäjän ollessa kiireinen toisen asian parissa, joitain tarkastuksia jää tekemättä.

Pienessä tai keskikokoisessa automaatiojärjestelmässä voidaan usein välttyä tässä työssä esitetyiltä ongelmilta, koska laitteiston määrä pysyy helposti käsiteltävänä. Työn kohteena oleva laitos on kuitenkin niin suuri, että sen hallinta perinteisillä menetelmillä ei ole tehokasta ja on altista käyttäjävirheille.

Tässä työssä tullaan etsimään ratkaisu, jolla edellä kuvatun kaltaisia toistuvia tarkistuksia voidaan jatkossa automatisoida. Työn lopputuotteena tulee olemaan ohjelmistokokonaisuus, joka esittelee menetelmät ja tekniikat, joilla tavoite saavutetaan. Tätä ohjelmistoa soveltaen tilaajaorganisaatiossa voidaan jatkossa luoda automatisoituja tarkastuksia ja tällöin käyttäjälle voidaan esittää vain valmiiksi yhdisteltyä ja analysoitua tietoa. Ohjelmiston kohderyhmänä on tilaajan organisaatiossa kehitys- ja asiantuntijatehtävissä toimivat henkilöt. Työssä ei keskitytä itse tarkastuksiin tai niiden merkitykseen järjestelmän toiminnan kannalta.

3 Tekniset rajoitukset ja valmiit ratkaisut

Työn toteutuksen suunnittelussa täytyy kiinnittää alusta pitäen huomiota tilaajan asettamiin tietoturvallisiin rajoitteisiin. Laitoksen tuotannon vaarantumisen estämiseksi on määrätty seuraavaa:

- Ohjelmistoilla ei ole suoraa pääsyä tuotannon tietokantaan.
- Kaikki käytettävät ohjelmistot täytyy olla avoimen lähdekoodin ohjelmistoja tai itse ohjelmoituja ohjelmistoja.
- Laitoksen tietoverkosta ei ole pääsyä laitoksen ulkopuolelle.

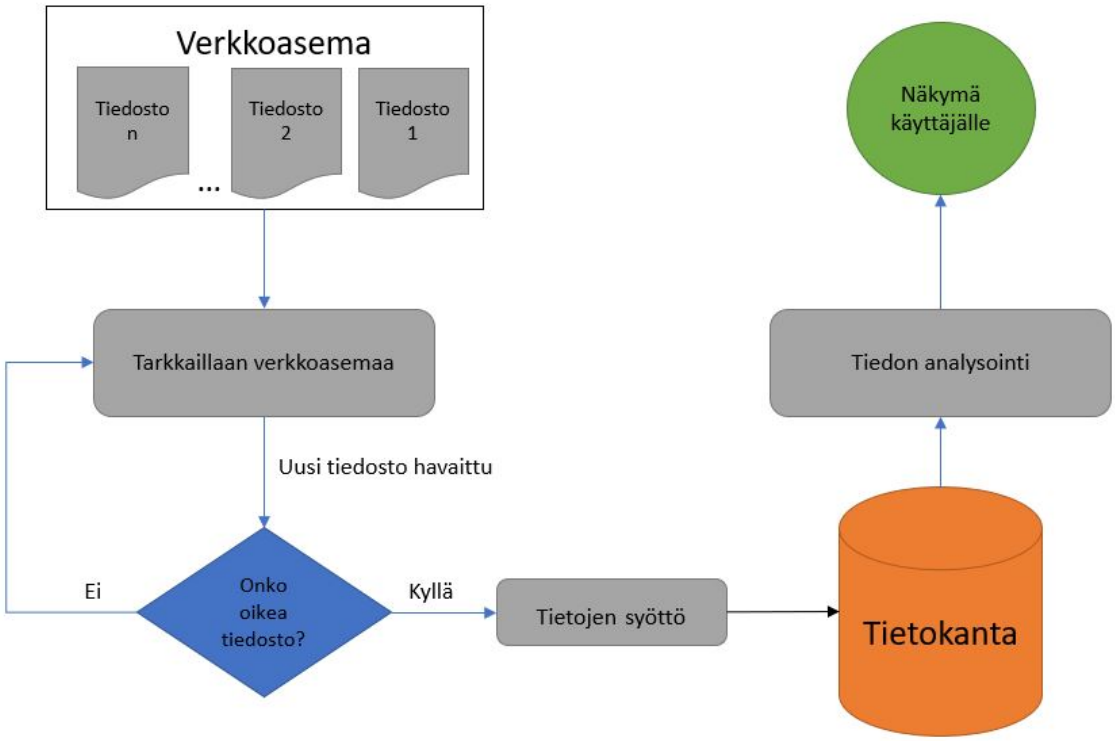
Automaattisen tiedon analysoinnin ja esittämisen ehtona on, että tieto on saatavilla sellaisessa muodossa, että sitä voidaan ohjelmallisesti lukea ja käsitellä. Koska tiedon kysely suoraan tietokannasta on turvallisuus syistä estetty, täytyy tieto kerätä valmiista raporteista. MIS sisältää toiminnon, joka mahdollistaa käyttäjän lataaman raportin tallentamisen kiintolevylle. Raportit tallentuvat sqlite-tiedostomuodossa. Raportteja on myös mahdollisuus ajastaa tallennettavaksi haluttuna aikana tai jatkuvasti tietyin väliajoin. Käytettäessä automaattista raportin tallennusta raportti ei tule näkyviin MIS-ohjelmassa, vaan se tallennetaan asetuksissa määritettyyn kansioon. Tämä on ainoa automaattinen tapa tallentaa tietoa automaatiojärjestelmästä.

Tiedon keräämiseen käytetyn ohjelman tulee siis pystyä havaitsemaan tallennettu tiedosto kansiossa ja osattava poimia raportista halutut tiedot analyysia varten. Analysointiin käytetyt menetelmät ovat useimmiten tiedon vertailua aiempaan tietoon sekä yksinkertaisia matemaattisia operaatioita, kuten keskiarvolaskentaa. Analysoinnin jälkeinen lopputulos tulisi olla sellaista tietoa, jolla on merkitystä käyttäjälle. Jotta työn tavoite käyttäjän kuormituksen vähentämisestä täyttyy, tulee analysoinnilla saada merkittävästi vähennettyä epäoleellista tietoa, eli esittää vain tietoa, joka vaatii käyttäjältä jatkoselvitystä tai muita toimenpiteitä. Tilaajalla on laitoksen sisäisessä verkossa toimiva web-selaimella käytettävä ohjelma, jota käytetään organisaatiossa raportoinnin työkaluna. Tilaajan toiveena on, että tätä olemassa olevaa ohjelmaa hyödynnettäisiin tiedon esittämiseen.

Ottaen huomioon edelle kerrotut ehdot, toiminnallisuudet ja toiveet suoritettiin etsintää valmiille ratkaisulle. Hyvin nopeasti kävi ilmi, että juuri tähän käyttöön soveltuvaa valmista ohjelmistoa ei ole saatavilla. Avoimen lähdekoodin vaatimus karsi kaikki kaupalliset analytiikka työkalut pois, toisaalta valtaosa ohjelmista myös tukeutuu hyvin vahvasti pilvipalveluiden hyödyntämiseen, joka myös on mahdotonta koska verkosta ei ole pääsyä ulkomaailmaan. Pienempiä avoimen lähdekoodin projekteja yhdistelemällä olisi ollut mahdollista saada ainakin osittain toteutettua tavoitteita, mutta sen toteuttamiseen arvioitu aika olisi ollut yhtä suuri tai suurempi kuin kokonaan oman ohjelman tekemiseen käytetty aika. Näillä perusteilla päädyttiin valmistamaan käytettävä ohjelmisto itse.

4 Ohjelmiston suunnittelu

Toteutettavan ohjelmiston toiminta perustuu kahteen erilliseen ohjelmaan sekä kaiken keskellä olevaan tietokantaan. MIS-ohjelmassa asetetaan automaattinen raportin tallennus päälle niin, että halutut tiedot sisältävä raportti tallennetaan halutuin aikavälein verkkoasemalla sijaitsevaan kansioon. Luodaan ohjelma, joka on jatkuvasti käynnissä ja tarkkailee kyseistä kansiota. Kun ohjelma havaitsee kansiossa uuden tiedoston, tarkistetaan tiedoston tyyppi. Mikäli tyyppi on oikea, tiedoston sisältämät tiedot syötetään perustettavaan tietokantaan. Tiedon analysointia varten tehdään ohjelma, johon voidaan ohjelmoida tiedon analysointia varten tarvittavat algoritmit. Tiedon esittämistä varten tehdään tilaajalla jo käytössä olevaan internet selaimella toimivaan ohjelmistoon uusi osio, jossa analysoitu tieto on selkeästi esillä. Tiedon analysointi tehdään aina kun käyttäjän näkymä ladataan uudelleen. Ohjelman toiminta on esitelty kuvassa 1. Valmiissa selainohjelmistossa on käytössä Apache http-palvelinohjelmisto [3], sekä Codeigniter [4] ohjelmistokehys, jota tullaan käyttämään myös tiedon esittämisessä.



Kuva 1. Ohjelmiston toiminta.

4.1 Esimerkkiautomatisoinnin esittely

Ohjelmiston toiminta varmistetaan tekemällä esimerkkiautomatisointi olemassa olevasta työtehtävästä. Tässä työssä käytetään esimerkkinä laitteiden häiriötiheyden muutoksen tarkkailua. Esimerkki sopii erinomaisesti automatisoitavaksi, koska se on jatkuvasti tehtävä seuranta, jonka tulos on riippuvainen käyttäjän tiedoista ja taidoista. Seurannan tarkoituksena on havaita kulumisesta tai vaurioitumisesta aiheutuvat ongelmat laitteissa, jotta niiden kunto voidaan arvioida ja mahdolliset ongelmat korjata, ennen kuin niistä muodostuu tuotantokatkoja. Tätä tarkkailua varten tarvittavia tietoja ovat laitteiden tunnukset ja tiedot häiriöistä. Häiriöllä tarkoitetaan tämän työn yhteydessä tilaa, jossa laite on pysähtynyt jonkin poikkeavan tilanteen vuoksi. Toimilaitteita ohjaavat PLC:t on ohjelmoitu niin, että mikäli laitteen toiminta poikkeaa odotetusta, pysäytetään laite ja muodostetaan häiriöviesti. Viestin sisältö määritetään laitteen tilan ja antureiden tilatietojen avulla. PLC välittää viestin WMS-järjestelmälle ja se taltioidaan tietokantaan. Häiriöviesti sisältää muun muassa laitteen tunnuksen, häiriön kategorisen tyypin, selitetekstin sekä aikaleiman. MIS-ohjelmasta löytyy raportti häiriöviestien tarkastelua varten.

Tässä esimerkissä analysointi suoritetaan vertaamalla laitteen häiriömääriä normaalitasoon. Mikäli havaitaan että häiriömäärät ovat nousussa, välitetään tieto käyttäjälle. Vertailu tehdään käyttäen hetkellistä häiriökeskiarvoa ja pitkänajan häiriökeskiarvoa. Keskiarvojen suhteellinen muutos lasketaan jakamalla hetkellinen keskiavo pitkänajan keskiarvolla. Esimerkki tapauksessa käyttäjälle halutaan esittää selkeä listaus laitteista, joiden vikaantumistiheydessä on havaittu sellainen poikkeama, että käyttäjän on selvitettävä mistä poikkeama johtuu.

4.2 Tietokanta

Tiedon tallennuksen ja esittämisen keskiössä on tietokanta. Tietokannat voidaan jakaa karkeasti kahteen pääkategoriaan: SQL- ja noSQL-tietokannat. SQL-tietokannalla tarkoitetaan relaatiotietokantaa, jonka sisältämät tiedot on tallennettu tauluihin. Taulujen välille muodostetaan yhteyksiä ja niissä olevaa tietoa voidaan esittää joustavasti. Tietokannasta haetaan tietoa muodostamalla SQL-lausekkeita, jotka suoritetaan tietokanta-

ohjelmistossa. SQL-tietokanta tarjoaa erittäin hyvän suorituskyvyn, mikäli tietoa on paljon ja se on hyvin homogeenistä. NoSQL tietokannalla tarkoitetaan yleisesti kaikkia muita kuin SQL tietokantoja. Yleisin noSQL tietokantatyyppi on tiedostotietokanta. Tiedostotietokannassa tieto on tallennettu tiedostoihin, avain-arvopareina. Tiedostotietokannat ovat tehokkaampia käsittelemään tietoa, jonka sarakkeiden määrä ja tietomuoto vaihtuu usein [5, s.12]. Koska tässä työssä tietokantaan tallennetaan ainoastaan automaatiojärjestelmän tuottamaa homogeenista tietoa, sopivin tietokanta on SQL relaatiotietokanta. Tietokantaohjelmistoksi valittiin avoimen lähdekoodin kriteerit täyttävä MariaDb-ohjelmisto [6].

Työssä käytettävää esimerkkitapausta varten tarvitaan perustettavaan tietokantaan kaksi taulua. Ensimmäiseen tauluun tallennetaan tiedot kaikista järjestelmän laitteista. Laitteista tarvitaan vähintään tunnukset, eli GC- ja LAC-tunnisteet, sekä nimet, jotta niiden tunnistaminen on käyttäjälle helpompaa. Toiseen tietokantatauluun tallennetaan häiriöviestit. Häiriöviestien sisällöstä tarvitaan vähintään aikaleimat, laitetunnukset sekä häiriön selitteet.

4.3 Ohjelmisto

Tiedon tallennukseen käytetty ohjelma tehdään käyttäen Python [7] -ohjelmointikieltä. Pythonin normaalikirjastoja laajennetaan ohjelmaa varten kahdella lisäkirjastolla. Watchdog-kirjasto sisältää kansion tarkkailuun tarvittavat työkalut ja Mysql.connector-kirjasto sisältää tietokantayhteyttä varten tarvittavat toiminnot.

Tiedon esittämiseksi valmiina käytössä oleva codeigniter-ohjelmisto on PHP-ohjelmointikieltä [8] käyttävä kevyt ja nopea sovelluskehys. Se on tarkoitettu käytettäväksi web-sovelluksen pohjana, tarjoamalla kehittäjän käyttöön valmiita luokkia, jotka toteuttavat yleisimpiä sovelluksissa käytettyjä toimintoja. Tällaisia toimintoja web-sovelluksissa on esimerkiksi tietokantaoperaatiot, istunnonhallinta, liikenteen reititys sekä päivämäärien ja kellonaikojen muunnokset. Codeigniter on suunniteltu erityisesti käytettäväksi MVC-arkkitehtuuria toteuttavien ohjelmistojen alustana.

MVC-arkkitehtuuri on ohjelmistoarkkitehtuuri, jonka ajatuksena on eriyttää ohjelmiston logiikka ja käyttöliittymä toisistaan. MVC-lyhenne tulee sanoista *model*, *view* ja *controller*.

Suomeksi puhutaan yleensä mallista, näkymästä ja ohjaimesta. Yksinkertaisessa MVC-arkkitehtuuria toteuttavassa ohjelmassa koodi on jaettu kolmeen tiedostoon. Malli tiedosto sisältää kaiken ohjelmiston tiedonkäsittelyn. Näkymä tiedosto sisältää ohjelman ulkoasun määrittelyn. Ohjain tiedosto sisältää koodin, jolla reagoidaan käyttäjän käskyihin ja muutetaan näkymää ja mallia ohjelman suorituksen edetessä. MVC-arkkitehtuurin etuina on, että sillä voidaan vähentää riippuvuussuhteita elementtien välillä. Malli ei ole riippuvainen näkymästä. Samaan malliin voidaan tehdä lukuisia erilaisia näkymiä. Ohjaimeen tai näkymään tehdyt muutokset eivät vaikuta mallin toimintaan.

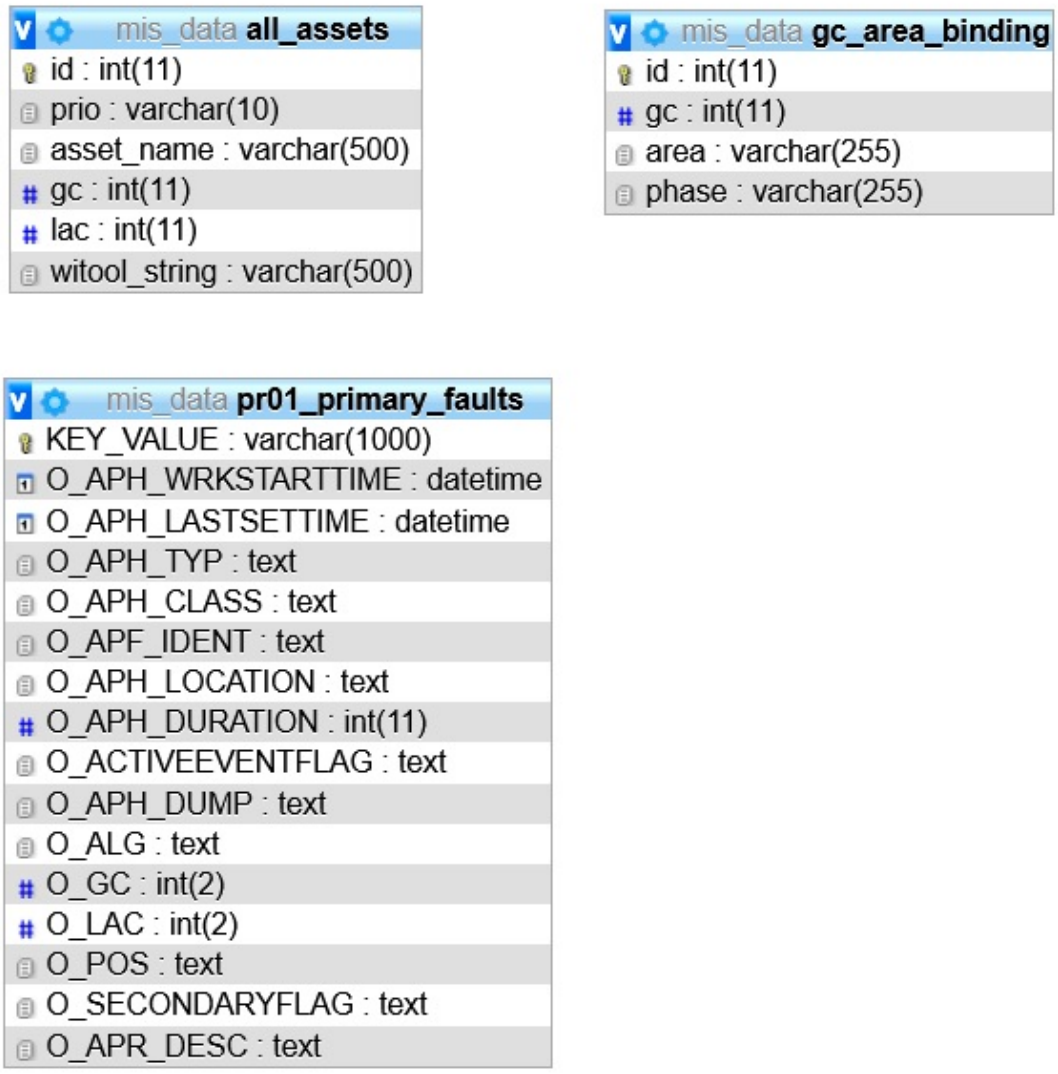
MVC-arkkitehtuuri soveltuu tähän työhön erinomaisesti. Tiedon analysointiin käytettävät mallit luodaan niin, että ne sisältävät kaiken analysointi logiikan. Näin ollen tulevaisuudessa uusia analysointeja voidaan lisätä ohjelmaan tekemällä uusia malleja.

5 Ohjelmiston toteutus

Työn tekeminen on jaettu kahteen osaan. Ensimmäisessä osassa perustetaan tietokanta ja tehdään tiedon tallentamiseen käytetty ohjelma valmiiksi. Kun ensimmäinen osio on valmis, siirrytään tekemään tiedon esittämiseen ja analysointiin tarvittavaa ohjelmistoa. Työn tilaaja määritteli projektin käyttöön jo valmiiksi olemassa olevan HP Z240-palvelintietokoneen. Palvelimessa suoritetaan myös yrityksen muita ohjelmistoja, mutta siinä on vielä hyödyntämätöntä kapasiteettia, joka voidaan luovuttaa tämän projektin käyttöön.

Työn toteuttamisen ensimmäinen vaihe on tietokanta ohjelmiston asennus ja tietokannan määrittely. MariaDb-tietokantaohjelmisto asennettiin valmistajan laatimien asennusohjeiden mukaisesti [2]. Suunnitelmasta poiketen tietokantaan päätettiin lisätä toteutusvaiheessa vielä kolmas taulu. Häiriöviestien ja laitelistauksen lisäksi haluttiin mukaan myös tieto alueiden nimistä. Kolmannessa taulussa on määritelty jokaisen GC:n vaikutusalue laitoksessa sanallisesti. Lopullinen taulukaavio on esitelty kuvassa 2. Esimerkiksi GC numero 51:n alue on ”DRY” ja tarkempi alue on ”Phase 1”. Nämä merkinnät vastaavat laitoksessa työskentelevien henkilöiden käyttämiä ilmaisuja ja tuovat siten helpotusta laitteen identifiointiin. Häiriöviestit sisältävään tauluun päätettiin tehdä sarakkeet kaikille niille tiedoille, jotka alkuperäisessä raportissa on. Tämä tehtiin sen vuoksi, että jos jotain tietoja tarvitaan tulevaisuudessa, on se jo valmiina tietokannassa. Kuvassa 2 on esitelty

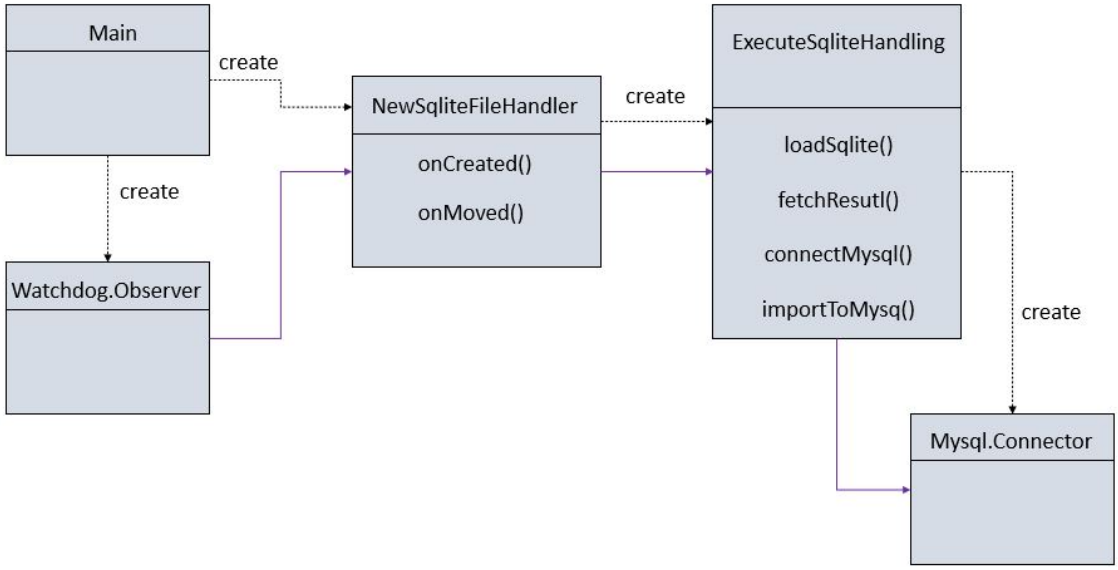
tietokantaan luodut taulut kenttineen. Toinen muutos, joka tietokantaan tehtiin toteutusvaiheessa, oli indeksointien lisääminen eniten käytetyille sarakkeille häiriöviestitaulussa. Indeksointi vaikuttaa SQL tietokannassa niin, että tietokanta tekee kopion indeksoiduista sarakkeista ja järjestää sarakkeen tiedot. Indeksoinnilla saadaan tietokantakyselyistä nopeampia, silloin kun taulussa on suurimäärä rivejä. Indeksoinnit lisättiin pr01_primary_faults taulun sarakkeisiin O_APH_LASTSETTIME, O_GC ja O_LAC.



Kuva 2. Tietokannan taulukaavio.

5.1 Tiedon tallennus

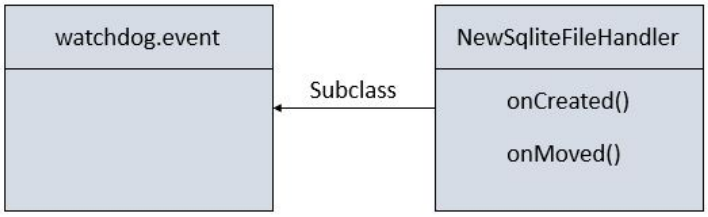
Tiedon tallennus tietokantaan perustuu suunnitelman mukaisesti MIS-järjestelmän automaattisesti tallentaviin raportteihin. Raportit tallentuvat sqlite-tiedostomuodossa. Sqlite-tiedostot ovat sqlite-tietokannanhallintajärjestelmän tuottamia tietokantoja. Jokainen kyseisessä muodossa tallennettu raportti on siis oma tietokantansa. Python-ohjelmointikielen standardikirjastojen mukana toimitetaan myös sqlite-kirjasto, jonka avulla tiedot voidaan helposti lukea. Toteutettu ohjelma on jaettu kahteen eri luokkaan ja niiden ympärillä toimiviin kirjastoihin. Watchdog-kirjastoa käytetään tiedostojen havaitsemiseen, sqlite-kirjastoa tiedon lukemiseen ja lopuksi mysql-kirjastoa tiedon tallentamiseen. Näiden kirjastojen keskellä on luokat NewSqliteFileHandler ja ExecuteSqliteHandling, jotka käsittelevät tietoa ja ohjaavat ohjelman kulkua.



Kuva 3. Tiedon tallentamiseen tehdyn ohjelman luokkakaavio.

Uusien tiedostojen havainnointiin kansiossa on käytetty watchdog-kirjastoa. Watchdog-kirjaston toiminta perustuu kahteen perusluokkaan, observers ja events. Observers-luokka havaitsee muutokset tiedostojärjestelmässä ja events-luokka reagoi muutoksiin. Observer-luokan ilmentymät ja event ilmentymät ajetaan eri säikeissä samanaikaisesti. Observer pystyy siis käynnistämään useita event ilmentymiä, vaikka edellisten ajo olisi vielä kesken. Ohjelmalla voidaankin käsitellä samaan aikaan useita tiedostoja. Kuvan 4

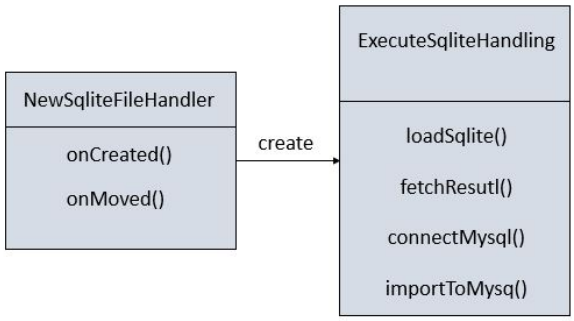
luokkakaavion mukaisesti event luokkaa käytetään oman tapahtumakäsittelijän yliluokkana. Aliluokassa tarkennetaan funktioina ne tiedostojärjestelmän muutokset, joihin halutaan reagoida. Tässä työssä on käytössä event luokan funktiot onCreated(), jolla havaitaan uudet tiedostot ja onMoved(), jolla havaitaan muutokset tiedostoissa.



Kuva 4. Tapahtumakäsittelijän ja watchdog.event luokan suhde

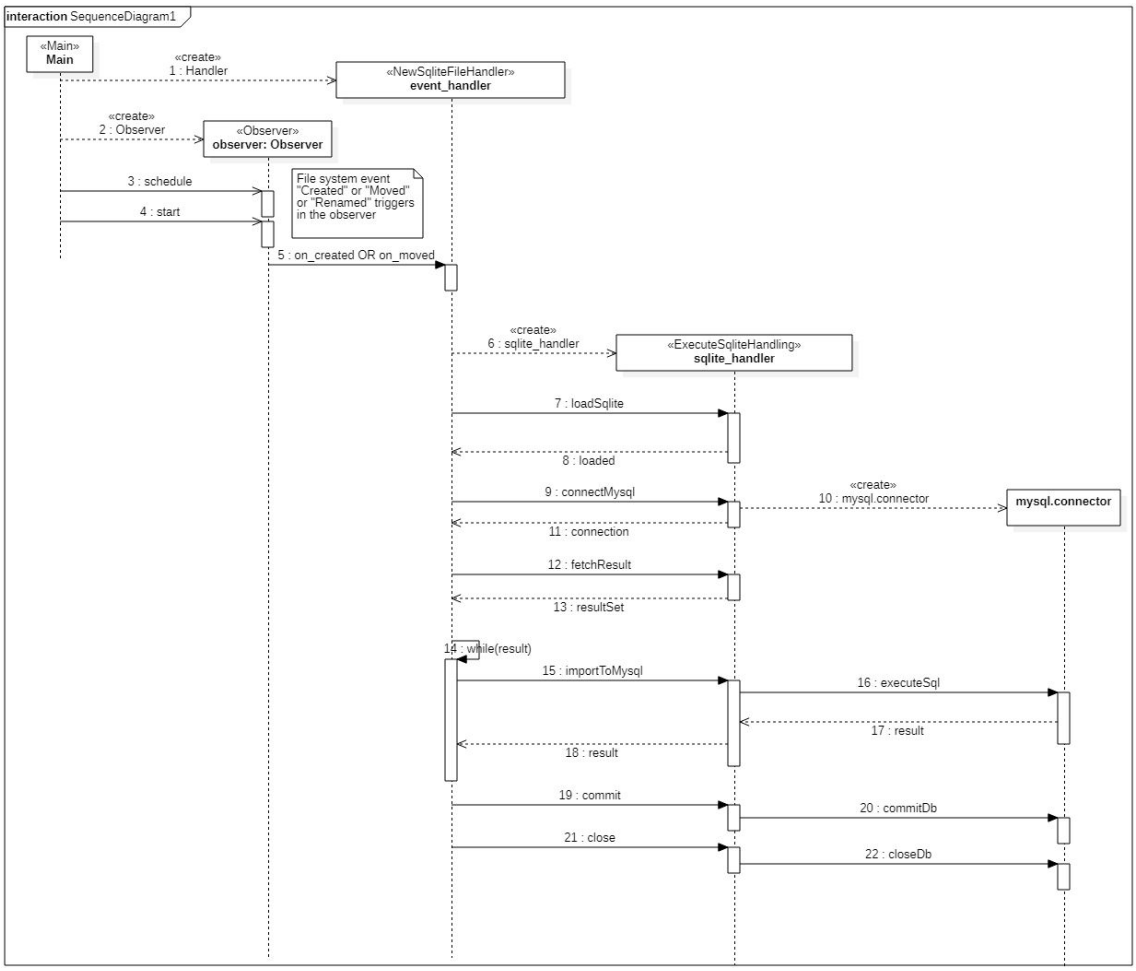
Kun observer havaitsee muutoksen tiedostojärjestelmässä, se kutsuu onCreated()- tai onMoved()-funktioita, sen mukaan onko kyseessä uusi tiedosto vai muutos olemassa olevaan tiedostoon. Testauksessa havaittiin, että WMS-järjestelmän raporttien tallennus toimii niin, että tallennuksen alussa kansioon luodaan tilapäinen tmp-päätteellinen tiedosto ja kun tietojen tallennus on valmis, tiedosto nimetään db-päätteiseksi. Tämän käyttäytymisen vuoksi onCreated()-funktiota ei voida suoraan hyödyntää tiedostojen käsittelyyn, koska se käynnistyy heti kun tmp tiedosto ilmestyy kansioon. Kun tmp tiedosto nimetään uudelleen, se ei ole enää uusi tiedosto, jolloin onCreated() ei käynnisty enää uudestaan. OnMoved() funktiota kutsutaan paitsi silloin kun tiedosto siirretään, myös silloin kun tiedostoa muokataan tai se nimetään uudestaan. OnCreated() funktio on olemassa ohjelmassa manuaalista tiedostojen syöttöä ja testausta varten. OnMoved() ja onCreated() funktioiden sisältämä logiikka on identtistä.

Tiedostojen ja tietokannan operaatioiden suorittamista varten ohjelmassa on luokka ExecuteSqliteHandling. Luokan kaavio on kuvassa 5. Luokka sisältää funktiot sqlite tiedoston avaamiseen, tiedon noutamiseen tiedostosta sekä tietokantayhteyden luomiseen ja tiedon tallentamiseen.



Kuva 5. ExecuteSqliteHandling-luokka käsittelee raportin tiedot.

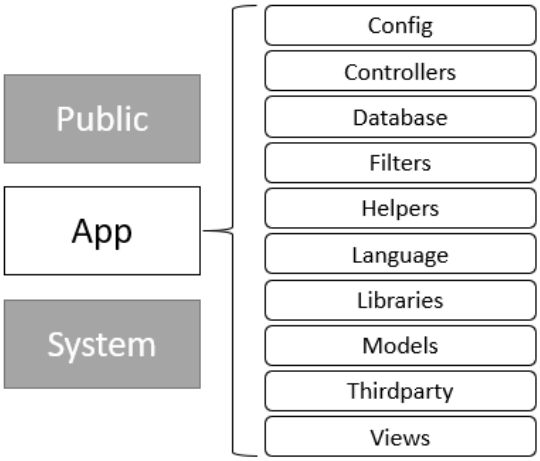
Ohjelman toiminta on esitetty tarkemmin sekvenssikaaviona kuvassa 6. Ohjelman suorituksen alkaessa luodaan ensin ilmentymä tiedostonkäsittelijästä, jonka jälkeen Observer-luokalle annetaan ilmentymää luotaessa tiedostonkäsittelijä parametrina. Näin saadaan kohdistettua Observer-ilmentymän kutsut oikealle tiedostonkäsittelijälle. Kun uusi tai muokattu tiedosto havaitaan, luodaan ilmentymä ExecuteSqliteHandling-luokasta. Ilmentymä käsittelee raportin tiedot ja syöttää ne tietokantaan käyttäen Mysql.connector-luokkaa. Tiedoston lopussa yhteys tietokantaan katkaistaan ja jäädään odottamaan uutta tiedostoa.



Kuva 6. Tiedon tallennuksen sekvenssikaavio.

5.2 Tiedon analysointi ja esittäminen

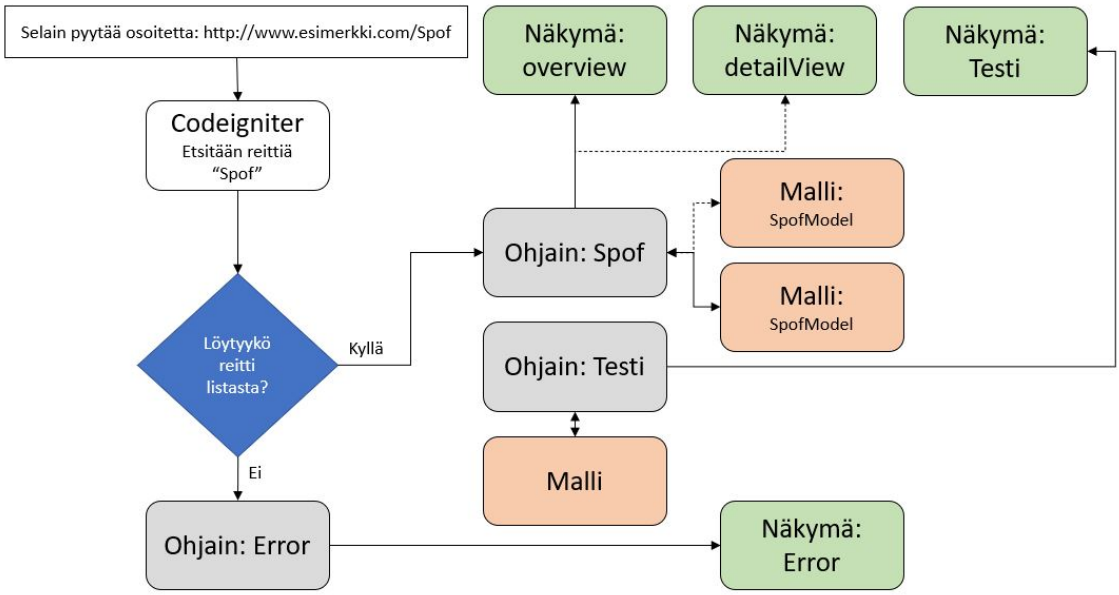
Codeigniter-ohjelmistokehys muodostuu kokoelmasta php-tiedostoja. Tiedostot on jaettu kolmeen päähakemistoon. Public-hakemisto sisältää julkiset tiedostot ja materiaalit, App-hakemiston sisälle luodaan varsinainen ohjelmisto ja System-hakemisto sisältää codeigniterin sisäänrakennetut luokat ja toimintalogiikan. App-hakemiston sisällä on omaa ohjelmaa varten valmiiksi luotu hakemistorakenne (taulukko 1).



Taulukko 1. Codeigniter-ohjelmistokehyksen hakemistorakenne.

Codeigniterin käyttöönotto on erittäin yksinkertaista. Valmistajan web-sivuilta (4) ladatut tiedostot kopioidaan haluttuun kansioon palvelimella ja käytetty http-palvelinohjelmisto asetetaan osoittamaan Public-hakemistoon. Tämän jälkeen Config-hakemistoon määritellään ohjelmiston asetukset, kuten käytettävän tietokannan osoite, käyttäjätunnukset, reitit tiedostoihin jne., jonka jälkeen asennus on valmis. Controllers-hakemistoon sijoitetaan kaikki MVC-arkkitehtuurin mukaiset ohjain tiedostot, Models-hakemistoon malli tiedostot ja Views-hakemistoon kaikki näkymä tiedostot. Käyttöönoton helppous ja nopeus ovatkin yksi selkeistä eduista, jotka kannustavat sen käyttöön. Yksinkertaisella asennuksella saadaan käyttöön selkeä, MVC-arkkitehtuuriin nojautuva alusta, joka sisältää monia valmiita luokkia yleisimpiin web-sovelluksien operaatioihin. Codeigniterin idea onkin nopeuttaa ja yksinkertaistaa web-sovellusten kehitystä. Tähän työhön kehitysalusta on erittäin sopiva, koska MVC-arkkitehtuuri mahdollistaa modulaarisuuden ja helpon laajennettavuuden, joka on koko työn perusta. Lisäksi luotavassa ohjelmistossa käytetään erittäin paljon tietokantakyselyitä, joiden luomista valmiit kirjastot tehostavat huomattavasti.

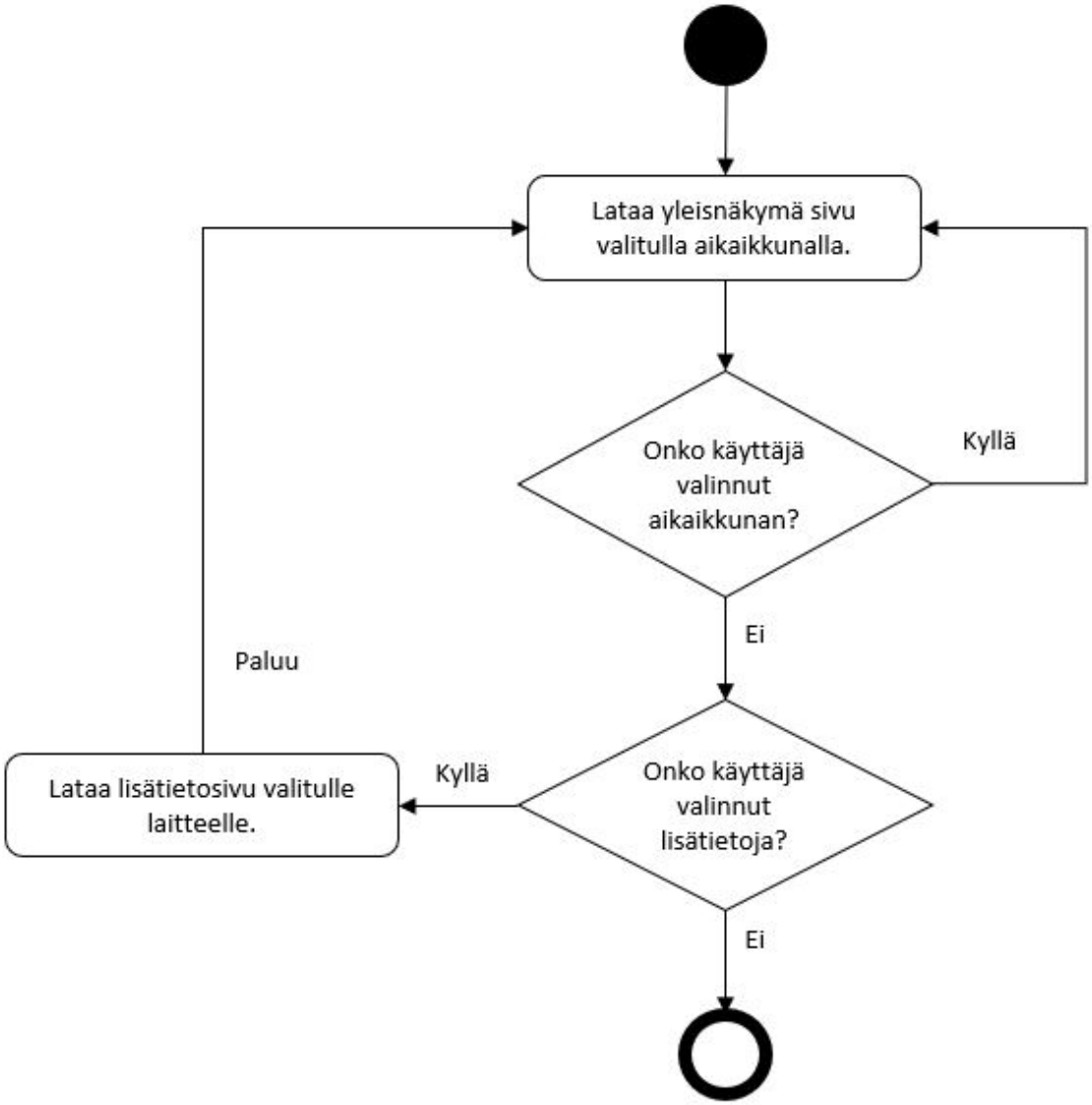
Codeigniter-ohjelmistokehyksellä toteutetut sovellukset toimivat niin, että selaimen pyytäessä sivua näytettäväksi, codeigniter tulkkaa pyynnöstä halutun sivun, reititys taulukon mukaan. Jos reitti löytyy, kutsutaan taulukossa määritettyä ohjainta, joka jatkaa ohjelman suoritusta. Mikäli reittiä ei löydy, esimerkiksi osoitteen kirjoitusvirheen vuoksi, kutsutaan error-ohjainta, joka näyttää käyttäjälle virheviestin. Codeigniter-ohjelman toimintaperiaate on esitelty kuvassa 7.



Kuva 7. Codeigniter-ohjelman toimintaperiaate.

Tätä työtä varten tehtiin reitit osoitteesta http://www.xxxxx.xxx/spof funktioon Spof.overview() sekä osoitteesta http://www.xxxxx.xxx/spof/details funktioon Spof.details(). Ensimmäinen sivu, yleisnäkymä, näyttää listana laitteet, joiden tämänhetkinen vikaantumiskeskiarvo ylittää merkittävästi pitkänajan keskiarvon. Keskiarvojen laskenta tehdään palvelimella aina kun käyttäjä lataa kyseisen sivun. Jokaisen listalla olevan laitteen kohdalla

esitetään käyttäjälle painike, jota painamalla käyttäjä voi avata laitekohtaisen lisätietosivun. Kuvassa 8 on tiedon esittämisen vuokaavio.



Kuva 8. Tiedon esittämisen toimintalogiikka.

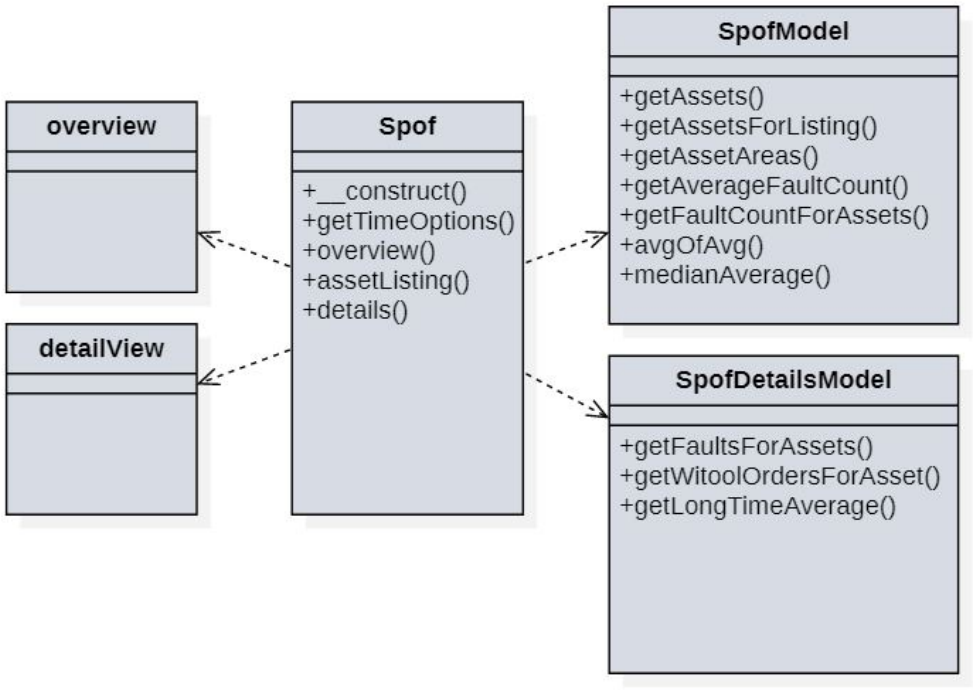
Lisätietosivua ei ollut ollenkaan työn alkuperäisessä suunnitelmassa vaan idea siitä syntyi kehityksen aikana. Lisätietosivulla näytetään yhden laitteen kaikki häiriöviestit käytettävältä aikaikkunalta, häiriöiden keskiarvomäärät pylväsdiagrammina sekä tilaajan käyttämään kunnossapitojärjestelmään tallennetut menneet ja tulevat työmääräykset lait-

teelle. Lisätietosivulta saatava hyöty koettiin niin suureksi, että se lisättiin työhön suunnittelun jälkeen. Toinen kehityksen aikainen lisäys oli aikaikkuna asetuksen valinta yleisnäkymäsivulla. Aikaikkunalla tarkoitetaan arvoa, jota käytetään tiedon analysoinnissa keskiarvolaskentaan. Sovelluksessa käytetään vain yhtä ohjain tiedostoa. Tämä sama tiedosto sisältää sekä yleisnäkymän, että lisätietonäkymän ohjauksen.

Mallit

Codeigniter ohjelmistokehykseen luodaan uusi malli hyvin yksinkertaisesti. Uusi malliluokka luodaan Codeigniter\Model-luokan aliluokaksi. Näin toimiessa saadaan mallissa käyttöön codeigniter:n toiminnallisuudet. Codeigniter sisältää paljon ohjelmointia helpottavia apuluokkia. Yksi näistä luokista on Database-luokka. Se mahdollistaa todella yksinkertaiset kyselyt tietokannasta. Tietokannan asetukset määritellään valmiiksi konfigurointitiedostoon, jolloin jo malliluokan olion luonnissa muodostetaan tietokantayhteys automaattisesti. Tämän jälkeen tiedon kysely ja syöttäminen onnistuu muutamalla komennolla.

Ohjelmaa varten luotiin kaksi eri malli luokkaa. Ohjelman olisi voinut tehdä myös yhdellä mallilla, mutta selkeyttämisen vuoksi lisätietosivun ja yleisnäkymän toiminnoista tehtiin omat malli luokkansa. SpofModel-malli sisältää funktiot keskiarvojen laskentaa varten. Tämän lisäksi mallissa on määritelty funktio, joka palauttaa tietokannasta listan kaikista olemassa olevista laitteista ja niiden alueista järjestelmässä. Alueet ja laitteet yhdistetään gc numerolla niin, että jokaiselle gc numerolle on määritelty tasan yksi alue. SpofDetailsModel-malli sisältää vain kolme funktiota. GetFaultsForAsset() palauttaa listan vikaviesteistä laitekohtaisesti halutulta aikaväliltä ja getLongTimeAvg palauttaa yksinkertaistetun 21 päivän häiriökeskiarvon. GetWitoolOrdersForAsset()-funktion yhdistää samalla palvelimella olevaan toiseen tietokantaan, jossa sijaitsee kaikki tilaajan käyttämän ennakkohuoltojärjestelmän työmääräykset. Funktio kyselee tietokannasta kaikki työt kyseiselle laitteelle ja palauttaa listan kutsujalle. Tätä funktiota varten lisättiin tietokannan tauluun all_assets, sarake witoolSearchString. Tämä oli tehtävä, koska ennakkohuoltojärjestelmässä laitteet on identifioitu hieman eri tavalla. Mallien, ohjaimen ja näkymien välinen suhde on esitelty luokkakaaviossa kuvassa 9.



Kuva 9. MVC-arkkitehtuuriin perustuva luokkakaavio.

Analysointi

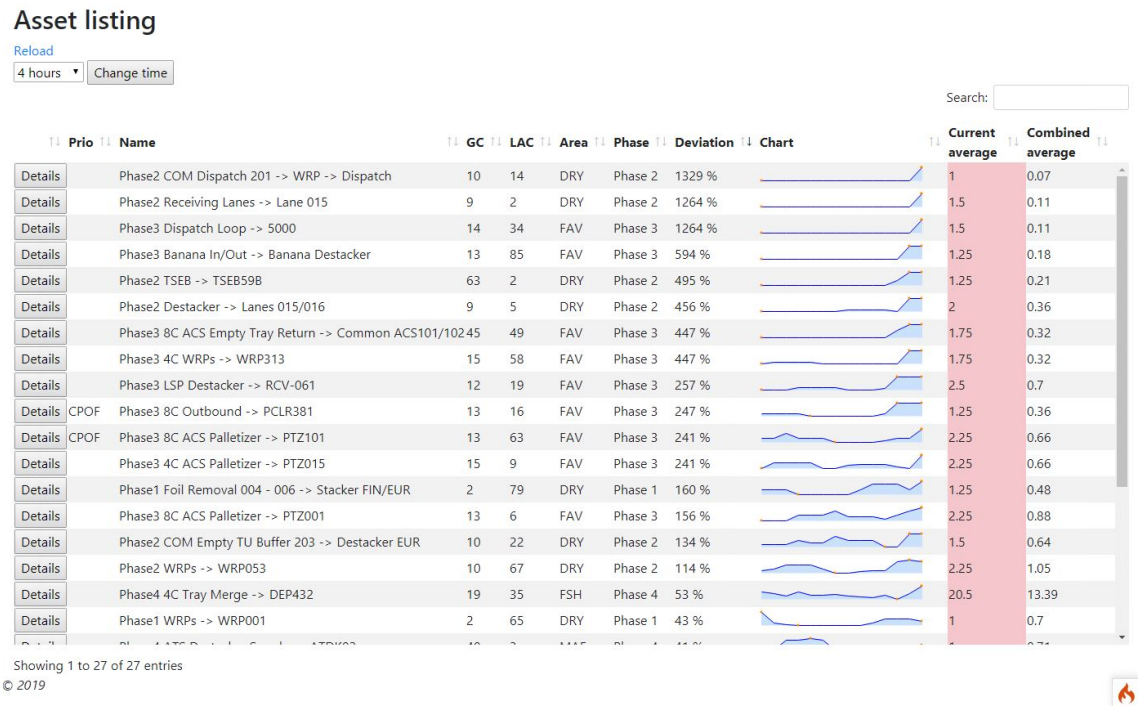
Esimerkkitapauksen tiedon analysointi suoritetaan niin, että laitteelle lasketaan ensin lyhyen ajan häiriökeskiarvo kaavalla: häiriöviestien määrä / aikaikkunan arvo. Lopputulos on muodossa häiriötä / tunti. Vertailuun käytetty pitkänajan keskiarvo lasketaan laskemalla 14 kertaa lyhyen ajan keskiarvo, niin että joka laskennalla siirretään aikaikkunaa yksi tunti taaksepäin. Lopuksi näiden 14 keskiarvon keskiarvo määrittää pitkänajan keskiarvon ja siten analyysissä käytetyn vertailuarvon. Testeissä todettiin, että käyttämällä keskiarvojen keskiarvoa, yksinkertaisen pitkänajan keskiarvon sijasta, saadaan pienennettyä yksittäisten häiriösarjojen vaikutusta vertailuarvoon.

Ohjelman ensimmäisessä versiossa, laitelistauksen hakemisen jälkeen laskettiin kaikkien laitteiden keskiarvot ja esitettiin ne käyttäjälle. Tämä aiheutti kaksi ongelmaa. Ensinnäkin lista oli aivan liian pitkä luettavaksi ja toiseksi, listan laskenta vei kohtuuttoman

pitkän ajan. Jokaista laitetta kohden piti laskea 16 keskiarvoa ja tehdä 15 tietokantakyselyä. Sivun latautumisaika saattoi olla useita minuutteja. Ratkaisin ongelman niin, että laitteelle lasketaan ensimmäisessä vaiheessa ainoastaan lyhyen ajan keskiarvo. Tätä varten tarvitsee tehdä yksi tietokantakysely ja laskea yksi keskiarvo. Mikäli laskentahetkellä laitteen vikaantumistiheys on suurempi kuin 1,0 häiriötä tunnissa, lasketaan pitkän ajan keskiarvo. Tähän ehtoon päädyttiin, koska voidaan ajatella, että tätä pienemmät vikaantumistiheydet ovat laitteen normaalia toimintaa ja ei vaadi asiaan perehtymistä. Näillä muutoksilla ratkaisin kaksi eri ongelmaa, laitelista pysyi luettavana sekä sivun lataamisaika putosi minuuteista joihinkin sekunteihin.

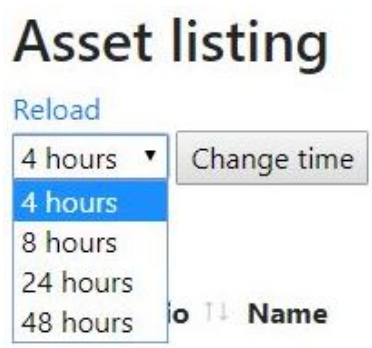
Näkymät

Ohjelman ulkoasu määritetään näkymätiedoston avulla. Yleisnäkymän asettelussa päädyin yksinkertaiseen ja selkeään esitystapaan, joka on esitelty kuvassa 10.



Kuva 10. Yleisnäkymä valmiissa ohjelmassa.

Sivun yläreunassa on otsikko sekä aikaikkunan vaihtamiseen käytetty alasvetovalikko ja hyväksymisnäppäin (kuva 11).



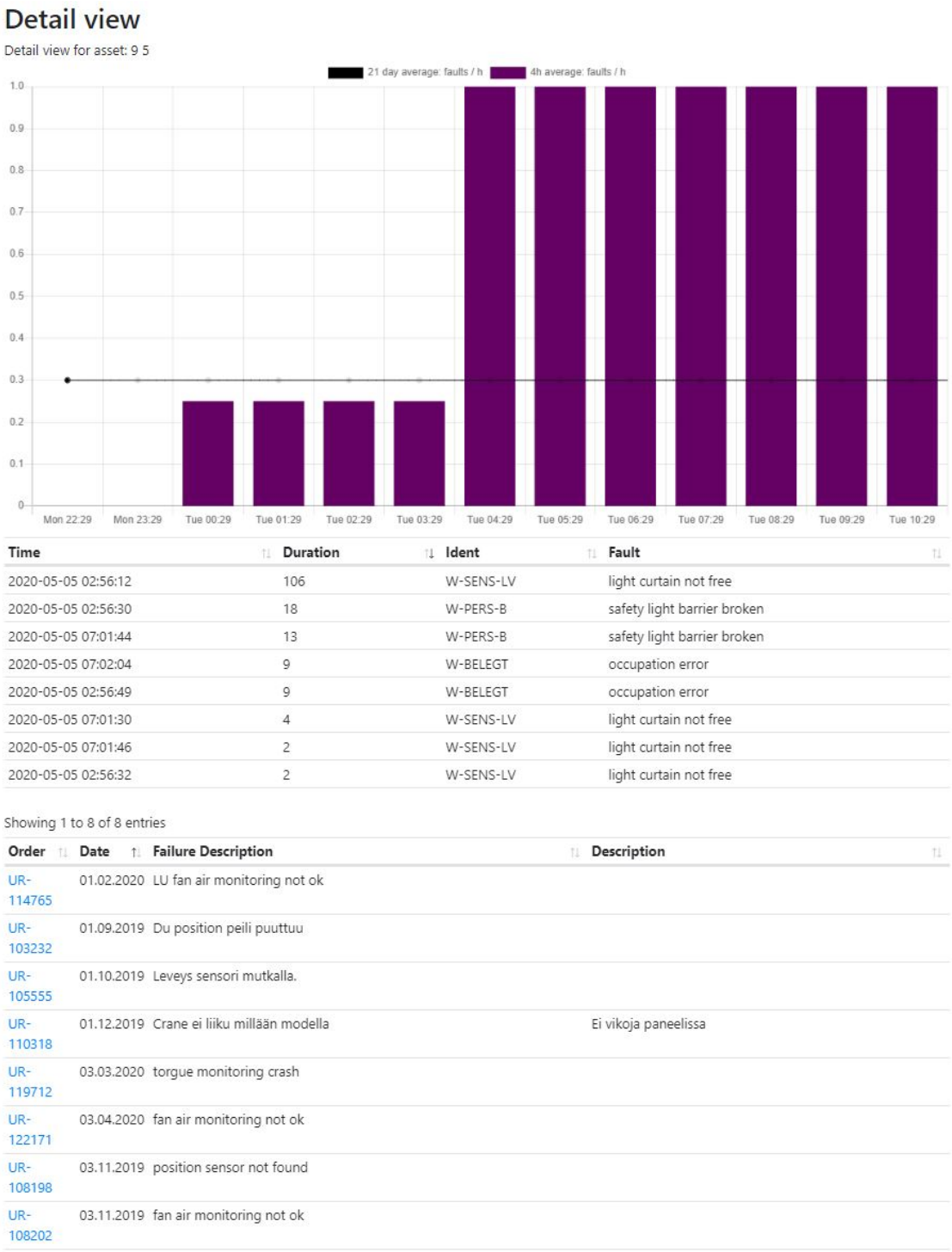
Kuva 11. Aikaikkunan valinta.

Sivulla näytetään taulukko, jossa jokainen rivi on yksittäinen laite järjestelmässä (kuva 12).

Kuva 12. Käyttöliittymän laiterivi.

Rivillä näytetään perustietoina laitteen nimi, gc, lac sekä alueet. Analyysin pohjalta näytetään tiedot keskiarvojen suhteellisesta muutoksesta sekä keskiarvojen lukuarvot. Keskiarvojen kehittymisen nopeaa havainnointia varten lisäsin riville vielä pienen viivadiagrammin, jossa esitetään keskiarvolaskennassa käytetyt 14 eri keskiarvoa. Lisäksi jokaisella rivillä on painike, josta voidaan siirtyä lisätietonäkymään.

Lisätietonäkymä (kuva 13) on jaettu kolmeen osaan. Näkymän yläreunassa näytetään pylväsdiagrammina samat tiedot, jotka esitetään yleisnäkymän pienessä diagrammissa. Tämän lisäksi diagrammissa näytetään mustalla viivalla erikseen laskettu 21 päivän häiriökeskiarvo laitteelle. Tämän viivan tarkoitus on luoda käyttäjälle perspektiiviä tulosten arviointiin. Sivun keskiosassa näytetään kaikki häiriöviestit valitun aikaikkunan ajalta ja alaosassa kaikki työmääräykset kunnossapitojärjestelmästä.



Kuva 13. Lisätietonäkymä.

Tiedon esittämisessä käytetään seuraavia JavaScript-kirjastoja:

- Datatables, helpottaa muokattavine taulukoiden tekemistä.

- Chart.js, käytetään graafien piirtämiseen.
- Jquery sparkline, käytetään pienten graafien piirtämiseen.
- Jquery, kirjasto, joka helpottaa javascript ohjelmien kehitystä.

Näiden lisäksi sivujen muotoilussa käytetään Bootstrap CSS-kirjastoa. Näkymäsivujen ulkoasu esitetään selaimessa HTML-merkkauskielellä. Ohjelma toimii niin, että varsinainen näkymä tiedosto sisältää PHP-koodia, joka generoi dynaamisesti sisällön sivulle. Näin yhtä näkymä tiedostoa voidaan käyttää mallina kaikille laitteille. Muuttujien välittäminen ohjaimesta näkymille on codeigniter:n työkalujen avulla hyvin yksinkertaista. Halutut muuttujat viedään array-tyypin muuttujaan avain-arvo pareina ja array muuttuja välitetään näkymälle. Tämän jälkeen avaimet ovat näkymän käytössä erillisinä muuttujina.

5.3 Lopputulos

Lopputuloksena syntynyt valmis ohjelmisto sisällytettiin jo olemassa olevaan ohjelmistoon. Tätä varten ohjelmiston päävalikkoon lisättiin uusi linkki, jonka kautta käyttäjät pääsevät siirtymään suoraan yleisnäkymään. Ohjelma on otettu ensin rajatun, noin 10 hengen, käyttäjäryhmän työkaluksi. Testijaksolla halutaan varmistua ohjelman riittävän vakaasta toiminnasta, sekä ennen kaikkea esimerkkinä käytetyn tarkkailun algoritmien toimivuudesta. Testiryhmältä saatu palaute kertoo siitä, että tämän kaltaisilla automatisoinneilla helpotetaan työntekijöiden työskentelyä ja ne otetaan hyvin vastaan organisaatiossa. Seuraavassa vaiheessa ohjelmiston käyttö sisällytetään organisaation sisäiseen toimintaprosessiin ja se tulee korvaamaan päällekkäiset manuaalisesti tehtävät tarkastukset. Jatkossa ohjelmistoa tullaan hyödyntämään useiden eri tarkastusten automatisointiin. Myöhemmin onkin mielekästä käynnistää kehitysprojekti, jossa luodaan ohjelman jatkoksi hälytysjärjestelmä. Hälytyksien avulla voitaisiin mahdollisesti ohjata useista eri automaattisista tarkastuksista löytyvät kriittiset tiedot suoraan käyttäjille, ilman että heidän tarvitsee avata jokaista sivua erikseen.

6 Yhteenveto

Tilaajan organisaatiossa oli jo pitkään ollut halu vähentää toistuvien tarkastus tyyppisten tehtävien suorittamista. Sen lisäksi että tehtäviin kuluu todella paljon aikaa, on toistuvien operaatioiden suorittaminen myös henkisesti haastavaa työntekijöille. Tässä työssä selvitettiin, onko työn esimerkin kaltaisten toistuvien analyysien tekemistä mahdollista automatisoida.

Tilaajan asettamien teknisten rajoitteiden vuoksi valmista ohjelmistoa, joka olisi suoraan ratkaissut tilaajan ongelman, ei ollut saatavilla. Ongelman ratkaisemiseksi suunniteltiin ja ohjelmoitiin oma ohjelmisto. Ohjelmiston jakaminen kahteen osaan, tiedon tallennukseen sekä esittämiseen ja analysointiin, oli luonteva tapa jäsentää työn kulkua. Ohjelman modulaarisuus mahdollistaa sen, että tiedon esittäminen on helppo sovittaa myös muihin alustoihin, kuin tässä käytettyyn codeigniter-ohjelmistokehykseen. Tiedon tallennukseen käytetty tietokanta on myös täysin vaihdettavissa mihin tahansa tietokantaratkaisuun. Myös tiedon tallentaminen pilvipalveluun olisi teknisesti yksinkertainen toteuttaa nykyisen ratkaisun päälle. Työn esimerkkinä käytetty automatisointi on ollut testikäytössä noin viiden kuukauden ajan. Tietokantaan on kertynyt rivejä yli 2 000 000. Haasteena työssä tuli vastaan tilanne, jossa tietokannan rivimäärät alkoivat selkeästi vaikuttaa ohjelmiston käyttöön, pitkinä latausaikoina. Ongelma ratkesi indeksoinnin lisäämisenä tietokantaan ja muokkaamalla algoritmien suoritusjärjestystä. Vaikka rivien määrä kasvaa jatkuvasti, ei ohjelman käyttö ole enää hidastunut. Käyttöönoton aikana ohjelmistosta löytyi muutamia ohjelmointivirheitä, jotka vaativat korjausta. Näiden ratkettua, on ohjelma osoittautunut erittäin luotettavaksi ja toimintahäiriöitä ei ole ilmennyt.

Uusien automatisointien tekeminen tämän mallin pohjalta on helppoa. Tiedon tallennus ohjelmaan määritellään uusi tiedostonkäsittelijä, joka syöttää uudet tiedot tietokantaan. Analyysia varten määritellään uusi malli -luokka ja haluttu tieto voidaan sen jälkeen esittää laajennuksena jo olemassa oleviin näkymiin, tai luoda uusi näkymä. Voidaankin todeta, että työlle asetetut tavoitteet täyttyivät ja lopputulos on onnistunut.

Lähteet

1 Witron references. Verkkoaineisto. Witron logistik + informatik GmbH. https://www.witron.de/en/references.html. Luettu 14.05.2020.

2 Inex lukuina. Verkkoaineisto. Inex Partners Oy. https://www.inex.fi/inex-yrityk-senae/inex-lukuina/. Luettu 02.05.2020.

3 Apache http server project. Verkkoaineisto. Apache software foundation. <https://httpd.apache.org/>. Luettu 3.10.2019.

4 Codeigniter web framework. Verkkoaineisto. Codeigniter Foundation. <https://co-deigniter.com/>. Luettu 3.10.2019.

5 Leavitt, Neal. 2010. Will NoSQL Databases Live Up to Their Promise?. Verkkoai-neisto. IEEE Computer. <http://www.leavcom.com/pdf/NoSQL.pdf>. Luettu 3.03.2020.

6 MariaDB Server, Verkkoaineisto. MariaDB Foundation. <https://mariadb.org/>. Luettu 14.06.2019.

7 Python Software. Verkkoaineisto. Python Software Foundation. <http://www.pyt-hon.org>. Luettu 3.10.2019.

8 PHP-Software. Verkkoaineisto. The PHP group. <https://www.php.net/>. Luettu 3.10.2019.

9 MariaDb installation instructions, Verkkoaineisto. MariaDB Foundation. <https://mariadb.com/kb/en/installing-mariadb-msi-packages-on-windows/>, lu-ettu 14.06.2019