OULUN AMMATTIKORKEAKOULU

Jarno Mattila

TIIMI-PORTAALIN HÄLYTYSTEN KÄSITTELY JA OHJAUS

TIIMI-PORTAALIN HÄLYTYSTEN KÄSITTELY JA OHJAUS

Jarno Mattila
Opinnäytetyö
Syksy 2018
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Jarno Mattila
Opinnäytetyön nimi: Tiimi-portaalin hälytysten käsittely ja ohjaus
Työn ohjaaja: Kari Jyrkkä
Työn valmistumislukukausi ja -vuosi: Syksy 2018
Sivumäärä: 86

Opinnäytetyön aiheena oli toteuttaa kiinteistöautomaatioon liittyvien laitteiden palvelinkeskeinen hälytystenkäsittely- ja ohjaustoiminto. Tarkoituksena oli kehittää vaihtoehto olemassa olevalle hälytystenkäsittelylle, jossa laite lähetti hälytykset vastaanottajalle tekstiviestillä asennuskohteessa olevan modeemin avulla. Uudesta mallista haluttiin palvelinkeskeinen ratkaisu, jolloin hälytysviestien vastaanottajia ja lähetystapoja voisi olla useita. Lisäksi ratkaisu mahdollistaisi keskitetyn hälytystenvalvonnan ja antaisi mahdollisuuden hälytysviestien lähettämisestä aiheutuvien kustannusten kohdentamiseen.

Kehitystyön vaiheet olivat vaatimusmäärittely, oliokeskeisen analyysi ja suunnittelu, ohjelmointityö ja valmiin järjestelmän testaus. Vaatimusmäärittely perustui IEEE 830-1998(R2009) -standardiin ja analyysi ja suunnittelu toteutettiin UML-mallinnuksella. Ohjelmointikielenä käytettiin ensisijaisesti PHP-skriptikieltä. Muita käytettyjä ohjelmointitekniikoita olivat JavaScript (jQuery), HTML5, CSS3, SQL ja bash-skriptit. Työ sisälsi myös paljon Debian GNU/Linux -järjestelmän hallintaan liittyviä toimia, kuten systemd-palveluiden määritys ja järjestelmälokien hallinta rsyslog- ja logrotate-toiminnoilla.

Toteutettu järjestelmä käsittelee palvelimelle saapuvat hälytykset 10 sekunnin välein. Taustapalveluna toimiva järjestelmä selvittää vastaanottajat ja lähettää hälytykset sähköpostilla tai tekstiviestillä.

Järjestelmä vastasi sille asetettuja laatuvaatimuksia. Se sisältää vikasietoisen kahdennetun konfiguraation ja asetustietojen ennakkotarkistuksen. Monitasoinen lokitoiminto ja ylemmän tason palveluidenhallinta seuraavat järjestelmän toimintoja ja lähettävät virhetilanteista ilmoituksen sähköpostilla ylläpitäjälle.

Toteutettu hälytysjärjestelmä oli tavoitteen mukainen ja täytti sille asetetut toiminnalliset ja laadulliset vaatimukset.

Asiasanat: olio-ohjelmointi, PHP, Linux, ohjelmistosuunnittelu, vaatimusmäärittely

ABSTRACT

Oulu University of Applied Sciences
Degree programme, Information Technology, software development

---

Author(s): Jarno Mattila
Title of thesis: Tiimi Portal alarm handling and redirecting
Supervisor(s): Kari Jyrkkä
Term and year when the thesis was submitted: Autumn 2018
Pages: 86

The subject of this thesis was to design and implement a server based alarm handling and redirecting system for building service technology devices. The goal was to find an alternative solution for existing system, which sent alarms directly from device to receiver by SMS. The new server-based solution allowed multiple recipients and sending methods. Additionally, the new system allowed centralized alarm monitoring and message-based cost allocation to the device owners.

The steps of the system development were requirements specification, object-oriented analysis and design (OOAD), programming and system testing. The requirements specification was based on IEEE 830-1998(R2009) standard, and OOAD was made by UML modeling language. Primary programming language was PHP. Other programming technics were: JavaScript (jQuery), HTML5, CSS3, SQL and bash-scripting. The work included plenty of Linux system administrating work, such as implementing services under systemd and system log control by syslog and logrotate services.

Developed system handles incoming device alerts by 10 sec. interval. Running as a background service, the system resolves the recipients and sends the alerts by SMS or email.

The system met the assigned quality requirements. It contains a fault tolerance duplicated system configuration ja parameter validation. A multilevel logging system and higher-level system control monitor the system events and if error occurs, automatically send an alert message to system administrator by email.

The developed alarm system fulfilled the functional and quality requirements that were set for the thesis and the given subject.

Keywords: object-oriented programming, PHP, Linux, software development, requirements specification

ALKULAUSE

Haluan kiittää opinnäytetyön toimeksiannosta Team-Control Oy:tä sekä erityisesti Pekka Mäkelää ja Tommi Mäkelää työn ohjaamisesta, neuvoista, opastuksesta ja kaikesta tuesta. Kiitän myös Oulun ammattikorkeakoulun opettajia, Kari Jyrkkää opinnäytetyön ohjaamisesta ja kannustavasta palautteesta sekä Tuula Hopeavuorta avusta ja ohjauksesta opinnäytetyön kirjoittamisessa.

Lopuksi kiitän työpaikkaani, työtovereitani, perhettäni ja läheisiäni tuesta ja kannustuksesta opinnäytetyötä tehdessäni.

Oulaisissa 17.11.2018

Jarno Mattila

SISÄLLYS

SANASTO

BaseControl	Asiakasohjelma, jolla käytetään säädinlaitteita.
BaseRelay	Palvelinohjelma, joka vastaa säädinlaitteen ja asiakasohjelman välisestä tiedonsiirrosta.
Bash ja shell	Linux-järjestelmän komentotulkkeja.
Debian GNU/Linux	Linux-pohjainen käyttöjärjestelmäjakelu eli distribuutio.
DPKG	Debian-järjestelmän ohjelmapakettien asennustyökalu.
GPRS	General Packet Radio Service, GSM-verkossa toimiva tiedonsiirtopalvelu.
GSM	Global System of Mobile Communications, matkapuhelinjärjestelmä.
HTML	Hyper Text Markup Language, web-ohjelmoinnissa käytetty merkkauskieli.
HTTP	Hypertext Transfer Protocol, internetissä yleisesti käytetty tiedonsiirtoprotokolla.
IoT	Internet of Things, esineiden internet. Teollinen internet-verkko, jossa fyysisiä laitteita käytetään ja hallitaan internet-verkon välityksellä.
Logrotate	Järjestelmälokien hallintatyökalu.
MariaDB	MySQL-relaatiotietokantaan perustuva avoimen lähdekoodin tietokantapalvelin-ohjelma
MySQL	Relaatiotietokanta.
MTA	Mail Transfer Agent, yleisnimitys sähköpostin lähettämiseen ja vastaanottamiseen käytettävistä ohjelmista.

PHP	PHP: Hypertext Preprocessor, avoimen lähdekoodin tulkattava skriptikieli.
PHP-CLI	PHP Command Line Interface, komentotulkista käytettävä PHP-tulkki.
Postfix	Sähköpostin lähettämiseen ja vastaanottamiseen käytetty ohjelma (MTA)
RFC	Requests for Comments, internet-standardien kokoelma.
SAPI	Server Application Programming Interface, palvelimella käytettävä ohjelmointirajapinta.
SMTP	Simple Mail Transfer Protocol, TCP-pohjainen protokolla (RFC-5321) sähköpostiviestien välittämiseen.
SMS	Short Message Service, tekstiviesti.
SQL	Structured Query Language, relaatiotietokantojen hallinnassa käytetty kyselykieli.
Systemd	Debian-järjestelmän hallintapalvelu.
Systemctl	Systemd-palvelun hallintaohjelma.
TCP	Transmission Control Protocol, internet-verkossa yleisesti käytetty tietoliikenneprotokolla (RFC 793).
UI	User Interface, käyttöliittymä.
UML	Unified Modeling Language, ohjelmistokehityksessä käytetty graafinen mallinnuskieli.

1 JOHDANTO

Tiimi-portaali on opinnäytetyön tilaajan, TeamControl Oy:n, kehittämä verkkopalvelu, jonka avulla IoT-säädinlaitteiden käyttäjät voivat käyttää ja hallita laitteitaan suojatun internetyhteyden avulla. Säädinlaitteet ovat kiinteistöautomaatioon kuuluvia laitteita, kuten lämmitys- ja ilmanvaihtosäätimet ja hälytysjärjestelmät.

Tiimi-portaalin kehitystyötä on tehty opetussuunnitelmaan kuuluvien yrityslähtöisten tuotekehitysprojektien aikana. Ensimmäisen projektin aikana kehitettiin säädinlaitteen selainpohjaista käyttöliittymää, toinen projekti keskittyi portaalin palvelinympäristön suunnitteluun ja toteutukseen ja kolmannessa projektissa kehitettiin portaalin käyttöliittymää. Portaali on edelleen kehitysvaiheessa, joten oli luontevaa valita opinnäytetyön aihe siten, että se hyödyttää ja täydentää aikaisemmin tehtyä työtä.

Opinnäytetyön tavoitteena on kehittää palvelinsovellus, joka valvoo säädinlaitteilta tulevia hälytyksiä, tunnistaa hälytysten kohteet ja luonteen ja välittää hälytykset edelleen käyttäjille eri viestikanavia käyttäen. Kehitettävän sovelluksen on lisäksi tallennettava hälytyshistoria vaaditussa muodossa ja säilytettävä tietoja määrätyn aikaa.

Laitehälytykset ovat järjestelmien toimivuuden ja turvallisuuden kannalta kriittisiä toimintoja, joten kehitettävän sovelluksen toimintavarmuus ja vikasietoisuus korostuvat. Sovelluksen on toimittava luotettavasti ja hallittava virhetilanteiden käsittely. Sovelluksen on myös valvottava omaa tilaansa ja hälytettävä ylläpitäjää vikatilanteen tai odottamattoman toiminnon sattuessa.

Opinnäytetyössä perehdytään laitehälytysten käsittelyyn, relaatiotietokannan käyttöön ja Debian Linux -palvelinkäyttöjärjestelmän tarjoamiin palveluihin. Lisäksi perehdytään tekstiviestien ja sähköpostien ohjelmalliseen lähettämiseen. Ensisijaisena ohjelmointikielenä on PHP skriptikieli. Lisäksi työssä käytetään Linux-komentoskriptejä ja SQL-kyselykieltä sekä yleisiä web-ohjelmointitekniikoita.

2 TAVOITTEET

Opinnäytetyön tavoitteena on toteuttaa Tiimi-ohjelmaportaaliin integroitu hälytysten ohjaustoiminto. Toiminnon tarkoituksena on valvoa säädinlaitteilta saapuvia hälytysviestejä, tunnistaa kohdelaite ja hälytyksen syy ja välittää hälytys kohdelaitteen omistajalle. Lopuksi toiminnon on tallennettava tapahtuma lokitiedostoon.

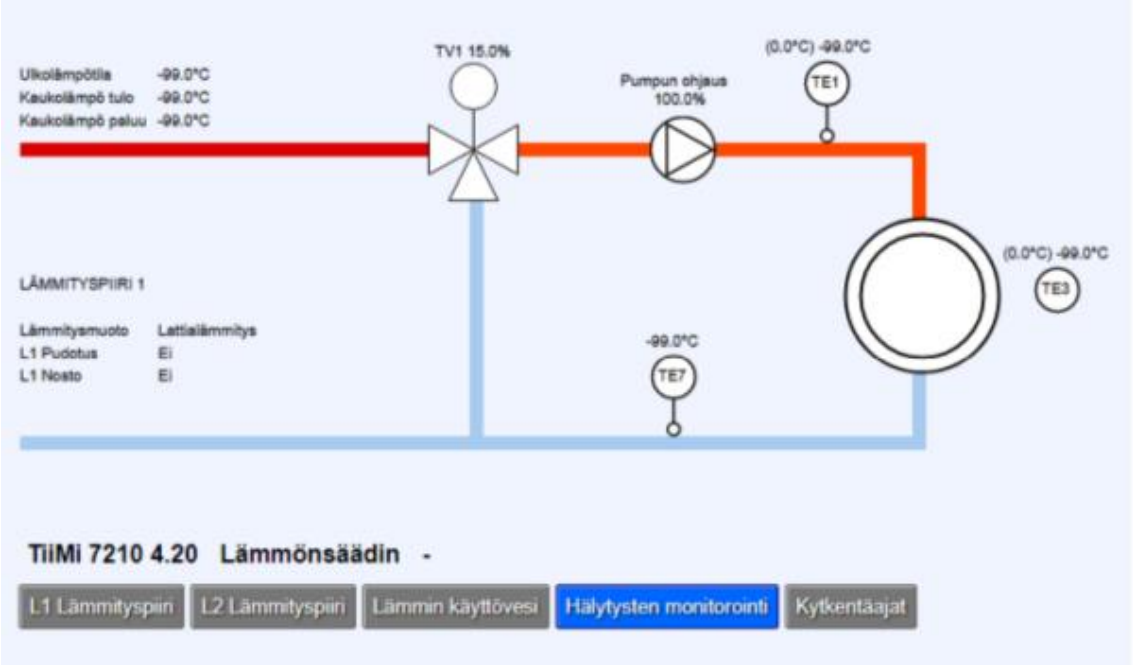
2.1 Lähtötilanteen esittely

Hälytysten käsittelytoiminto on tarkoitus lisätä olemassa olevaan järjestelmään, jonka osat ovat mikrokontrollerilla ohjattava säädin, yhteysmodeemi, BaseRelay-palvelinohjelma ja asiakassovellukset. Säädinlaitteita ovat esimerkiksi TiiMi 7210B lämmityksen säädin (kuva 1), TiiMi 7210B ilmanvaihdon säädin ja TiiMi 7610 hälytys- ja ohjausmoduuli (1).

KUVA 1. TiiMi 7210B ilmanvaihdon säädin (2)

Asiakasohjelmat

Asiakasohjelmia ovat Windows-PC-tietokoneella käytettävä BaseControl-hallintaohjelma ja selainkäyttöinen Web-valvomo (kuva 2). Asiakasohjelmat sisältävät säädinkohtaiset käyttöliittymät. Ohjelmat ovat yhteydessä säätimiin BaseRelay-palvelinsovelluksen avulla.



KUVA 2. TiiMi Web-valvomo, lämmityspiirin käyttöliittymä (3)

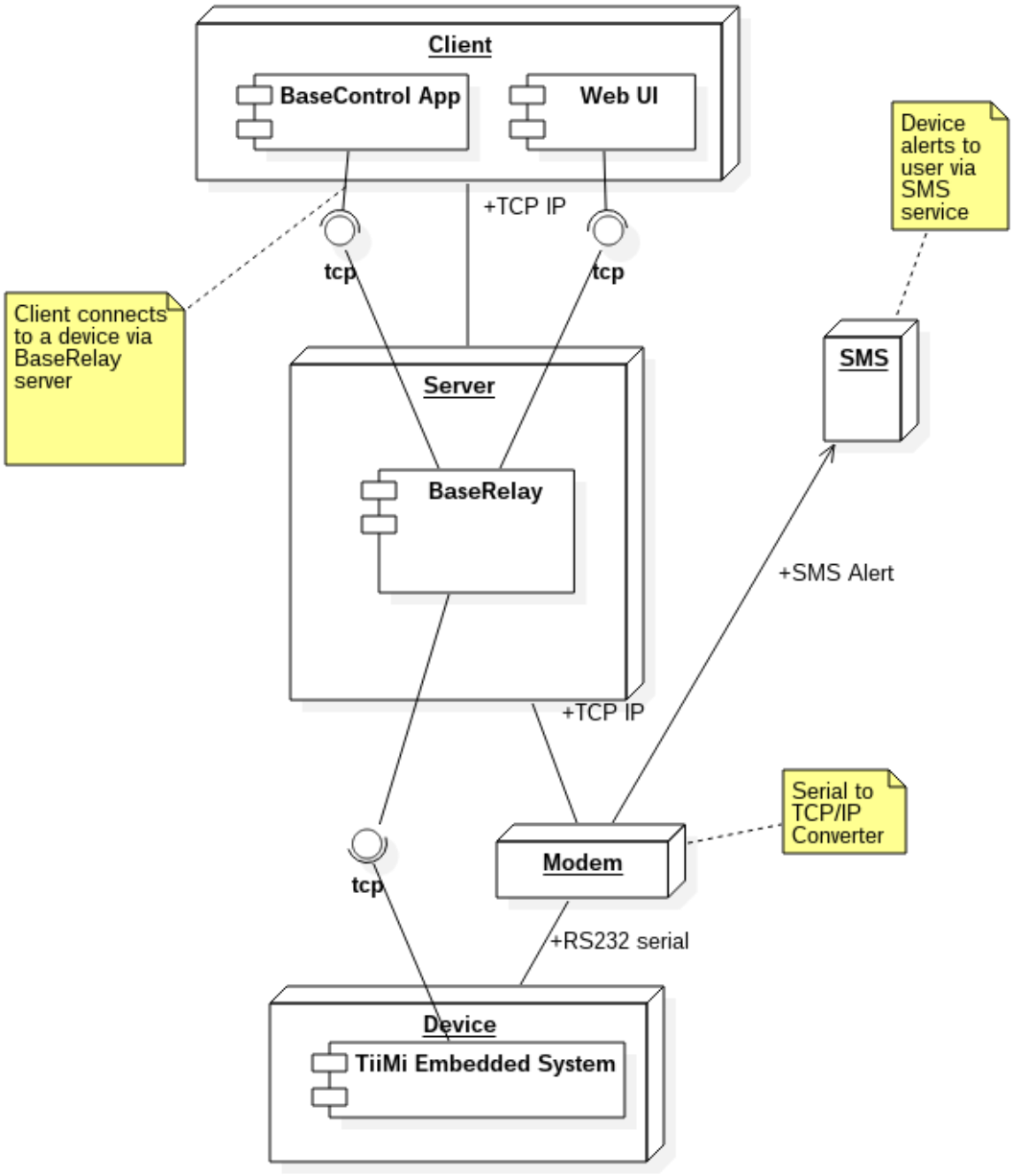
Laiteyhteydet

Yhteysmodeemi on GSM/GPRS-modeemiterminaali (kuva 3). Laitteen avulla voidaan toteuttaa säätimien hälytysten lähetys tekstiviestein käyttäjän matkapuhelimeen. Laite on kytketty säätimeen 9-pinnisen RS232-sarjaliikenneportin kautta, jolloin se mahdollistaa säätimen etähallinnan asiakasohjelmalla GPRS-yhteyden avulla (4).

KUVA 3. TiiMi GSM-Set GSM/GPRS-modeemiterminaali (4)

BaseRelay-palvelinsovellus

BaseRelay on järjestelmän palvelinsovellus, jonka kautta asiakasohjelmat muodostavat yhteyden säätimiin. Yhteyden muodostus tapahtuu asiakasohjelmasta käsin. Ohjelma muodostaa yhteyden BaseRelay-palvelimeen, tunnistautuu ja välittää tiedon, mihin modeemiin tai laitteeseen yhteys muodostetaan. BaseRelay-palvelin huolehtii tiedonsiirron reitityksestä asiakasohjelman ja oikean laitteen välillä. (Kuva 4.)



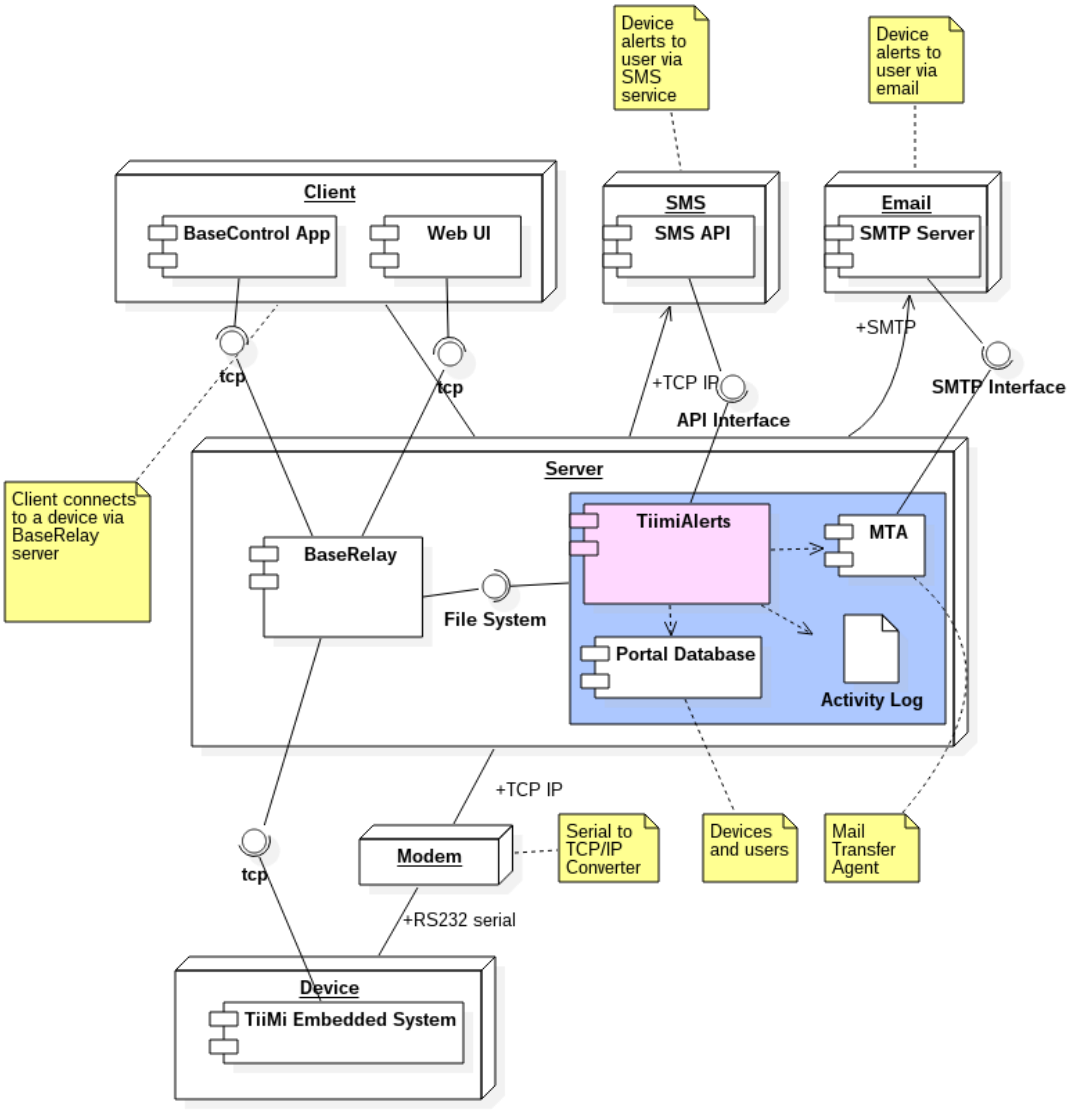
KUVA 4. Toteutuskaavio palvelinkeskeisestä BaseRelay-järjestelmästä

2.2 Tavoitteet

Kehitettävän sovelluksen ensisijainen tehtävä on laitehälytysten käsittely ja ohjaus. Lähtökohtaisesti kyseessä on yksinkertainen taustatoiminto, joten sovellus ei sisällä esimerkiksi omaa käyttöliittymää. Sovelluksen suorittaman tehtävän luonne, hälytysten käsittely, asettaa kuitenkin laadullisia lisävaatimuksia. Hälytysten käsittely on kokonaisuuden kannalta kriittinen toiminto. Esimerkiksi laitevikasta aiheutuvan hälytyksen nopea ja luotettava välittäminen on tärkeää.

2.2.1 Toiminnallinen tavoite

Työn tavoitteena on kehittää edellä kuvatulle järjestelmälle palvelinkeskeinen hälytystenkäsittely- ja ohjaustoiminto. Uusi toiminto täydentää edellä kuvattua BaseRelay-ympäristöä kuvan 5 toteutuskaavion mukaisesti. Palvelinkeskeisessä hälytystenkäsittelyssä BaseRelay-palvelinsovellus ottaa vastaan laitteilta tulevat hälytysviestit ja aktivoi hälytystoiminnon. Hälytystoiminto hyödyntää portaalin tietokannan laite- ja asiakastietoja ja lähettää hälytysviestit käyttäjille joko teksti- tai sähköpostiviestinä. Lähetystapa voidaan valita hälytyskohtaisesti ja hälytysviesti voidaan lähettää myös molemmilla tavoilla.



KUVA 5. Toteutuskaavio BaseRelay-järjestelmästä, laajennettuna palvelinkeskeisellä hälytysten käsittelyllä

Palvelinkeskeisellä järjestelmällä saavutetaan luotettavampi ja monipuolisempi hälytysten käsittely. Hälytykset voidaan lähettää kullekin asiakkaalle parhaiten soveltuvalla tavalla, jolloin ne menevät varmemmin perille. Vaihtoehtoinen viestikanava voi toimia varayhteytenä, parantaen hälytystoiminnon vikasietoisuutta.

Palvelinkeskeinen hälytysten käsittely mahdollistaa myös hälytyshistorian tallentamisen. Hälytyshistoriaa voidaan hyödyntää hälytyksistä aiheutuvien kustannusten kohdentamisessa, järjestelmän jatkokehittämisessä ja mahdollisten vika- ja ongelmatilanteiden selvittämisessä.

2.2.2 Laatutavoitteet

Tavoitteena on kehittää vikasietoinen ja luotettava toiminto, joka käynnistyy automaattisesti ja toimii palvelimen taustaprosessina. Hälytystoiminto toimii siis itsenäisesti, muusta järjestelmästä riippumatta, ja on aina käynnissä.

Hälytystoiminnon on osattava valvoa omaa tilaansa, tunnistaa mahdolliset toimintahäiriöt ja vikatilanteet ja ilmoittaa ongelmatilanteista ylläpitäjälle. Toiminnon on myös ylläpidettävä toimintolokia, johon tallentuvat kaikki tapahtumat, mukaan lukien palvelun käynnistymiset, viestienkäsittelyn vaiheet ja mahdolliset vikatilanteet.

2.2.3 Sovellussuunnittelun tavoitteet

Kehitettävä sovellus on järjestelmän toimivuuden kannalta kriittinen taustapalvelu, joka toimii järjestelmäkokonaisuuden osana, valvoo sen toimintaa ja reagoi mahdollisiin vikatilanteisiin. Sovellus on riippuvainen järjestelmän muista osista, joiden ohjelmointirajapinnat on tunnettava, ennen kuin ohjelmaa voidaan ryhtyä tekemään. Toiminnan kannalta kriittisen sovelluksen (mission critical) kehittämisen on suositeltavaa perustua vähintään perustason ennakkosuunnitteluun, joka sisältää vaatimusmäärittelyyn perustuvan analyysin ja suunnittelun, ainakin tärkeimpien toimintojen osalta. Lisäksi toiminto testataan testaussuunnitelman mukaisesti. Versionhallinnan avulla varmistetaan ohjelmaversion yhteensopivuus järjestelmän kanssa. (5, s. 31.)

Tavoitteena on laatia kehitettävälle sovellukselle vaatimusmäärittely, jossa kuvataan tehtävät ja vaatimukset. Vaatimusmäärittelyn pohjalta suoritetaan analyysi, jossa tärkeimmät luokat ja aktiviteetit kuvataan UML-mallinnuksena.

3 KÄYTETYT TEKNOLOGIAT

3.1 Debian GNU/Linux

Debian GNU/Linux on yleisnimitys Debian projektin tuottamalle Linux-pohjaiselle käyttöjärjestelmäjakelulle eli distribuutiolle. Nimi ”GNU/Linux” viittaa Debian käyttöjärjestelmän Linux-ytimeen ja GNU-projektin kehittämiin Linux-yhteensopiviin työkalu- ja apuohjelmiin. Ne yhdessä muodostavat Debian käyttöjärjestelmän perustan.

GNU/Linux-perustan lisäksi Linux-jakelun keskeisiä elementtejä ovat pakettienhallintajärjestelmä ja automaattinen asennustyökalu. Niiden tarkoituksena on helpottaa ohjelma-asennuksia automatisoimalla asennusprosessia. Paketinhallintajärjestelmällä luodaan ja asennetaan ohjelmia ja ohjelmakirjastoja ja automaattinen asennustyökalu ylläpitää tietoa asennuslähteistä ja pakettien välisistä riippuvuuksista. Debian jakelu sisältää DPKG-pakettienhallintajärjestelmän ja APT-asennustyökaluohjelmiston. (6, s. 20.)

GNU projekti on Richard Stallmanin johtaman Free Software Foundation -säätiön (FSF) ohjelmistokehitysprojekti, joka kehittää avoimen lähdekoodin vapaita GPL-lisensoituja ohjelmia (6, s. 2). Linux on Linus Torvaldsin kehittämä käyttöjärjestelmäydin eli kernel, jonka ensimmäinen versio julkaistiin vuonna 1991 (7).

Debian on Ian Murdockin 1993 käynnistämä projekti, jonka tavoitteena oli kehittää Linux-ytimen ympärille laadukas, ei-kaupallinen ja huolellisesti ylläpidetty käyttöjärjestelmä (6, s. 2).

Opinnäytetyön palvelinalustana käytetään olemassa olevaa Tiimi-portaalin palvelinta, jonka käyttöjärjestelmä on Debian GNU/Linux 9, koodinimeltään ”*Stretch”*. Palvelimen sisältämiä työssä käytettäviä käyttöjärjestelmäpalveluja ovat Bash-komentotulkki, taustapalveluiden hallinta (systemd) ja tapahtumalokien hallinta (logrotate ja rsyslog).

3.2 GNU Bash

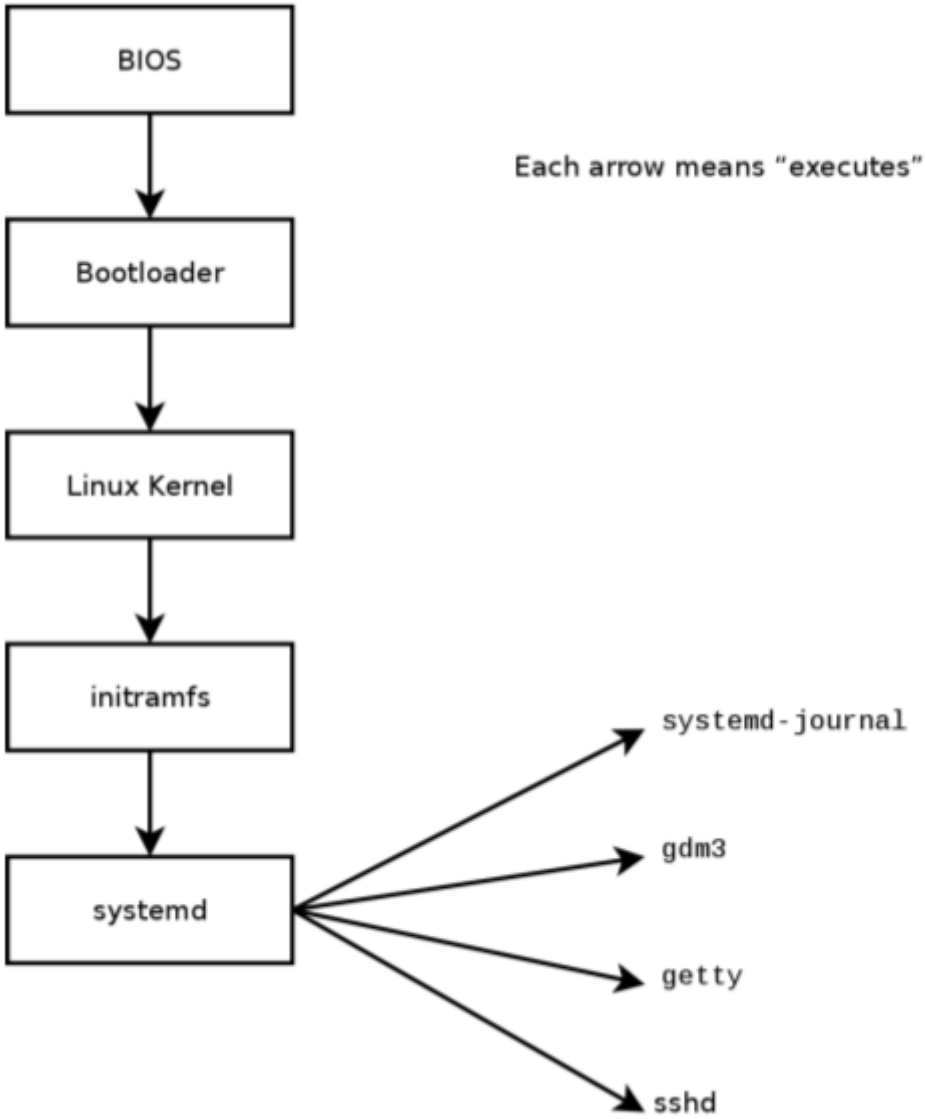
Bash on Linux-yhteensopiva komentotulkki (shell). Nimi *Bash* on akronyymi sanoista Bourne-Again Shell, mikä viittaa alkuperäisen Unix shell -ohjelman tekijään Stephen Bourneen. Bash on laajalti yhteensopiva Unix-shellin kanssa. Se sisältää lisäksi käyttöä helpottavia toimintoja, kuten komentohistoria ja tabulaattorilla toimiva automaattitäydennys. (6, s. 134.)

Shellin avulla voidaan suorittaa komentoja, käynnistää ohjelmia ja käyttää tiedostojärjestelmää. Komennot voidaan syöttää suoraan komentokehotteeseen kirjoittamalla tai tallentaa skriptitiedostoksi ja käynnistää komennot suorittamalla tiedosto. Skriptitiedostot voivat sisältää useita komentoja, ehtolauseita ja silmukoita, joten ne ovat itsessään pieniä ohjelmia. Shell on siis komentotulkin lisäksi ohjelmointikieli. Debian jakelun oletuskomentotulkki on GNU Bash. (6, s. 134.)

Tässä työssä shell-skriptejä käytetään vaihtoehtoisena ohjelmointikielenä, silloin kun ne soveltuvat käyttötarkoitukseen parhaiten. Tällaisia tilanteita ovat esimerkiksi palveluiden käynnistäminen ja pysäyttäminen ohjelmallisesti.

3.3 Systemd

Systemd (oikea kirjoitusasu on systemd, pienellä alkukirjaimella) on Linux-pohjaisen Debian käyttöjärjestelmän pääprosessi, joka on ensimmäinen käynnistyvä prosessi ja hallinnoi kaikkia muita järjestelmän prosesseja (kuva 6). Systemd on taaksepäin yhteensopiva aikaisempien palveluiden hallintajärjestelmien kuten SysV initin kanssa, joten se tukee myös vanhoja palvelumäärityksiä. (8, luku 1.)



KUVA 6. Systemd pääprosessia käyttävän Linux järjestelmän käynnistysjärjestys (6, s. 183)

Palveluiden määrittämisen ja käynnistämisen lisäksi systemd-palvelua voidaan käyttää muiden palveluiden valvontaan ja ajastamiseen. Se huolehtii palveluiden välisistä riippuvuuksista ja oikeasta käynnistysjärjestyksestä.

Erityisiä vakautta ja vikasietoisuutta lisääviä ominaisuuksia ovat palveluiden eristäminen ja järjestäminen hallintaryhmiksi (control group) ja jumiutuneen palvelun uudelleenkäynnistys, liitäntäpistettä (socket) menettämättä. Control group (cgroup) -toiminnon avulla Linux kernel eristää palvelut toisistaan ja osoittaa niille tarvittavat resurssit.

Socket-toiminnon avulla varmistetaan yhteyden säilyminen palvelua mahdollisesti käyttävien muiden komponenttien kanssa. Järjestelmän komponentit voivat kommunikoida keskenään socket-liitäntäpisteiden avulla, joita palvelut muodostavat käynnistyessään. Kun palvelu pysäytetään, socket poistuu ja yhteydet katkeavat. Systemd pystyy käynnistämään jumiutuneen palvelun uudelleen socketia menettämättä, jolloin palvelua käyttävät komponentit eivät häiriinny uudelleenkäynnistyksestä. (8, luku 1.)

Systemd-palveluita hallitaan systemctl-ohjelman avulla. Systemctl (oikea kirjoitusasu on systemctl pienellä alkukirjaimella) sisältää komentoja sekä palveluiden että koko järjestelmän hallintaan. Komentojen avulla palveluita otetaan käyttöön, käynnistetään, sammutetaan ja poistetaan käytöstä. Myös palveluiden tilat voidaan tarkistaa systemctl-komennoilla. Esimerkkejä systemctl-ohjelman komennoista on esitelty taulukossa 1. (8, luku 2.)

TAULUKKO 1. Usein käytettyä systemctl komentoja (8, luku 2.2)

Komento	**Selitys**
systemctl enable|disable [service]	enable aktivoi palvelun, disable poistaa palvelun käytöstä
systemctl mask|unmask [service]	sama toiminto kuin enable|disable, mutta voimakkaampi, estää palvelun aktivoimisen millään tavalla.
systemctl start|stop|restart [service]	palvelun käynnistys, pysäytys tai uudelleenkäynnistys
systemctl status [service]	näyttää palvelun tilan
systemctl show [service]	näyttää palvelun ominaisuudet (konfiguraatio)
systemctl halt|poweroff|reboot	järjestelmän sammutus- ja uudelleenkäynnistyskomentoja

Tiimi-portaalin hälytysten käsittelyssä systemd vastaa toiminnon ajastuksesta ja valvonnasta. Ajastus toteutetaan systemd timer -yksilöllä ja prosessin valvonta

systemd service -yksiköllä. Toiminnon pääohjelma määritetään palveluna (service), jonka tilaa systemd valvoo.

3.4 Logrotate

Logrotate-ohjelma (oikea kirjoitusasu on logrotate, pienellä alkukirjaimella) on tarkoitettu helpottamaan sellaisten järjestelmien hallintaa, jotka tuottavat runsaasti lokitietoa. Logrotaten avulla voidaan kierrättää, pakata ja poistaa lokitiedostoja automaattisesti. Logrotate voi myös lähettää lokitietoa sähköpostilla. Logrotate voi käsitellä lokitiedostot ajastetusti päivä-, viikko- tai kuukausisyklillä tai silloin, kun lokitiedoston koko kasvaa yli määrätyn raja-arvon. (9.)

Hälytysten käsittelyn kaikki toimet on tarkoitus tallentaa lokitiedostoon, mukaan lukien ajastettuna tapahtuva saapuneiden hälytysten tarkistus. Lokitietoa tallentuu koko ajan, vaikka hälytyksiä ei tulisikaan. Logrotaten avulla varmistetaan, että lokitietoa säilytetään tarpeeksi kauan, kuormittamatta kuitenkaan palvelimen levyresurssia tarpeettomasti.

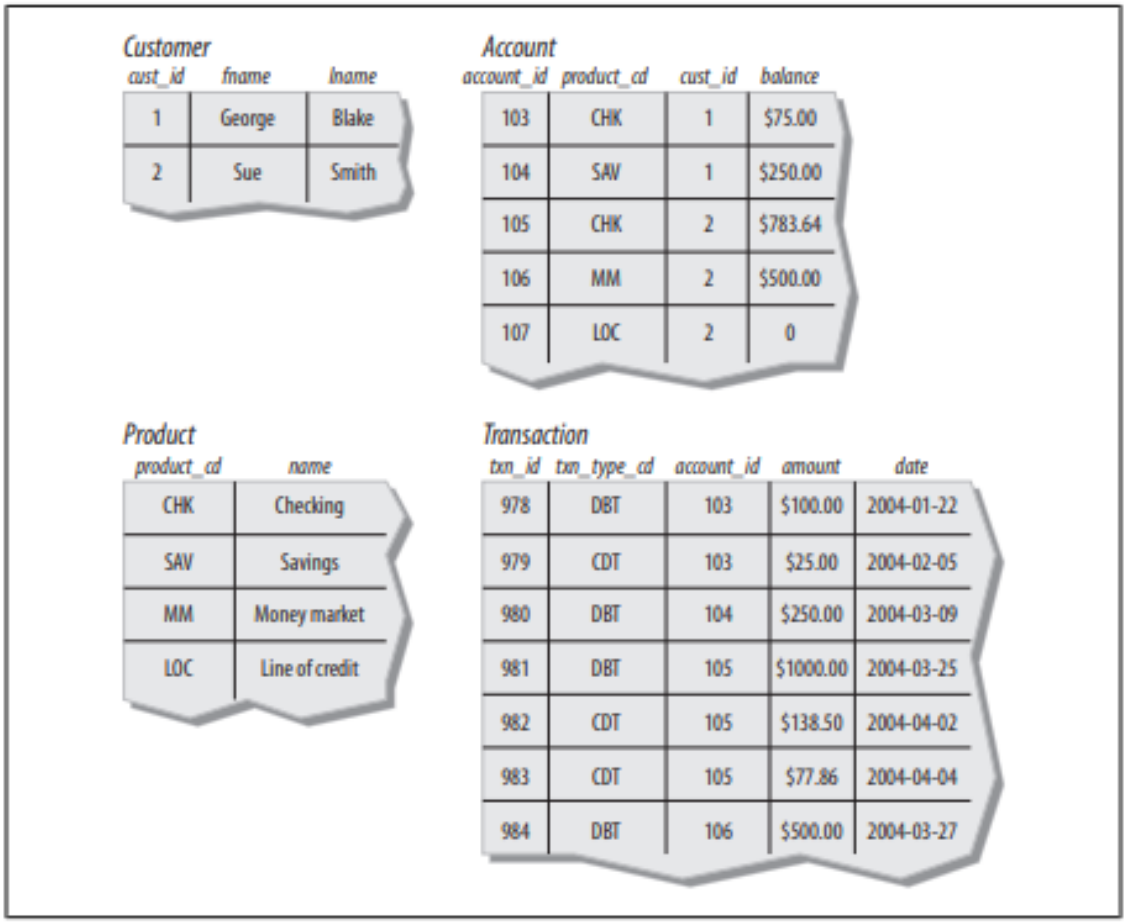
3.5 Postfix

Postfix on palvelimen sähköpostiliikenteestä vastaavat ohjelma, eli MTA (Mail Transfer Agent) (10, s. 22). Alun perin IBM:n kehittämä ohjelma on laajasti käytetty vaihtoehto Unix-pohjaisissa järjestelmissä aikaisemmin käytetylle sendmail MTA -ohjelmalle (10, s. 16). Debian-jakelun pakettienhallintajärjestelmällä asennettu Postfix sisältää ohjatun asennustoiminnon, jonka avulla ohjelman käyttöönotto on helppoa. Ohjattua asennusta seuraamalla ohjelma myös asentuu turvallisesti tarkasti rajatuilla käyttöoikeuksilla, jolloin anonyymin sähköpostiliikenteen välittäminen on oletusarvoisesti estetty. Postfix on sendmailin kanssa yhteensopiva, jolloin yleisimmin käytetyt sendmail-komennot toimivat sellaisenaan Postfixissa. (6, s. 252.)

Tässä työssä Postfixia käytetään sendmailin tapaan välittämään hälytystoiminnon tila- ja lokiviestejä ylläpitäjälle systemd tai logrotate ohjelmista tai PHP skriptillä mail-funktiota käyttäen.

3.6 MariaDB

MariaDB on vuonna 2009 kehitetty avoimen lähdekoodin relaatiotietokantaohjelma. Se perustuu alkujaan avoimen lähdekoodin MySQL-tietokantaan ja on suurelta osin yhteensopiva MySQLn kanssa (11). Relaatiomalli esiteltiin vuonna 1970, julkaisussa ”A Relational model of Data for Large Shared Data Banks” (Edgar F. Codd, IBM). Relaatiotietokannassa data on järjestetty taululuihin, jotka voivat sisältää viittauksia, eli relaatioita, toisiin tauluihin. Viittaukset toteutetaan käyttämällä tauluissa ylimääräisiä tunnistekenttiä, joiden avulla taulussa olevaan tietoon voidaan viitata toisessa taulussa. Esimerkiksi taulussa Customer olevaan asiakkaaseen voidaan viitata taulussa Account, käyttämällä Customer taulun cust_id tunnistekenttää (kuva 7). (12, s. 4.)



Customer

cust_id	fname	lname
1	George	Blake
2	Sue	Smith

Account

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LOC	2	0

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market
LOC	Line of credit

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

KUVA 7. Esimerkki relaatiotietokannasta ja taulujen välisistä relaatioista (12, s. 4)

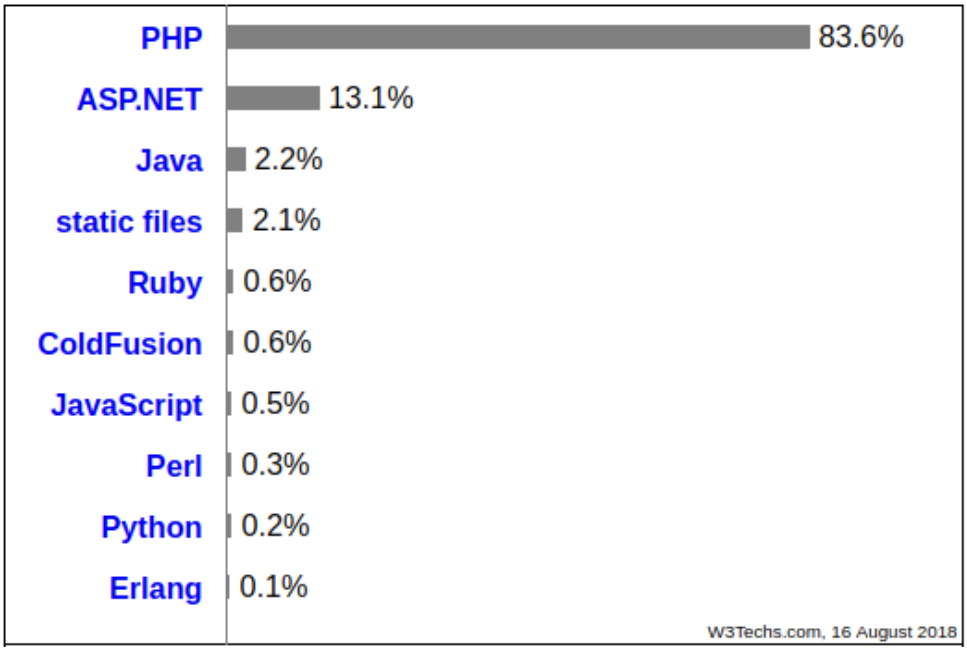
Relaatiomallin esitellyssä julkaisussa ehdotettiin uuden kielen kehittämistä, joka olisi tarkoitettu erityisesti relaatiotietokannan käyttöön. IBM-yhtiön käynnistämän

kehitysprojektin seurauksena kehitettiin sequel-kieli eli SQL (Structured Query Language). SQL on kyselykieli, jolla tietoa käsitellään relaatiotietokannassa. SQL-kyselylauseella luettu tieto vastaa rakenteeltaan tietotaulua, joten kyselyn tuloksena saatu taulukko voidaan tallentaa uutena tietotauluna tai siihen voidaan yhdistää muiden taulujen tai SQL-kyselyjen tuloksia. (12, s. 7.)

Tässä työssä hyödynnetään olemassa olevaa Tiimi-portaalin relaatiotietokantaa, joka on toteutettu MariaDB-tietokantaohjelmalla. Tietokannasta haetaan hälytysten välittämiseen tarvittavat laite- ja asiakastiedot. Tietokantaan tallennetaan hälytyshistoria ja lähetettävien hälytysten muodostama hälytysjono.

3.7 PHP ja PHP-CLI

PHP (Personal Home Page, tai PHP: Hypertext Preprocessor) on suosittu yleiskäyttöinen skriptikieli, joka on varsinaisesti kehitetty web-ohjelmointia varten (kuva 8) (13).



KUVA 8. Suosituimmat verkkosivuilla käytetyt palvelinpuolen skriptikielet, elokuu 2018 (14)

PHP on tulkattava skriptikieli. PHPllä ohjelmoitu lähdekoodi tallennetaan palvelimelle, usein upotettuna HTML-dokumenttiin. PHP-skripti merkitään HTML-dokumenttiin erityisellä merkillä (tagi), jonka HTTP-palvelin tunnistaa ja suorittaa PHP-tulkin avulla. Tämän jälkeen HTTP-palvelin palauttaa tulkin käsittelemän PHP-

skriptin tulosteen, joka on yleensä dynaamista HTML- tai XML-muotoista tekstiä. (13.)

PHP-skriptikieltä voidaan käyttää myös ilman HTTP-palvelinta, komentokehotteessa. Tällöin PHP-kieltä käytetään bash-skriptin tapaan. PHP-skriptiä kutsutaan shell-konsolissa PHP-CLI SAPIn (Server Application Programming Interface) avulla, joka suorittaa skriptin ja palauttaa tulosteen shell-konsoliin. PHP-CLI SAPI käyttäytyy samalla tavalla kuin HTTP-palvelin, vaikkakin joitain eroavaisuuksia on. PHP-CLI SAPI ei esimerkiksi lähetä HTTP-otsikoita (header) oikean HTTP-palvelimen tapaan, eikä sen avulla suoraan voi käsitellä GET- tai POST-metodilla lähetettyä tietoa. PHP-CLI SAPI sisältää kuitenkin suurimman osan PHP-kielen funktioista ja luokista, joten yleisesti ottaen PHP-CLI SAPI ympäristöön tarkoitettu PHP-koodi on sellaisenaan toimivaa myös HTTP-palvelimen kanssa käytettynä. (15, s.7-8.)

Tässä työssä PHP on ensisijainen ohjelmointikieli ja sen käyttöympäristö on PHP-CLI SAPI. PHP soveltuu käyttötarkoitukseen hyvin sen polymorfisuuden, eli monikäyttöisyyden vuoksi. Tiimi-portaalin muu palvelinpuolen ohjelmointi on toteutettu PHP-kielellä, joten PHP-CLI SAPI -ympäristöön ohjelmoituja luokkia ja funktioita voidaan mahdollisesti hyödyntää myös muissa Tiimi-portaalin toiminnoissa. PHP on myös ennestään tuttu ohjelmointikieli Tiimi-portaalin kehittäjille, joten sillä kirjoitettua ohjelmakoodia on helppo ymmärtää ja ylläpitää kehittäjien toimesta.

3.8 TortoiseHg (Mercurial)

TortoiseHg on graafinen versionhallintaohjelmisto, joka toimii käyttöliittymänä Mercurial-versionhallintajärjestelmälle. Versionhallintajärjestelmän avulla ylläpidetään paikallista säilökansiota (repository) projektin tiedostoista. Versionhallinta mahdollistaa myös tiedostoversioiden säilyttämisen, samanaikaisten versiotiedostojen ylläpitämisen ja säilökansion jakamisen.

Tässä työssä TortoiseHg-versionhallintaa käytetään koodien jakamiseen ja varmuuskopiointiin kloonaamalla säilökansio eri sijaintiin.

4 VAATIMUSMÄÄRITTELY

Vaatimusmäärittely auttaa ohjelman kehittäjää ymmärtämään, mitä asiakas tarkalleen ottaen haluaa ohjelman tekevän. Vastaavasti se auttaa asiakasta kuvaamaan täsmällisesti millainen ohjelman tulisi olla. Hyvä vaatimusmäärittely luo perustan asiakkaan ja ohjelman kehittäjän väliselle sopimukselle. Se keventää kehitystyötä vähentämällä ylimääräistä suunnittelu-, ohjelmointi- ja testaustyötä. Vaatimusmäärittely tarjoaa myös perustan kustannuslaskennalle ja toimitusaikataululle. (16, johdanto.)

Vaatimusmäärittely voi toimia perustana kehitettävän ohjelman arvioinnille ja testaukselle. Siinä voidaan esimerkiksi määritellä perusteet, mitä toimintoja ja millä tavalla on tarkoitus testata ja arvioida. (16, johdanto.)

Asiakkaan osallistuminen vaatimusmäärittelyn tekemiseen helpottaa valmistautumista tulevaan käyttöönottoon. Vaatimusmäärittelyn ansiosta asiakasorganisaatiolla on hyvä käsitys kehitettävästä ohjelmasta, jolloin se voi valmistautua käyttöönottoon perehdyttämällä käyttäjiä etukäteen tulevan ohjelman ominaisuuksiin ja suunnittelemalla käyttöönoton vaiheet etukäteen. (16, johdanto.)

Vaatimusmäärittely kuvaa kehitettävää tuotetta, ei kehittämistyötä. Näin se voi toimia perustana jo käyttöönotetun ohjelman myöhemmälle kehitystyölle. (16, johdanto.)

4.1 Vaatimusmäärittelyn sisältö

Tiimi-portaalin hälytysten käsittely- ja ohjaustoiminnon vaatimusmäärittely perustuu IEEE:n suositelluille käytännöille vaatimusmäärittelyn tekemiseen IEEE standardi 830-1998(R2009) (16). Standardi kuvaa suositeltuja lähestymistapoja ohjelmiston vaatimusmäärittelyn tekemiseen. Se sisältää vaatimusmäärittelyssä käytettävien avaintermien määritelmät, kuvauksen vaatimusmääritelmän käsitteestä ja hyvälle vaatimusmääritelmälle tunnusomaisista piirteistä, kuten yksiselitteisyys, kattavuus ja johdonmukaisuus.

Standardin mukaan vaatimusmäärittelyssä mainittujen vaatimusten tulee olla todennettavissa kustannustehokkaasti. Vaatimukset kuten ”toimii hyvin” tai ”hyvä käyttöliittymä” eivät ole todennettavissa, koska termejä ”hyvä” ja ”hyvin” ei voi tarkasti määritellä. Hyvä vaatimus on täsmällinen ja siinä esiintyvät suureet ovat mitattavissa. Esimerkiksi vaatimus ”ohjelman vasteaika on oltava enintään 20 sekuntia 60 % käyttötapauksista ja enintään 30 sekuntia 100 % käyttötapauksista” on täsmällinen, koska siinä käytetään täsmällisiä termejä ja mitattavia määriä. (16, s. 7.)

Hyvä vaatimusmäärittely on helposti muokattavissa, rikkomatta dokumentin alkuperäistä rakennetta ja tyyliä. Vaatimusten tulee olla myös jäljitettävissä sekä eteen- että taaksepäin. Taaksepäin jäljitettävyys tarkoittaa, että vaatimusmäärittelyyn tehdyt muutokset kehityksen eri vaiheissa voidaan jäljittää. Eteenpäin jäljitettävyys tarkoittaa alkuperäisestä vaatimusmäärittelystä erotettujen osien jäljitettävyyttä. Tällöin voidaan huomioida alkuperäiseen dokumenttiin tehtyjen muutosten vaikutus mahdollisiin muihin järjestelmiin, jotka perustuvat vaatimusmäärittelyn osiin. Jäljitettävyys voidaan saavuttaa esimerkiksi vaatimusmäärittelyn eri versioiden yksilöllisellä nimeämisellä ja numeroiduilla viittauksilla eri versioihin. (16, s. 8.)

4.2 Vaatimusmäärittelyn rakenne

Standardi sisältää liiteosan, jossa on vaatimusmäärittelyn mallipohjia järjesteltynä eri lähestymistapojen mukaan. Esimerkkejä eri lähestymistavoista ovat olio- ja luokkaperusteinen, käyttötapausperusteinen, ärsykeperusteinen ja toimintoperusteinen. Valmis mallipohja kuvaa vaihtoehtoisten vaatimusmäärittelyjen rakenteen otsikoittain jäsenneltynä (16, liite A). Standardin sisältämät mallit ovat suosituksia. Niiden käyttö on vapaaehtoista ja niitä voi muokata käyttötarpeen mukaan.

Tiimi-portaalin hälytysten käsittely -toiminnon vaatimusmäärittely perustuu IEEE 830 -standardissa esitettyyn toimintoperusteiseen dokumenttimalliin (16, s. 23). Vaatimusmäärittely kuvaa kehitettävän ohjelman ulkoiset rajapinnat, toiminnot ja ominaisuudet, suoritus- ja toimintakykyvaatimukset, suunnittelun riippuvuudet ja

muut mahdolliset vaatimukset. Vaatimusmäärittelydokumentti on laadittu englanniksi, joka on yleinen ohjelmistokehityksessä käytetty kieli.

4.3 Vaatimusmäärittelyn tekeminen

Vaatimusmäärittelyn tekeminen aloitettiin kokoamalla opinnäytetyön alussa kerättyjä esitietoja kehitettävästä ohjelmasta. Esitiedot kerättiin asiakkaan kanssa käydyistä sähköpostikeskusteluista ja tulevan ohjelman toimintoa kuvaavista muistioista, jotka asiakas oli laatinut ohjelmointiongelman kuvaamista varten. Muistiot sisälsivät kehitettävän hälytystoiminnon toiminnallisen yleiskuvauksen ja toimintokaavion. Muistioiden teknisessä osassa oli esitelty hälytysviestien rakenne ja tietoja käsiteltävistä tiedoista. Tekninen osio sisälsi myös yleisiä vaatimuksia toteutustavasta, kuten käytettävä ohjelmointikieli ja virhetilanteiden käsittely.

Esitiedoista kerätyt tiedot kirjoitettiin vaatimusmäärittelydokumentin ensimmäiseksi versioksi, jäsentämällä dokumenttia standardipohjan mukaisesti. Ensimmäinen versio sisälsi kuvaukset kehitettävän ohjelman käyttötarkoituksesta, toimintaympäristöstä ja tärkeimmistä toiminnoista. Tavoitteena oli varmistaa, että työn tekijä oli ymmärtänyt oikein esitiedoissa kuvatun ohjelman tarkoituksen ja tehtävät.

Vaatimusmäärittely toimitettiin asiakkaalle etukäteen luettavaksi ja kommentoitavaksi. Tämän jälkeen määrittelyjä jatkettiin yhteisillä määrittelypalavereilla, joissa vaatimukset ja toiminnot käytiin yksityiskohtaisesti läpi, päivittäen määrittelydokumenttia samalla tarkemmaksi. Vaatimusmäärittelyn katsottiin olevan valmis, kun se sisälsi sekä työn tekijän että asiakkaan mielestä riittävän kattavan ja yksityiskohtaisen kuvauksen kehitettävän ohjelman toiminnoista ja vaatimuksista. Valmis vaatimusmäärittely sisälsi myös kuvauksen ohjelman toimintaympäristöstä, käytettävistä ohjelma- ja tiedonsiirtorajapinnoista ja yleisiä vaatimuksia suorituskyvystä, vikasietoisuudesta ja toimintavarmuudesta.

5 ANALYYSI JA SUUNNITTELU

Analyysi tarkoittaa ohjelmointiongelman ja vaatimusmäärittelyn tutkimista. Jos ohjelmaa tarvitaan, miten sitä on tarkoitus käyttää ja mitä toimintoja siinä pitää olla? Suunnittelu tarkoittaa käsitteellistä ohjelmointiongelman ratkaisua, joka täyttää asetetut vaatimukset. Analyysi ja suunnittelu voidaan tiivistää lauseeseen: ”tee oikea asia (analyysi) ja tee asia oikein (suunnittelu)”. (17, luku 1.3.)

5.1 Hyvän suunnittelun piirteitä

Hyvälle ohjelmistosuunnittelulle ominaisia piirteitä ovat: yksinkertaisuus, helppo ylläpidettävyys, laajennettavuus, uudelleenkäytettävyys, riippumattomuus, siirrettävyys ja kerrostuneisuus. (5, s. 74–81.)

Hyvä suunnitelma on ennemmin yksinkertainen ja helposti ymmärrettävä kuin kätevä tai fiksu. Helposti ylläpidettävä suunnitelma on itsensä selittävä, jolloin sitä on helppo lukea ja ymmärtää myöhemmin, kun ohjelmaa päivitetään mahdollisesti jonkun toisen ohjelmoijan toimesta. Hyvin suunniteltu ohjelma on laajennettavissa helposti, rikkomatta ohjelman rakennetta, tai jotain toista ohjelman komponenttia. Uudelleenkäytettävyys tarkoittaa, että ohjelman osia voidaan helposti käyttää jossain toisessa ohjelmassa. (5, s. 80–81.)

Hyvin suunniteltu ohjelma sisältää mahdollisimman vähän riippuvuuksia ohjelman eri osien välillä. Luokkien väliset sidokset ovat löyhiä ja niitä on vähän, luokan sisällä käytetään mahdollisimman vähän muita luokkia, eikä ohjelma sisällä ylimääräisiä osia. Tämä helpottaa ohjelma käyttöönottoa, testausta ja ylläpitoa. (5, s. 80.)

Suunnittelun kerrostuneisuus tarkoittaa, että katsottiinpa ohjelmaa järjestelmä-, paketti-, luokka- tai toimintotasolla, katsoja saa yhtenäisen ja johdonmukaisen kuvan suunnitelmasta. (5, s. 81.)

5.2 Analyysin tekeminen

Hälytystoiminto-ohjelman analyysi perustui tehtyyn vaatimusmäärittelyyn. Vaatimusmäärittely oli laadittu toimintoperusteisesti, kuvaamalla tärkeimmät toiminnot

ja niihin liittyvät vaatimukset. Lisäksi vaatimusmäärittelyssä oli kuvattu yleisellä tasolla ohjelmaympäristö ja käytettävät rajapinnat. Vaatimusmäärittelyyn oli sisällytetty lähdeviitteitä, joista saatiin lisätietoa ja tarkempia kuvauksia käytettävistä rajapinnoista, kuten tekstiviestien lähettämiseen tarkoitetusta SMS-palvelun rajapinnasta.

Vaatimusmäärittely sisälsi myös suorituskyky- ja laatuvaatimuksia, sekä rajoituksia suunnitteluun ja toteutukseen. Huomioitavia suunnitteluun liittyviä rajoituksia olivat seuraavat:

- Ohjelmointikielen on oltava PHP, käyttäen PHP-CLI SAPIa.
- Jokainen päätoiminto on toteutettava erillisenä PHP skriptinä.
- Jokainen päätoiminto on ajettava itsenäisenä palvelun, jota hallitaan systemd-järjestelmäpalvelulla.

Analyysivaiheessa vaatimusmäärittelystä tunnistettiin vaaditut toiminnot ja niihin liittyvät vaatimukset. Sanallisista kuvauksista etsittiin luokkia kuvaavia substantiiveja, niiden ominaisuuksia kuvaavia adjektiiveja ja metodeja kuvaavia verbejä. Analyysin aikana asiakkaalta kysyttiin tarkennuksia ja lisätietoa, jotta tulevasta ohjelmasta saatiin mahdollisimman selvä ja tarkka käsitys. Vaatimusmäärittelydokumentti päivitettiin ajan tasalle annettujen lisätietojen pohjalta.

Analyysin jälkeen ohjelmaa ryhdyttiin suunnittelemaan UML-mallinuksen avulla.

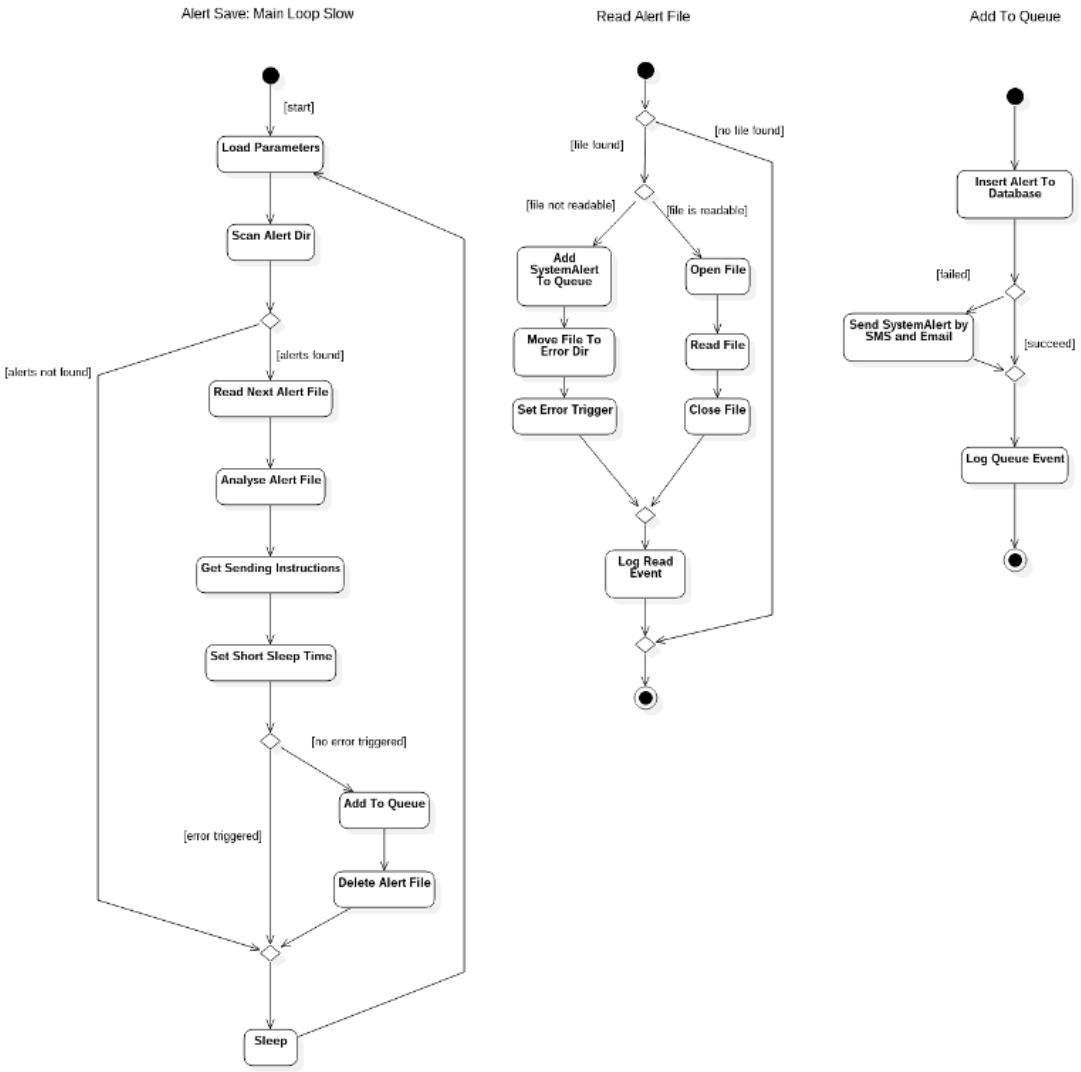
5.3 Suunnittelu UML-mallinnuksella

UML on pääosin graafinen mallinnus- eli kuvauskieli, jota käytetään suunnitelmien kuvaamiseen (18, s. 2). Graafisessa mallinnuksessa laaditaan kaavioita, jotka kuvaavat ohjelman luokkia, toimintoja, käyttötapauksia, vuorovaikutusta ja tilaa. UML-mallinnuksen avulla voidaan esittää käsitteitä selvemmin ja täsmällisemmin, kuin luonnollisella kielellä. Mallinnuskieli on kuitenkin yleistävää, eikä niin täsmällistä kuin varsinainen ohjelmointikieli. Sen avulla voidaan korostaa suunnittelun kannalta tärkeitä yksityiskohtia. (18, s. 7.)

Hälytysjärjestelmä oli kuvattu vaatimusmäärittelyssä toimintoperusteisesti, joten suunnittelu perustui myös ensisijaisesti toimintokaavioihin (activity diagram).

UML-mallinnuksen avulla ohjelman tärkeimmistä toiminnoista laadittiin toimintokaaviot yleisellä tasolla. Jos kaaviossa oleva toiminto koostui useammasta peräkkäisestä toiminnosta, kukin toiminto tulkittiin aliohjelmaksi, josta laadittiin oma mallinnuskaavio. Näin edeten toimintokaavioita laadittiin niin kauan, että kaikki kaavioiden toiminnot pystyttiin esittämään aliohjelman kaaviolla, tai ohjelmointikielen sisältämillä funktioilla. (Kuva 9.)

Kaaviossa toiminnot kuvattiin pyöristetyillä suorakulmioilla, joiden välinen nuoli kuvasi ohjelman kulkusuunnan. Valintatilanteet kuvattiin ruutukuviolla, jota seurasi nuolten jakautuminen vaihtoehtoisiin valintoihin. Valinnat kirjoitettiin nuolten viereen hakasuluilla rajatuiksi direktiiveiksi.



KUVA 9. Toiminto Alert Save, sen sisältämä aliohjelma Read Alert File ja pää- ja aliohjelman sisältämä aliohjelma Add to Queue

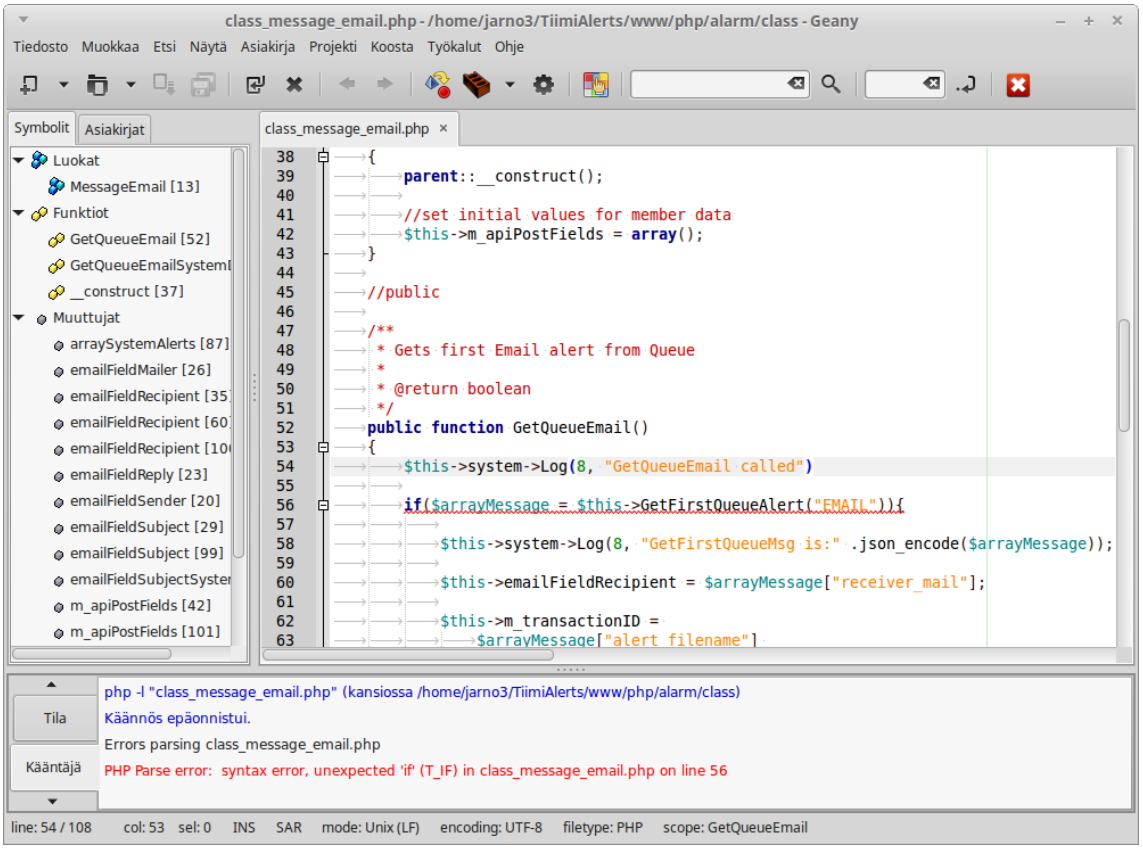
6 TOTEUTUS

6.1 Kehitysympäristö

PHP on tulkattava skriptikieli, jonka suorittaminen komentokehotteessa edellyttää PHP-CLI SAPI -rajapintaa. Vaatimusmäärittelyn perusteella oli tiedossa, että ohjelmaa käytettäisiin Debian Linux -palvelimella ja tietovarastona tulisi olemaan MariaDB-relaatiotietokanta. Sujuvan työn varmistamiseksi kehitysympäristöksi valittiin Debian-pohjainen Linux Mint -työasemakäyttöjärjestelmä, johon asennettiin natiivi MariaDB-tietokantaohjelma ja PHP-CLI SAPI.

Koodieditorina käytettiin Geany-tekstieditoria. Geany on kevyt lähdekoodieditori, joka tarjoaa pienikokoisen ja nopean IDE:n (Integrated Development, Environment) erityisesti web-ohjelmakehitykseen. Geany tukee PHP-skriptikieltä, sisältää lähdekoodin värikorostukset ja koodausta helpottavia automaattisia toimintoja. (19.)

Erityisen hyödyllinen Geanyn lisäominaisuus on editoriin integroitu bash-konsoli ja mahdollisuus suorittaa PHP-koodi suoraan editorissa. Tällöin koodausvaiheen testaus ja virheiden etsiminen tapahtuu nopeasti samassa IDE-ympäristössä (kuva 10).



KUVA 10. Geany IDE, PHP suoritusvirhe rivillä 56

6.2 Ohjelman rakenne

6.2.1 Suoritettavat toiminnot

Vaatimusmäärittelyn mukaan ohjelman toiminnot olivat

- saapuvien hälytysten käsittely
- hälytysviestien lähettäminen tekstiviestillä
- hälytysviestin lähettäminen sähköpostilla
- hälytysjonon valvonta
- hälytystietojen säilytys ja raportointi.

Vaatimuksen mukaan kunkin toiminnon oli toimittava itsenäisesti ajettavana palveluna.

Ohjelman rakenne vastasi vaatimusmäärittelyä. Se sisälsi yhteensä kuusi vaatimusten mukaisesti ”br_” etuliitteellä nimettyä, palveluna ajettavaa skriptiä:

1. **br_alarm_save**: saapuvien hälytysten käsittely
2. **br_alarm_sms**: hälytysviestien lähettäminen tekstiviestillä
3. **br_alarm_email**: hälytysviestien lähettäminen sähköpostilla
4. **br_alarm_queue**: hälytysjonon valvonta
5. **br_alarm_systemalert**: järjestelmähälytysten lähettäminen
6. **br_alarm_notifymailer**: valvontahälytysten lähettäminen.

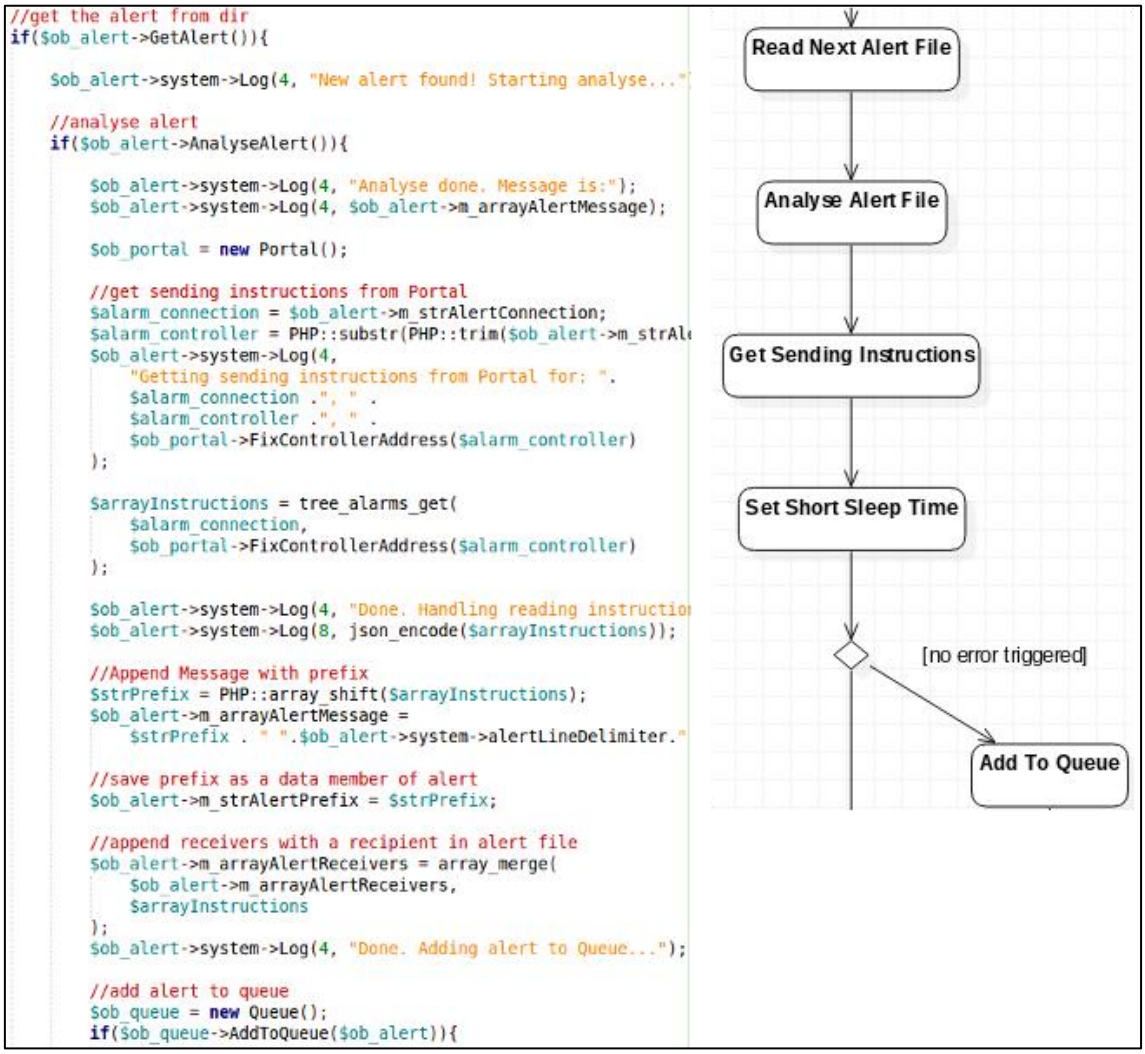
Kolme viimeksi mainittua toimintoa liittyvät kaikki hälytysjonon valvontaan. Toiminto jaettiin kolmeen eri palveluun suorittamistaajuuden perusteella: br_alarm_queue suoritetaan tunnin välein, br_alarm_systemalert suoritetaan 10 minuutin välein ja br_alarm_notifymailer suoritetaan tarvittaessa.

Kaikki toiminnot br_alarm_notifymaileria lukuun ottamatta kirjoitettiin PHP:llä. br_alarm_notifymailer on ylemmän tason systemd-palveluidenhallinnan käytössä. Se toteutettiin bash-komentoskriptinä, jolloin se toimii esimerkiksi silloin kun PHP-CLI SAPI ympäristö ei toimi, eikä PHP-koodia voida suorittaa.

Taustapalveluna suoritettavien toimintojen lisäksi, järjestelmä sisälsi selainkäyttöisen käyttöliittymän, jonka avulla ylläpitäjä voi seurata hälytysjonoa ja tehdä hälytysjonon hallintaan liittyviä toimia.

6.2.2 Ohjelmaluokat ja paketit

Toimintoja varten, mukaan lukien selainkäyttöliittymä, ohjelmaan tehtiin luokkakirjasto, josta muodostettuja olioita toiminnot käyttävät. Olio-ohjelmoinnin avulla toimintojen eteneminen oli helpompaa jäsentää ja koodista tuli helppolukuista ja pitkälti itsensä selittävää. Käsitteillä ja toiminnoilla oli ymmärrettävät nimet ja eteneminen oli loogista: hae hälytys (GetAlert), analysoi hälytys (AnalyseAlert), lisää hälytysjonoon (AddToQueue). Vastaavia käsitteitä ja toimintoja oli mietitty jo suunnitteluvaiheessa, joten lähdekoodia toimintokaavioon vertaamalla voitiin varmistua ohjelman toimivan suunnitellusti. (Kuva 11.)



KUVA 11. Koodinäyte br_alarm_save, vertailu UML-toimintokaavioon

Luokat ja niitä käyttävät toiminnot ryhmiteltiin suunnitteluvaiheessa paketeiksi. Paketointi on UML-mallinuksessa käytetty tapa ryhmitellä järjestelmän elementtejä, kuten luokkia, toisia paketteja ja käyttötapauksia. Paketoinnin avulla voidaan hahmottaa järjestelmän eri osien välisiä riippuvuuksia suuressa mittakaavassa. (17, s. 320.)

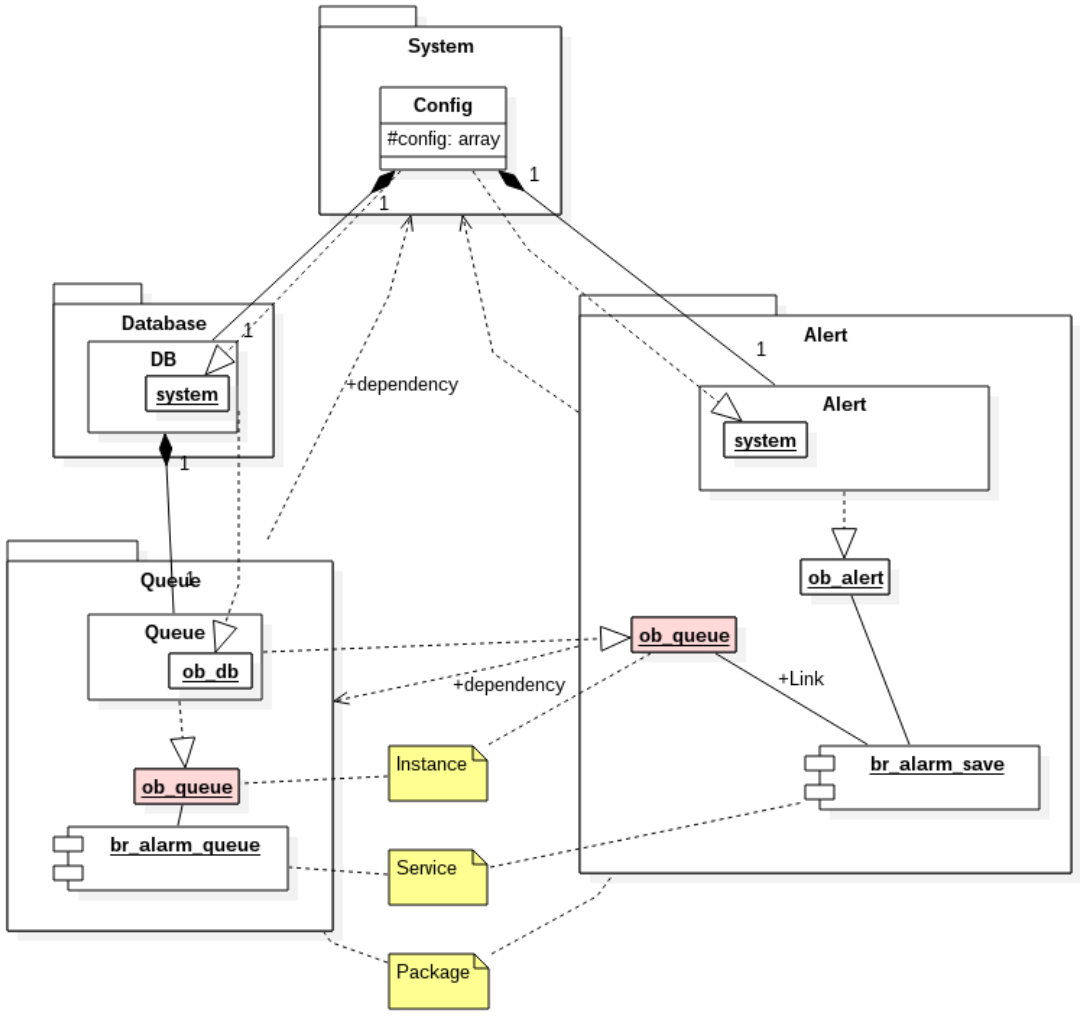
Tässä toteutuksessa paketointia ei hyödynnetty pelkästään suunnitteluun ja mallintamiseen, vaan paketoinnin avulla myös määritettiin toimintoja. Järjestelmän asetusparametrit sijoitettiin hierarkkiseen tietorakenteeseen, jossa pakettikohtaiset parametrit muodostavat oman ryhmänsä. Paketoinnin avulla rajattiin konfiguraatio siten, että luokalla on käytössään aina vain sen paketin parametrit, missä sen ilmentymä on. Tämä mahdollistaa sen, että olio voi toimia eri paketeissa eri

tavalla. Järjestelmässä tätä hyödynnettiin esimerkiksi tapahtumien kirjaamisessa lokitiedostoon. Saman luokan eri paketeissa sijaitsevat ilmentymät voivat kirjata tapahtumia eri tarkkuudella. Tästä on hyötyä, kun halutaan seurata tietyn ohjelmakokonaisuuden toimintaa tarkemmin. Asettamalla kirjaustaso korkeammaksi, saadaan kyseisen paketin tapahtumien seuranta informatiiviseksi, muiden tapahtumien häiritsemättä liikaa seurantaa. (Kuva 12.)

```
class Config
{
    protected $configDefault = array(
        "Alert"     =>  array(
            "alertDir"              =>  "/var/www/node/baserelay/d
            "errorDir"              =>  "/var/www/node/baserelay/e
            "alertFilePrefix"       =>  "SMS1_",
            "alertFileLines"        =>  3,
            "alertLineDelimiter"    =>  "/",
            "alertSleeptimeLong"    =>  10,
            "alertSleeptimeShort"   =>  1,
            "logLevel"              =>  1
        ),
        "Queue"     =>  array(
            "maxRetry"              =>  "5",
            "logLevel"              =>  4
        ),
        "Message"   =>  array(
            "sender"                =>  "        ",
            "messageSleeptimeLong"  =>  10,
            "messageSleeptimeShort" =>  1,
            "logLevel"              =>  1
        ),
        "Portal"    =>  array(
            "portalApiPath"         =>  "node_alarms_test.php",
            "logLevel"              =>  1
        ),
        "System"    =>  array(
            "emailAdminRecipient"   =>  "test@mail.example",
            "logLevel"              =>  8
        )
    );
}
```

KUVA 12. Hierarkkinen oletuskonfiguraatio paketeittain ryhmiteltynä. logLevel-parametrin arvo vaihtelee paketeittain

Esimerkiksi Queue-luokan ilmentymä ob_queue voi esiintyä sekä Queue-paketissa että Alert-paketissa. Esiintyessään Queue-paketissa lokikirjaus tapahtuu korkeammalla 4-tasolla, joka tarkoittaa informaatiotapahtumaa. Alert-paketissa esiintyessään saman luokan ilmentymä kirjaa lokiin alemmalla 1-tasolla, joka tarkoittaa virhetapahtumaa. Näin luokkaa ja sen ilmentymää voidaan seurata erityisesti tietyn toiminnon osalta. (Kuva 13.)



KUVA 13. Luokan ilmentymän käyttäminen eri paketeissa

Käytettäessä luokkien ilmentymiä eri paketeissa pakettien välille syntyy riippuvuuksia, jotka on huomioitava ohjelmoitaessa toimintoja. Elementin määritelmän muuttuminen toisessa paketissa voi aiheuttaa muutoksen myös toisessa paketissa (18, s. 96). Oli tärkeää tietää, mihin pakettiin luokan ilmentymä kului. Tämä

huomioitiin lisäämällä lähdekoodin alussa olevaan dokumentaatio-osaan merkintä paketista, johon toiminto tai luokka kuuluu. Merkintä tehtiin tarkoitukseen määritetyllä @package-tagilla. Oletuspaketti määritetään luokan konstruktorissa, mutta se on mahdollista asettaa myös olion luomisen jälkeen. Paketti määritetään käyttämällä System-luokan SetPackage-metodia.

6.2.3 Parametrit ja konfiguraatio

Edellä viitattiin järjestelmäparametrien sijoittamisesta hierarkkiseen konfiguraatio-tietorakenteeseen. Järjestelmän vaatimusmäärittelyssä oli konfiguraatioon liittyviä lisävaatimuksia:

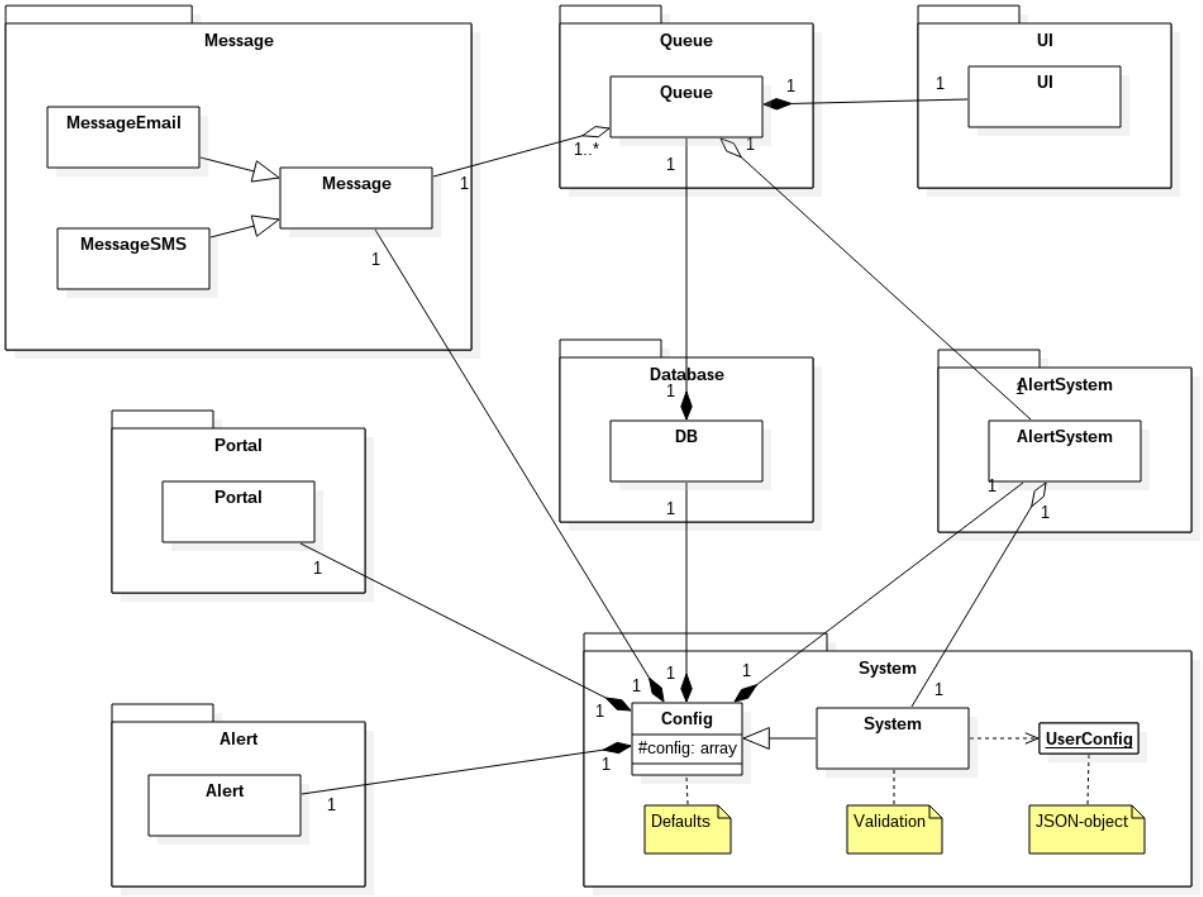
- Järjestelmäparametrit on sijoitettava erilliseen konfiguraatiotiedostoon.
- Järjestelmäparametrit on validoitava ennen käyttöönottoa.
- Järjestelmäparametrien oletusarvot on sisällytettävä lähdekoodiin.
- Järjestelmän on säilytettävä toiminnallisuus silloinkin, kun konfiguraatiotiedostoa ei ole käytettävissä tai se on virheellinen.
- Järjestelmäparametreja on voitava muuttaa, palveluita pysäyttämättä.

Järjestelmäparametrit sijoitettiin JSON-tietorakenteeseen ja tallennettiin erilliseen tekstitiedostoon. JSON (JavaScript Object Notation) on kevyt tietorakenne, jota käytetään usein eri järjestelmien väliseen tietojen vaihtoon. Se on ohjelmointikielestä riippumaton tapa esittää tietoa, jota on helppo lukea sellaisenaankin (21). JSON-tietorakenne on määritetty ECMA-404-standardissa (22).

JSON-tietorakenteeseen tallennetun konfiguraation rakenne vastaa edellä kuvattua oletuskonfiguraatiota (kuva 12). Siinä on sama hierarkkisuus ja parametrit paketoitu vastaaviin ryhmiin. JSON-konfiguraatio on järjestelmän ensisijainen konfiguraatio. Siihen tallennetut parametrien arvot ovat käytössä, jos ne täyttävät parametrille asetetut validointiehdot. Jos ehdot eivät täyty, järjestelmä käyttää oletusparametria.

Konfiguraation validointi on kaksivaiheinen: ensin validoidaan tiedoston tietorakenteen oikeellisuus, sitten validoidaan parametri. Jos konfiguraatiotiedoston tietorakenne ei ole validi JSON-objekti, sitä ei käytetä, vaan oletuskonfiguraation

astuu voimaan kokonaisuudessaan. Validin JSON-konfiguraation jokainen parametri validoidaan aina sitä kutsuttaessa. Validointi tapahtuu parametrikohtaisesti, joten ajon aikainen konfiguraatio voi sisältää parametreja sekä JSON-konfiguraatiosta että oletuskonfiguraatiosta. Parametrit validoidaan System-luokan Validate-metodissa. Tästä seuraa, että kaikilla järjestelmän luokilla on vahva sidos, eli kompositio, System-luokkaan joko suoraan, tai jonkun toisen luokan kautta. (Kuva 14.)



KUVA 14. Luokkien väliset sidokset ja konfiguraation validointi

6.2.4 Ulkoiset tietolähteet ja palvelut

Sovelluksen tuli olla osa laajempaa Tiimi-portaalin kokonaisuutta, joten sen tuli voida käyttää ulkopuolisia tietolähteitä ja palveluita. Sovelluksen koko toiminta perustuu BaseRelay-palvelinsovelluksen välittämiin hälytystietoihin, jotka välitetään sovellukselle määrämuotoisena tekstitiedostona.

Portaalin tietokanta sisältää hälytystietoihin liittyviä lähetysohjeita, kuten vastaanottajan puhelinnumero ja sähköpostiosoite. Sovelluksen oli käytettävä tietojen hakemiseen portaalisovelluksen tarjoamaa valmista funktiota, joka ottaa parametreikseen hälyttävän laitteen yhteys- ja laiteosoitteet, ja palauttaa hälytysviestin välittämiseen tarvittavat lisätiedot. Funktio oli kirjoitettu PHP-kielellä, joten se voitiin ottaa hälytysjärjestelmän käyttöön sellaisenaan.

Palvelimella oli käytössä ylläpitäjän määrittämä lokikäytäntö. Tapahtumalokiin kirjattiin tapahtumia eri tasoilla, jotka ovat: virhe, varoitus, informaatio ja testaus (debug). Palvelin tarjosi lokitiedostojen käyttöön valmiin funktion, jota sovellus käytti osana tapahtumatietojen tallennusta. Käyttämällä valmista funktiota, varmistettiin tapahtumalokien yhdenmukaisuus eri järjestelmien välillä.

Järjestelmän tuli lähettää hälytysviestejä tekstiviesteinä. Määrittelyvaiheessa tekstiviestipalveluksi oli valittu kolmannen osapuolen palvelu, joka tarjosi valmiin JSON REST -rajapinnan tekstiviestien lähettämiseen. Tekstiviestin tiedot lähetetään palveluntarjoajan rajapintaan JSON-objektina HTTP-protokollaa käyttäen. Palvelussa tiedoista muodostetaan tekstiviesti, joka lähetetään edelleen tiedoissa mainittuun puhelinnumeroon.

6.3 Ohjelmointikäytännöt

6.3.1 Lähdekoodin dokumentointi

Hyvä lähdekoodin kommentointi parantaa huomattavasti koodin luettavuutta, kun taas huonolla kommentoinnilla luettavuus vain heikkenee (5, s. 763). Lähdekoodin kommentoinnissa noudatettiin seuraavia periaatteita:

- Kommentointi sisennetään samalla tavalla kuin kommentoitava lähdekoodi.
- Kommentin yläpuolella on aina tyhjä rivi.
- Kommentti pm kirjoitettu aina omalle riville.
- Lyhyttä kommentointimerkintää (//) käytetään selittämään, mitä kyseisessä koodin kohdassa on tarkoitus tehdä (kuva 15).
- Pitkää kommentointimerkintää (/* */) käytetään koodin dokumentoimiseen.
- Pitkän kommentin aloitusmerkki sisältää aina ylimääräisen *-merkin.

- Jokainen kooditiedosto, luokka, luokan datajäsen ja metodi on dokumentoitu pitkällä kommenttimerkinnällä (kuva 16).
- Pitkä kommentti jäsennetään tageilla, jotka tunnistetaan @-merkistä.
- Pitkä kommentointi noudattaa phpDocumentor DocBlock merkitsemiskäytäntöjä (23).

```php
// scan dir and get the first file
$arrayAllFiles = PHP::scandir($this->system->alertDir);
if($this->m_strAlertFilename = $this->FirstFile($arrayAllFiles)){
    $this->system->Log(8, "Alert File Found");
    return true;
}
else {
    $this->system->Log(8, "Alert File Not Found");
    return false;
}
```

KUVA 15. Koodin toimintaa selitetään lyhyellä kommentilla

```php
/**
 * Summary:     Class for an incoming alert.
 *
 * @package     Alert
 * @author      Mattila, Jarno
 * @since       2018-09-26  Initial version
 * @since       2018-10-13  Logger implementation
 */

class Alert
{
    /**
     * member variables
     */

    /** @var object $system Main system */
    private $system;

    /** @var array $m_arrayAlertReceivers Receiver list */
    protected $m_arrayAlertReceivers = array();
```

KUVA 16. Luokan ja datajäsenten dokumentointi pitkällä kommentilla

6.3.2 PHP-versiomuutosten hallinta apuluokan avulla

Ohjelman toiminnot testattiin ohjelmointivaiheessa tuetuilla PHP:n versioilla 5.6, 7.0 ja 7.2. Versio 7.2 on aktiivisen tuen piirissä, eli sitä päivitetään säännöllisesti. Versioiden 5.6 ja 7.0 tuki on voimassa ainoastaan havaittujen tietoturva-aukkojen korjauksiin liittyvien päivitysten osalta, ja tuki päättyy kokonaisuudessaan vuoden 2018 loppuun mennessä (taulukko 2).

TAULUKKO 2. Tuetut PHP-versiot 28.10.2018 (24)

Versiohaara	Julkaisupäivä	Aktiivinen tuki päättyy	Tietoturvatuki päättyy
5.6	28.8.2014	19.1.2017	31.12.2018
7.0	3.12.2015	3.12.2017	3.12.2018
7.1	1.12.2016	1.12.2018	1.12.2019
7.2	30.11.2017	30.11.2019	30.11.2020

Taulukosta havaitaan, että uusi PHP-versio julkaistaan säännöllisesti vuosittain. Kunkin version aktiivinen tuki on kaksi vuotta, jonka jälkeen versiota tuetaan vielä vuoden ajan tietoturvaan liittyvien päivitysten osalta. Poikkeuksen muodostaa versio 5.6, jonka tuki on kuitenkin myös päättymässä.

PHP-kieleen tulee päivityksissä uusia ominaisuuksia ja kieli laajenee. Samalla kuitenkin ominaisuuksia myös poistuu ja niiden käyttö muuttuu. (25.)

Kehitettävän ohjelman elinkaari tulisi olemaan pidempi kuin yhden PHP-version elinkaari, joten oli perusteltua varautua tuleviin versiomuutoksiin. Oli myös mahdollista, että ohjelman käyttöympäristö muuttuu vaatimusmäärittelyssä kuvatusta Debian-palvelimesta, joten ohjelman oli oltava alaspäin yhteensopiva valmistushetkellä tuettuihin versioihin. Ohjelman toimintavarmuutta PHP-kielen muutoksia vastaan parannettiin PHP-apuluokalla, jonka avulla PHP-tulkin valmiit funktiot voitiin suorittaa hallitusti staattisten metodien sisällä.

Metodit nimettiin funktioita vastaavilla nimillä. Ohjelmakoodissa metodia kutsutaan samoilla parametreillä ja samalla nimellä kuin alkuperäistä funktiota. Esimerkiksi funktiokutsun *dirname("/path/to/file.txt")* sijaan kutsutaan staattista metodia *PHP::dirname("/path/to/file.txt").* Luokan metodin sisällä suoritetaan varsinainen funktiokutsu, ja palautetaan funktion palauttama arvo.

Funktion käytettävyys tarkistetaan aina ennen sen kutsua, ja lisäksi voidaan tarkistaa myös käytettävä PHP-versio. Jos funktio on käytettävissä ja PHP-versio on sopiva, funktio suoritetaan sellaisenaan. Tarvittaessa parametrien arvoja voidaan käsitellä ennen funktiokutsua, tai voidaan kutsua jotain toista funktiota, jonka tiedetään suorittavan vaadittu tehtävä paremmin. Joissain tapauksissa koko funktio voidaan korvata omalla funktiolla. (Kuva 17.)

```
/**
 * string dirname ( string $path [, int $levels = 1 ] )
 */
public static function dirname($path, $levels = 1)
{
    //workaround for < 7 versions
    $strNewPath = "";
    $arrayDirs = PHP::explode("/", $path);
    for($i = 1; $i < count($arrayDirs) - $levels; $i++){
        $strNewPath .= "/" . $arrayDirs[$i];
    }
    if(function_exists("dirname")){

        if(PHP::version_compare(PHP_VERSION, '7.0.0.', '>=')){

            return dirname($path, $levels);
        }
        else {

            return $strNewPath;
        }

    } else {

        return $strNewPath;
    }
}
```

KUVA 17. dirname-funktion vaihtoehtoinen käyttö eri PHP-versioilla

Ohjelman sisältämä PHP-apuluokka tuli sisältämään yli 30 staattista metodia, joita käytettiin eri puolilla varsinaista ohjelman lähdekoodia. Käyttämällä apuluokan metodeja suorien funktiokutsujen sijaan, ylläpitäjä voi hallita keskitetysti mahdolliset tulevat versiopäivitysten aiheuttamat muutokset funktioiden toiminnassa. Metodit lueteltiin luokan lähdekoodin alussa olevassa dokumentaatiossa, josta ohjelmoija voi nopeasti tarkistaa voiko tiettyä funktiota kutsua luokan metodilla. (Kuva 18.)

```
<?php
/**
 * Summary:     Helper class to handle php legacy issues.
 *
 * Description: Original PHP function is documented above a method
 *              with the same name. If original function is gone
 *              (deprecated) it can be replaced here by a new one,
 *              without replacing it all over the source code.
 *
 * @package     Helper
 * @author      Mattila, Jarno
 * @since       2018-10-26: In alphabetic order:
 *              array_pop, array_shift, basename,
 *              ctype_digit, ctype_print, curl_close, curl_exec,
 *              curl_init, curl_setopt, date, date_default_timezone_set,
 *              dirname, explode, file, file_exists, file_get_contents,
 *              filter_var, intval, is_dir, json_decode, json_encode,
 *              mail, preg_match, rename, scandir, sleep, strlen,
 *              substr,  time, trim, unlink, version_compare
 */

class PHP
{
```

KUVA 18. PHP-apuluokan dokumentaatio

6.3.3 Nimeämiskäytäntö

Hyvä yhtenäinen nimeämiskäytäntö on tärkeä osa lähdekoodin luettavuutta. Muuttujien, luokkien ja metodien nimien tulee olla niin täsmällisiä ja kuvaavia kuin mahdollista. Epämääräinen tai ympäripyöreä nimi, joka soveltuu käytettäväksi useampaan kuin yhteen tarkoitukseen, ei ole hyvä. Hyvä nimeämiskäytäntö myös erottelee lähdekoodin eri rakenteita. Käyttämällä esimerkiksi muuttujan ja metodin nimeämisessä erilaista kirjoitusasua, helpotetaan eri asiayhteyksien jäsentämistä lähdekoodissa. (5, s. 289.)

PHP-kielessä muuttujan nimi alkaa aina dollarimerkillä ($) ja sitä seuraavalla kirjaimella tai alaviivalla (26). Muuttujan tietotyyppiä ei ilmoiteta muuttujaa esiteltäessä, vaan tietotyyppi määräytyy muuttujan käyttötarkoituksen ja arvon mukaan (27).

Järjestelmän lähdekoodin nimeämiskäytännössä tavoitteena oli mahdollisimman tarkasti kutakin asiaa kuvaava nimi ja rakenteellinen yhdenmukaisuus. Luokan sisällä olevat muuttujat nimettiin yksilöllisesti. Nimi alkaa aina pienellä kirjaimella, mutta jos nimi koostuu useista sanoista, alkusanaa seuraavat sanat alkavat isolla kirjaimella eikä sanojen välissä käytetä välimerkkejä. Tietotyypin puuttuminen ratkaistiin lisäämällä muuttujan nimen alkuun sen tietotyyppiä kuvaava sana. Eri tyyppisten muuttujien nimiä ovat esimerkiksi

- $strAlertFile (merkkijonomuuttuja)
- $boolValidationResult (boolean-muuttuja)
- $intSegment (kokonaislukumuuttuja)
- $arrayAddressSegments (taulukkomuuttuja)
- $mixDatavalue (tietotyyppi vaihtelee).

Luokan ja metodin nimi alkaa aina isolla kirjaimella. Muuten nimeämiskäytäntö oli sama kuin muuttujissa, kuitenkin niin että tietotyyppiä ei ilmoiteta nimen alussa, vaan se ilmoitetaan metodin dokumentaatiossa @return tagilla merkattuna. Luokan ja metodien nimiä ovat esimerkiksi Queue (luokan nimi) ja AddToQueue (metodin nimi).

Ohjelmoitaessa eri toimintoja, muuttujien nimeämiskäytäntö saattoi poiketa edellä kuvatusta. Nimissä voitiin käyttää myös pelkästään pieniä kirjaimia ja sanat voitiin erotella alaviivaa käyttäen, esimerkiksi $ob_alert (olio) tai $alarm_connection (merkkijono). Tavoitteena oli näissäkin tapauksissa kuvaava ja yksilöllinen nimi.

6.4 Ohjelmointityön eteneminen

6.4.1 Tekstiviestin lähettäminen

Ensimmäinen ohjelmointitehtävä oli valitun tekstiviestipalvelun testaus. Testaus haluttiin tehdä heti aluksi, jotta varmistuttaisiin ulkopuolisen palvelun laadusta ja toimivuudesta. Tekstiviestipalvelun tarjoaja saattoi siis olla kyseinen palveluntarjoaja tai mahdollisesti joku muu. Lisäksi vaatimusmäärittelyn perusteella oli tiedossa, että tekstiviestien lisäksi sovelluksen on voitava lähettää viestejä myös sähköpostilla.

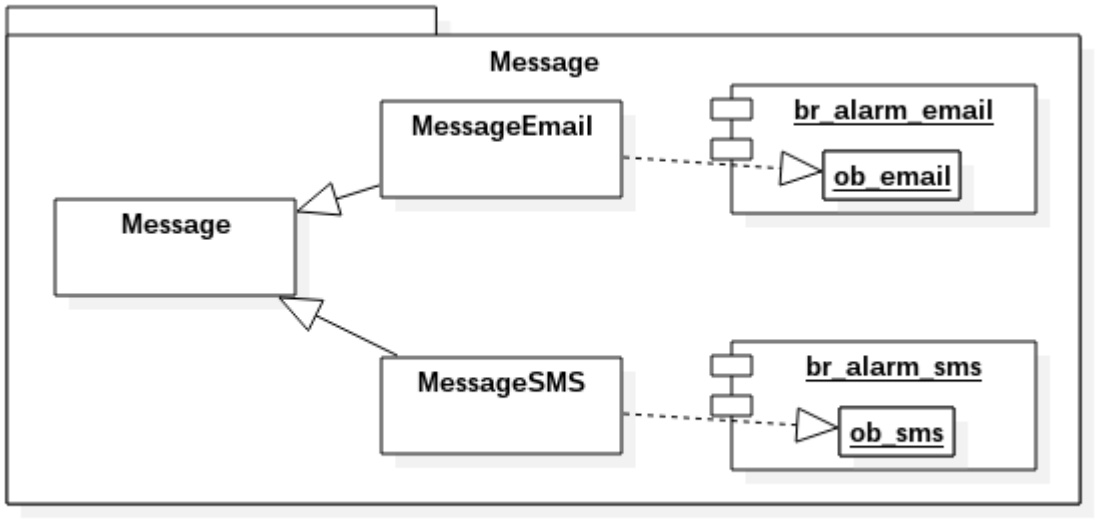
Viestejä varten päätettiin tehdä yleinen viestiluokka Message, joka sisälsi yleiset viestin ominaisuudet ja toiminnot, kuten vastaanottajatiedot ja lähetysmetodit (kuva 19).

```php
<?php
/**
 * Summary:     Abstract class for an outgoing alert message.
 *
 * @package     Message
 * @author      Mattila, Jarno
 * @since       2018-09-19 Initial version
 * @since       2018-09-20 Replaced separate endpoint methods with
 *              general SendMessage mehtod. Added m_sendLock to
 *              prevent sending if not allowed
 * @since       2018-10-13  Logger implementation
 */

abstract class Message
{
```

KUVA 19. Yleisen Message-luokan määrittäminen abstract määritteellä

Yleisestä eli abstraktista luokasta ei voi muodostaa oliota, vaan luokka periytetään lapsiluokalle, josta varsinainen olio muodostetaan. Abstrakti luokka voi sisältää myös abstrakteja metodeja, jotka esitellään abstraktissa luokassa, mutta määritellään lapsiluokassa. Jos luokka sisältää yhdenkään abstraktin metodin, myös luokan on oltava abstrakti. (28.)

Tekstiviestin tietosisältö on yleisesti ottaen sama, palvelun tarjoajasta riippumatta, joten lähetysmetodi sisällytettiin abstraktiin luokkaan, mutta metodi itsessään ei ollut abstrakti. Näin lapsiluokan koodia saatiin yksinkertaistettua, sisällyttämällä siihen vain valitun palveluntarjoajan palveluun liittyvät ominaisuudet ja metodit. Lapsiluokan lisäksi ohjelmointiin toinen lapsiluokka sähköpostien lähettämistä varten. Luokat yhdessä niitä käyttävien pääohjelmien kanssa muodostivat Message-paketin, jolla ohjelma välittää kaikki hälytysviestit vastaanottajille. (Kuva 20.)



KUVA 20. Message-paketin luokat, oliot ja pääohjelmat

Tekstiviestien lähettäminen testattiin kirjoittamalla lyhyt driver-ohjelma ja sen käyttöön tynkäluokka (stub). Stub-ohjelmaa käytetään korvaamaan alemman tason komponentti testattaessa ylemmän tason toimintoa, driver on ylemmän tason ohjelma, jolla testataan alemman tason komponenttia (29). Driver ja stub ovat ylimääräisiä ohjelma-aihioita, joita ei sisällytetä lopulliseen ohjelmaan. Ylimääräisten ohjelmien kirjoittaminen lisää ohjelmointityötä, joten ne kirjoitetaan nopeasti ja mahdollisimman yksinkertaisesti, mutta kuitenkin niin, että haluttua toimintoa voidaan testata ja arvioida riittävän luotettavasti (30, s. 487).

Testiä varten tehty luokka sisälsi setParameter-metodin, jolla olioon tallennettiin yhteystestissä tarvittavat tiedot: käyttäjätunnus ja salasana. Driver-ohjelma sisälsi toiminnon, joka teki yksinkertaisen CreditCheck-kyselyn SMS-palveluun ja tulosti vastaanotetut tiedot konsoliin. (Kuva 21.)

```php
<?php

$server = " ";
$username = " ";
$password = " ";
$endpoint = "/CheckCredits";

$ob_sms = new MessageSMS($server, $username, $password, $endpoint);
$ob_sms->setParameter("UserName", $username );
$ob_sms->setParameter("Password", $password);
echo json_encode($ob_sms->parameters, JSON_FORCE_OBJECT).PHP_EOL;
echo $ob_sms->api_url.PHP_EOL;
?>
```

KUVA 21. SMS-Driver tulostaa lähetystiedot ja vastaanotetun palautteen konsoliin

Testausta jatkettiin lisäämällä driver-ohjelmaan tekstiviestin vastaanottaja ja lyhyt viestiosa. SMS-palvelu toimi kaikilla testauskerroilla moitteettomasti, joten se valittiin järjestelmän tekstiviestipalveluksi. Lisäarvona testauksesta saatiin idea toiminnosta, jolla palvelun saldo tarkistetaan CreditCheck-kyselyllä aina ennen varsinaista tekstiviestin lähettämistä. Jos saldo on alle asetetun miniarvon, viestiä ei lähetetä ja järjestelmä generoi error-tason lokimerkinnän ja järjestelmähälytyksen.

Tekstiviestiin liittyvät luokat ja pääohjelma kirjoitettiin valmiiksi. Pääohjelma on nimeltään br_alarm_sms. Toiminto tarkistaa hälytysjonoon tulleet viestit 10 sekunnin välein. Jos jonossa on lähettämätön tekstiviestihälytys, toiminto lähettää hälytyksen ja tarkistaa sen jälkeen jonon uudelleen heti sekunnin kuluttua edellisestä lähetyksestä.

Ohjelman käyttämien luokkien metodit ovat:

- GetQueueSMS: julkinen metodi, jota pääohjelma käyttää. Metodi hakee yhden hälytyksen hälytysjonosta, tarkistaa SMS-palvelun lähetyssaldon, muodostaa hälytyksen tiedoista tekstiviestin ja tallentaa tiedot luokan datajäsenten arvoiksi.

- GetFirstQueueAlert: peritty metodi (protected), jota GetQueueSMS käyttää viestin hakuun hälytysjonosta. Metodi hakee viestin käyttämällä Queue-luokan GetAlertToSend-metodia
- CheckCredits: privaatti metodi, jota GetQueueSMS käyttää. Metodi palauttaa käytettävissä olevan viestisaldon.
- SendMessage: peritty metodi (public), jota pääohjelma käyttää, jos GetQueueSMS on palauttanut lähetettävän hälytyksen. Metodi suorittaa varsinaisen viestin lähettämisen SMS-palveluun.
- UpdateDelivered: peritty metodi (public), jonka avulla pääohjelma asettaa lähetetyn viestin lähetetty -tilaan. Asettaminen tapahtuu tallentamalla aikaleima tietokantaa, jolloin lähetysaika tallentuu viestin tietoihin.
- AddRetryValue: peritty metodi (public), jonka avulla pääohjelma kasvattaa lähetyslaskurin arvoa, jos lähetys epäonnistuu.

6.4.2 Sähköpostin lähettäminen

PHP-kieli tarjoaa sähköpostin lähettämiseen valmiin mail-funktion. Funktio käyttää paikallista sendmail-yhteensopivaa MTA:ta viestin lähettämiseen. Kohdepalvelimella oli käytössä Postfix MTA, joten postin lähettäminen voitiin toteuttaa mail-funktiolla. (31.)

Järjestelmän täytyi lähettää kahdenlaisia hälytyksiä sähköpostilla: laitehälytyksiä (varsinainen toiminto) ja järjestelmähälytyksiä (virhetilanteet). MessageEmail-lapsiluokkaan kirjoitettiin omat metodit kumpaakin tapausta varten. Metodit hakevat hälytysjonosta edellä mainittuja viestejä ja välittävät ne SendEmail-metodille, joka suorittaa varsinaisen viesti lähettämisen. Toiminnot testattiin ohjelmointivaiheessa, lähettämällä testiviestejä eri sähköposteihin. Testauksen perusteella viestiotsikoihin lisättiin utf8-merkistokoodaus, joka osoittautui testauksessa luotettavimmaksi tavaksi esittää skandinaaviset kirjaimet eri sähköpostiohjelmissa.

Sähköposti-toiminnon pääohjelma on nimeltään br_alarm_email, jonka tehtävä on lähettää laitehälytyksiä. Jonon tarkistusväli on sama kuin br_alarm_sms-ohjelmalla. Ohjelma käyttää jonon tarkistukseen GetQueueEmail-metodia, jonka toimintaperiaate on sama kuin GetQueueSMS-metodilla. Lähetyskuittaus ja lähetyslaskurin päivitys tapahtuvat samalla tavalla kuin tekstiviestin lähetyksessä.

Toinen sähköpostihälytyksiä lähettävä toiminto on br_alarm_systemalert. Sen tehtävänä on lähettää ilmoitusviestejä ylläpitäjälle, jos järjestelmässä tapahtuu virhe. Toiminnon ensimmäisessä versiossa virhe lähetettiin sähköpostilla heti kun se havaittiin. Lähetyksessä hyödynnettiin tapahtumalokitoimintoa, jossa alimman tason (error) lokimerkinnät generoivat aina ilmoituksen. Käytännössä tällainen toiminto kuitenkin helposti täyttää ylläpitäjän sähköpostin toistuvilla virheviesteillä. Lisäksi vaatimusmäärittelyssä vaatimuksena oli mainittu, että virheviestit lähetetään 10 minuutin välein, joten toimintoa täytyi kehittää edelleen.

Toiminto toteutettiin siten, että virhelokin generoimat viestit tallennettiin hälytysjonoon SystemAlert-tunnisteella merkittynä. SystemAlert-hälytyksiä käsittelevä br_alarm_systemalert-ohjelma tarkistaa jonosta SystemAlert-viestit 10 minuutin välein, ja koostaa kaikista virheistä yhden viestin (digest), jonka lähettää ylläpitäjälle (kuva 22).

KUVA 22. SystemAlert-sähköpostiviesti: validointivirhe. Esiintyy jonossa 1 kertaa

Valitulla ratkaisulla saatiin järjestelmähälytysten ilmoitustaajuus vaatimusmäärittelyn mukaiseksi. Kokoamalla kaikki järjestelmähälytykset yhteen viestiin, saatiin koko informaatio toimitettua ylläpitäjälle yhdessä viestissä.

SystemAlert-viestejä ei kuitata lähetetyksi ohjelman toimesta, vaan viestejä lähetetään 10 min välein, kunnes ylläpitäjä poistaa SystemAlert-tietueet hälytysjonosta.

6.4.3 Hälytysviestien käsittely

Hälytysten käsittely ohjelmoitiin br_alarm_save-pääohjelmaan. Ohjelma tarkistaa saapuneet hälytykset 10 sekunnin välein. Ohjelma käsittelee vain yhden tiedoston kerrallaan, mutta saapuneen hälytyksen käsittelyn jälkeen, seuraava tarkistus tehdään yhden sekunnin kuluttua. Toiminto siis käsittelee hälytyksiä nopeutetulla 1 sekunnin taajuudella ja toimii muutoin hitaalla 10 sekunnin taajuudella.

Toiminto käyttää Alert-, Portal- ja Queue-luokkia, joista Alert-luokka sisältää varsinaiset hälytysten käsittelyyn tarvittavat metodit. Saapuvien hälytysviestien käsittelyyn tarvittavat Alert-luokan metodit ovat seuraavat:

- FirstFile: privaatti metodi, joka palauttaa hälytyskansiosta yhden hälytystiedoston, joka on nimetty vaatimusmäärittelyssä määritellyllä tavalla.
- GetAlert: julkinen metodi, jota pääohjelma kutsuu. Metodi tallentaa FirstFile-metodin palauttamat tiedot luokan datajäseniksi olion käyttöön.
- AnalyseAlert: julkinen metodi, jota pääohjelma kutsuu, jos GetAlert ilmaisee uuden hälytyksen (paluuarvo true). Metodi tarkistaa hälytyksen pakolliset tiedot ja hyväksyy tai hylkää hälytyksen analyysin perusteella. Hylätty tiedosto siirretään virhekansioon, ja tapahtumasta muodostetaan error-lokimerkintä ja järjestelmähälytys.
- FixNewLine: privaatti metodi, jota AnalyseAlert kutsuu. Metodi korjaa erityyppisten rivinvaihtojen aiheuttaman ongelman.
- ValidateField: privaatti metodi, joka validoi analysoitavan hälytystiedoston, tarkistamalla yhteys- ja laiteosoitteiden rakenteen.
- RemoveAlert: julkinen metodi, jota pääohjelma kutsuu. Metodi poistaa hälytystiedoston, kun hälytys on käsitelty ja siirretty lähetysjonoon.

Toiminto hakee Portaalin tietokannasta hälytyksen lähetysohjeet. Ohjeet haetaan käyttämällä valmista funktiota, jolle viedään parametrina tarvittavat osoitetiedot. Ohjelma alustaa parametrit oikeaan muotoon, käyttäen apuna Portal-luokan

FixConnectionAddress-metodia, joka korjaa osoitetiedoissa olevat ylimääräiset etunollat (zero-fill).

Validi viesti, joka sisältää tarvittavat lähetysohjeet, tallennetaan hälytysjonoon. Hälytysjono on tietokantataulu, jonka käsittelyyn tarvittavat SQL-lausekkeet sijaitsevat Queue-luokassa. Uuden hälytyksen lisääminen jonoon tapahtuu Queue-luokan AddToQueue-metodilla. Jos metodi palauttaa true-arvon, tallennus on onnistunut ja hälytystiedosto voidaan poistaa.

Hälytysviestien käsittelytoiminto oli järjestelmän toiminnoista monimutkaisin toteuttaa. Toiminto sisälsi esimerkiksi tiedoston käsittelyä, merkkijonojen käsittelyä ja muokkausta ja tietojen hakemista ulkopuolisesta lähteestä.

Lisähaastetta tehtävään toi hälytystiedoston vaihteleva rakenne ja saman tiedon esittäminen eri muodossa eri tietolähteissä. Ohjelman täytyi lukea laiteosoite hälytystiedoston kolmannen rivin lopusta, mutta rivin pituus ja rakenne saattoivat vaihdella. Monimutkaisten validointisääntöjen lisäksi, tiedoston käsittely vaati kolmen aputoiminnon ohjelmointia:

1. FixNewLine-metodi tarkistaa tiedoston rivinvaihdon merkin. Rivinvaihtoja voi olla kolmenlaisia: Unix-tyyppinen line feed (LF), Mac-tyyppinen carriage-return (CR), tai Windows-tyyppinen yhdistelmä CR/LF. Metodi tarkistaa, onko rivien lukeminen ja taulukointi onnistunut LF-rivinvaihdolla ja jos ei ole, niin se taulukoi rivit CR-rivinvaihdon perusteella.
2. FixAlertReceiver-metodi poimii viestin toiselta riviltä vastaanottajan puhelinnumeron. Rivi sisältää numeron lisäksi modeemin alustuskomentoja (AT), jotka täytyy erottaa varsinaisesta puhelinnumerosta.
3. FixControllerAddress-metodi poistaa laiteosoitteesta ylimääräiset etunollat, jotta se toimii hakukriteerinä lähetysohjeita haettaessa. Esimerkiksi osoite 101.111.222.001 on oltava muodossa 101.111.222.1.

Laiteosoite muistuttaa IPv4-osoitetta. Siinä on neljä 8-bittistä kokonaislukua pisteellä eroteltuna. Kyseessä ei kuitenkaan ole IP-protokollan osoite, joten osoitetta ei voitu validoida IP-osoitteen tavoin, vaan jokainen osoitesegmentti validoitiin

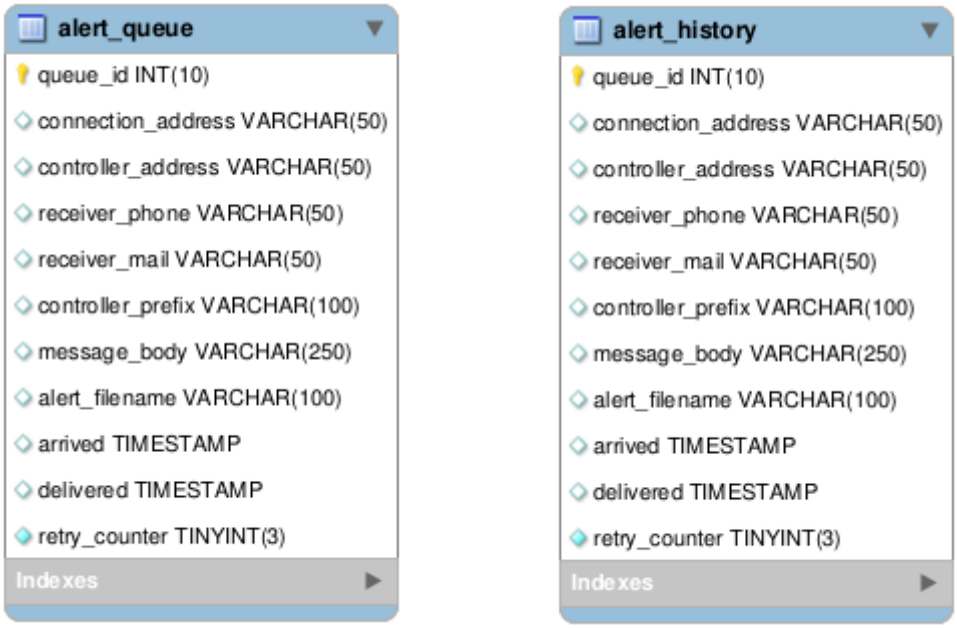
vaatimusmäärittelyn mukaisesti 8-bittiseksi kokonaisluvuksi, jonka desimaalimuotoinen arvo voi vaihdella välillä 0-255. (Kuva 23.)

```
//set validation trigger as true
$boolValidationResult = true;

//check columns by validating 8-bit integer value
foreach($arrayAddressSegments as $srtSegment){

    //validate numeric digits
    $boolFilterDigit =  PHP::preg_match('/^\d+$/', $srtSegment);

    //validate 8-bit integer
    $intSegment = PHP::intval($srtSegment);

    $boolFilterInt =  PHP::filter_var(
        $intSegment, FILTER_VALIDATE_INT, array(
            "options" => array(
                "min_range" => 0,
                "max_range" => 255
            )
        )
    ) !== false && $boolFilterDigit === 1;

    $this->system->Log(8,
        "Validation of " . $srtSegment . " = " .
        $boolFilterDigit . ":" .
        $intSegment . ":" .
        $boolFilterInt
    );
    //validate only if last validation is passed
    $boolValidationResult = $boolFilterInt && $boolValidationResult;
}
```

KUVA 23. Laiteosoitteen validointi

6.4.4 Hälytysjonon ylläpito

Hälytysjonon ylläpitotoiminnon tehtävänä on valvoa jonon tilaa ja arkistoida lähetetyt hälytykset tietokannassa olevaan erilliseen historiatauluun. Historiataulu on rakenteeltaan varsinaista jonotaulua vastaava, mutta siinä oleva avainkenttä ei ole automaattisesti kasvava laskurikenttä (auto increment), vaan myös avainkentän arvo kopioidaan jonotaulusta. Taulujen välillä ei ole relaatiota, vaan historiataulu on yksiselitteisesti hälytysjonon tietoarkisto. (Kuva 24.)



KUVA 24. Tietokantarakenne ER-kaaviona. Jonotaulu ja historiataulu

Ylläpitotoiminto on nimeltään br_alarm_queue. Se ajastettiin vaatimusmäärittelyn mukaisesti suoritettavaksi tunnin välein. Toiminnon ajastus poikkeaa kaikista muista järjestelmän toiminnoista. Muut toiminnot suoritetaan jatkuvassa while-silmukassa ja pysäytetään silmukan lopussa määrätyksi ajaksi sleep-funktiota käyttämällä. Ylläpitotoiminto ei sisällä while-rakennetta, vaan sen ajastus toteutettiin systemd-järjestelmäpalvelun timer-yksiköllä (6, s. 187).

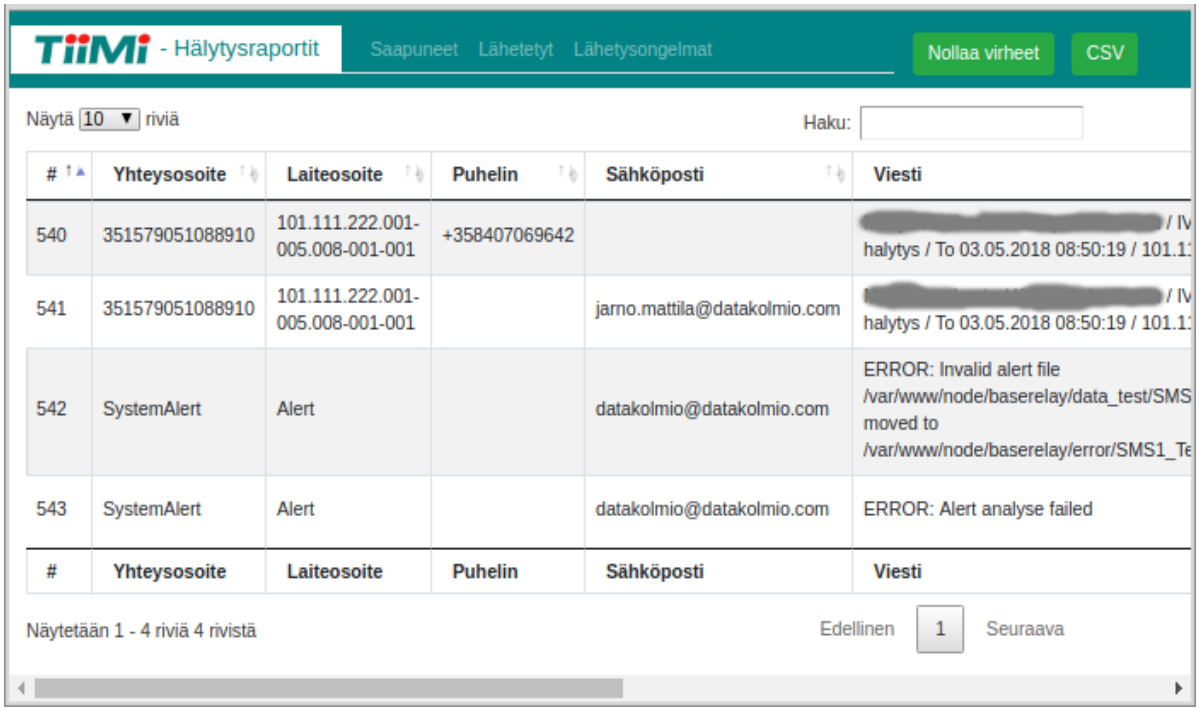
Ylläpitotoimintoa varten ohjelmoitiin Queue-luokkaan kaksi metodia:

- Archive-metodi siirtää lähetetyksi kuitatut hälytykset jonosta historiatauluun
- RetryCounterSystemAlert-metodi tarkistaa onko jonossa hälytyksiä, joilla on maksimimäärä uudelleenlähetyksiä. Toiminto muodostaa löydetyistä tietueista SystemAlert-hälytykset ja tallentaa ne jonoon, josta br_alarm_systemalert-toiminto lähettää ne edelleen ylläpitäjälle.

6.4.5 Käyttöliittymä

Vaatimusmäärittelyn mukaan järjestelmässä tuli olla käyttöliittymä, josta ylläpitäjä voi seurata hälytysjonon tilaa ja tehdä ylläpitotoimia. Ylläpitotoimina oli mainittu

viestin uudelleenlähetyslaskurin nollaus, hälytystapahtumien hakutoiminto ja hälytystapahtumien vienti CSV-tiedostoon. CSV-tiedosto (comma-separated values) on tekstitiedosto, jossa taulukkomuotoinen data on tallennettuna riveille erotinmerkillä eroteltuna (32). (Kuva 25.)



KUVA 25. Raportti-käyttöliittymä

Käyttöliittymän toteutus

Käyttöliittymä toteutettiin HTML5-dokumenttina, johon ohjelmoitiin toiminnallisuus JavaScript-skriptikielellä. Käyttöliittymän ohjelmoinnissa käytetyt ulkopuoliset ohjelmakirjastot olivat:

- jQuery: yleinen JavaScript-kirjasto, joka helpottaa ja yksinkertaistaa ohjelmointityötä (33).
- Bootstrap 4: responsiiviset tyylimäärittelyt (34).
- DataTables jQuery-kirjasto: taulukkomuotoisen datan esittämiseen kehitetty monipuolinen kirjasto (35).

Käyttämällä valmiita ohjelmakirjastoja ohjelmoinnissa voitiin keskittyä varsinaisten toimintojen toteuttamiseen. Toiminnot aktivoitiin tapahtumankuuntelijoiden avulla (event listener), jotka reagoivat esimerkiksi painikkeen klikkaukseen. Kukin

tapahtumankuuntelija saa toimintonsa dynaamisessa funktiossa, joka kirjoitettiin suoraan tapahtumankuuntelijaan.

Ajax ja asynkroninen tiedonsiirto

Käyttöliittymä kommunikoi palvelimen hälytysjonon kanssa Ajax-tekniikan avulla (Asynchronous JavaScript + XML). Perinteisesti web-käyttöliittymä kommunikoi palvelimen kanssa lataamalla tiedot palvelimelta asiakasohjelmaan (selain) ja esittämällä ne HTML-dokumenttimäärityksen mukaisesti. Tällöin asiakasohjelman on odotettava palvelimen vastausta, ennen kuin käyttöliittymä voidaan päivittää uudella tiedolla. Tietojen latauksesta johtuva viive pysäyttää käyttöliittymän toiminnan, ja toiminta palautuu vasta kun tiedot ovat latautuneet palvelimelta.

Ajax-tekniikka mahdollistaa nimensä mukaisesti asynkronisen tiedonsiirron, jossa käyttöliittymä lähettää pyynnön palvelimelle, jäämättä odottamaan vastausta. Palvelimen lähettäessä vastauksen, asiakasohjelma päivittää tiedot käyttöliittymään sitä mukaa kun tietoa palvelimelta saapuu. Ajaxin avulla käyttöliittymä säilyttää toiminnallisuuden ja vuorovaikutuksen käyttäjän kanssa koko ajan, myös tiedonsiirron aikana. (36.)

UI-luokka

Käyttöliittymää varten järjestelmään ohjelmoitiin UI-luokka, joka kommunikoi hälytysjonon kanssa Queue-luokan avulla. Koska käyttöliittymän toiminto oli täysin riippuvainen hälytysjonosta, UI-luokan ja Queue-luokan välille muodostettiin vahva kooste, eli kompositio. UI-luokka ja Queue-luokka kuuluvat oletuksena eri paketteihin. Jotta Queue-luokka voi käyttää UI-luokan parametreja, täytyi Queue-luokan ilmentymä määrittää kuulumaan UI-pakettiin. Pakettimääritys tehtiin System-luokan SetPackage-metodilla.

Luokkien välinen kompositio ja Queue-ilmentymän pakettimääritys toteutettiin UI-luokan konstruktorissa, jolloin luokkien välinen kooste toteutuu pakotetusti, aina kun UI-luokan ilmentymä luodaan (kuva 26).

```php
/**
 * Summary:     Class for Report UI
 *
 * @package     UI
 * @author      Mattila, Jarno
 * @since       2018-10-18 Initial version
 *
 */

class UI
{
    /**
     * member variables
     */

    /** @var object $objQueue */
    private $objQueue;

    function __construct()
    {
        $this->objQueue = new Queue();
        $this->objQueue->system->SetPackage("UI");
    }
```

KUVA 26. UI-luokan ja Queue-luokan kompositio muodostetaan UI-luokan konstruktorissa

UI-luokan ja käyttöliittymän väliset rajapintafunktiot sijoitettiin www-palvelimen julkiseen hakemistoon, josta asiakasohjelma kutsuu niitä http-protokollalla. Funktiota kutsutaan http-osoitteen do-argumetilla (esim. do=ResertAlert). Koska käyttäjä voi muokata do-argumentin arvoa ja lähettää näin satunnaisia funktiokutsuja, argumentti on tarkistettava ennen käyttöä. Luotettava tarkistus saatiin määrittämällä etukäteen funktioiden nimet, jotka hyväksytään do-argumentin arvoiksi. Nimistä muodostettiin taulukko, jota voitiin tarkastella PHP-kielen boolean-tyyppisellä in_array-funktiolla. Se tarkistaa datan esiintymisen taulukossa. Jos annettu argumentti sisältyy taulukkoon, in_array palauttaa tosiarvon (true) ja pyydetty funktio suoritetaan. Muussa tapauksessa pyyntö hylätään. (Kuva 27.)

```php
/** @var array arrayFunctionsAllowed Validation array */
$arrayFunctionsAllowed = array(
    "GetAllQueue",
    "FilterStage",
    "MaxRetries",
    "ResetAlert",
    "ResetAlertAll"
);

//requested function
$function = isset($_GET["do"]) ? $_GET["do"] : null;

//validation of requested function
if(in_array($function, $arrayFunctionsAllowed)){

    if(function_exists($function)){
        $function();
    }
}
```

KUVA 27. Funktion nimen validointi taulukon ja in_array-funktion avulla

UI-luokan käyttö rajapintafunktioiden avulla

Rajapintafunktiot ovat yksinkertaisia 2-rivin funktioita. Ensimmäinen rivi alustaa UI-luokan ilmentymän (olio), ja toinen rivi suorittaa pyydetyn metodin. Funktio tulostaa metodin paluuarvon käyttöliittymän ymmärtämässä muodossa. (Kuva 28.)

```php
/**
 * Reset retry counter values
 *
 * @return boolean
 */
function ResetAlert()
{
    $objUI = new UI();
    echo $objUI->ResetRetryCounter($_GET["id"]);
}

/**
 * Records with max retry valu exceeded
 *
 * @return json object
 */
function MaxRetries()
{
    $objUI = new UI();
    echo "{\"data\":". $objUI->GetQueueMaxRetries() ."}";
}
```

KUVA 28. Esimerkkejä rajapintafunktioista

Tapahtumankäsittelijän toiminta

Käyttöliittymän tapahtumankäsittelijä sisältää Ajax-kutsun ja sitä seuraavan käyttöliittymän päivityksen. Käyttöliittymä päivittyy automaattisesti, kun palvelimen vastaus vastaanotetaan. Tapahtumankäsittelijä voi sisältää useampia kuin yhden Ajax-kutsun. Esimerkiksi painikkeen klikkaustapahtuma lähettää ensimmäisessä kutsussa painikkeen do-funktiokutsun, ja vastauksen jälkeen taulukko päivitetään lähettämällä uusi Ajax-kutsu. (Kuva 29.)

```
$('#buttonReset').click(function(){
    $.get( "report_table.php?do=ResetAlertAll", function( data, status ){
        $('#alarm').DataTable( {
            "destroy"    :   true,
            "ajax"       :   jsonURL,
            "language"   :   lang_fi,
            "columnDefs": [
            {
                "visible": false,
                "targets": [5]
            },
            {
                "targets": 10,
                "render": function ( data, type, row, meta ) {
                    var columnRetry = row[10];
                    var columnID = row[0];
                    return columnRetry > 0 ? columnRetry + buttonReset(columnID, jsonURL) : columnRetry;
                }
            } ]
        } );
    });
});
```

KUVA 29. Asynkroninen $.get-funktiokutsu ja DataTable-objektin päivitys ajax-metodilla

6.5 Asennus ja käyttöönotto

Vaatimusmäärittelyn mukaan järjestelmän toimintaympäristö oli Debian GNU/Linux palvelin, joka sisälsi järjestelmän tarvitsemat palvelut: tietokanta, MTA, web-palvelin ja PHP-tulkki. Järjestelmän toiminnot tuli määrittää palvelimelle erillisiksi palveluiksi (service), joita hallitaan systemd-palveluidenhallinnan avulla. Lisäksi palvelimen omistaja määritti asennuskansiot ja muut järjestelmän toimintaan liittyvät hakemistot.

6.5.1 Asennuksen alkutoimet

Järjestelmää oli testattu kehitysvaiheessa driver-testiohjelmilla. Asennusvaiheessa varsinaiset suoritettavat palvelut ohjelmoitiin käyttämällä driver-ohjelmia

toimintojen aihioina. Lähdekoodit kommentoitiin asianmukaisesti, ja kaikki ylimääräiset merkit, rivinvaihdot ja testikoodit poistettiin. Valmis toiminto oli rakenteellisesti asianmukaisesti jäsennelty ja kommentoitu lähdekoodi, joka voitiin määrittää systemd:n alla suoritettavaksi palveluksi.

Edellä mainitut testitiedostot säilytettiin myös järjestelmän osana. Niiden avulla toimintoja voidaan ajaa bash-konsolissa, testattaessa toimintoja jatkokehitystä varten. Testitiedostot erotettiin varsinaisista toimintokoodeista nimeämiskäytännön avulla. testitiedostojen nimet sisältävät _test.php lopputunnisteen (esim. br_alert_sms_test.php). Palveluna ajettavista toimintokoodeista poistettiin tiedostotunnisteet kokonaan, ja tiedostonimen alert-sana korvattiin sanalla alarm, joka on yhdenmukainen systemd unit-tiedostojen nimeämiskäytännön kanssa.

Testiskriptit käynnistetään konsolissa php-cli-ohjelman argumenttina. (esim. php br_alert_sms_test.php). Palveluna ajettavien toimintojen kooditiedostojen alkuun lisättiin shell-komento, joka määrittää koodin suoritettavaksi php-cli ohjelmassa. Näin palveluiden määrittäminen systemd unit-tiedostossa oli yksinkertaisempaa, koska php-cli ohjelmaa ei tarvinnut erikseen kutsua, vaan komento sisältyi skriptiin itseensä .

6.5.2 Palveluiden määrittäminen

Debian 9 -järjestelmässä palveluiden määritys tapahtuu systemd-järjestelmäpalvelun avulla. Järjestelmäpalvelu on laaja kokonaisuus, joka ohjaa ja valvoo toimintoja koko järjestelmän tasolla. Kaikki systemd-järjestelmäpalvelun tehtävät on järjestelty yksiköiksi (unit). Yleisin unit on palvelu (service), lisäksi yksiköitä ovat esimerkiksi liitospiste (mount) ja ajastin (timer). Tässä kappaleessa keskitytään vain hälytysjärjestelmään liittyvien palveluiden ja ajastimien määrittämiseen. (8.)

Palvelut määritettiin unit-määritystiedostoilla, jotka sijoitettiin /etc/systemd/system -hakemistoon. Unit-tiedostoon kirjoitettiin määritettävän palvelun tiedot ja riippuvuudet muista palveluista. Tiedostot tallennettiin service-tiedostotunnisteella (esim. alarm_sms.service). Lisäksi unit-tiedostolla määritettiin ajastin (timer), joka määrittää hälytysjonon tarkistuspalvelun suoritettavaksi tunnin välein.

TAULUKKO 3. Hälytysjärjestelmän palvelut

Nimi	Tyyppi	Toiminto	Selitys
alarm_sms.service	palvelu	br_alarm_email	hälytysviestien lähettäminen sähköpostilla
alarm_email.service	palvelu	br_alarm_notifymailer	hälytysviestien lähettäminen sähköpostilla
alarm_queue.service	palvelu	br_alarm_queue	hälytysjonon valvonta
alarm_queue.timer	ajastin	alarm_queue.service	alarm_queue.service ajastus
alarm_systemalert.service	palvelu	br_alarm_systemalert	järjestelmähälytysten lähettäminen
alarm_notifymailer@.service	palvelu	br_alarm_notifymailer	valvontahälytysten lähettäminen

UNIT-tiedoston sisältö on ryhmitelty otsikoimalla määritystiedot hakasulkeilla eritellyillä otsikoilla. Hälytysjärjestelmän palveluissa käytetyt tietoryhmät ovat unit, service ja install. (Kuva 30.)

```
[Unit]
Description=Tiimi Alarm Email
After=alarm_save.service
OnFailure=alarm_notifymailer@alarm_email.service

[Service]
Type=simple

# Service to start
ExecStart=/var/www/php/alarm/br_alarm_email
WorkingDirectory=/var/www/php/alarm

# Restarting
Restart=always
RestartSec=10
StartLimitBurst=3
StartLimitInterval=100


# Killing by C-c interrup is ok
SuccessExitStatus=SIGINT

# Logging
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=alarm_email

[Install]
WantedBy=multi-user.target
```

KUVA 30. alarm_email.service-palvelun unit-määritystiedosto

Unit-määritysosa sisältää palvelun nimen (description) ja riippuvuustiedon. Riippuvuudet määritettiin after-avainsanalla, joka ilmoittaa palvelun tai palvelut, jotka täytyy olla käynnistetty ennen kuin kyseinen palvelu käynnistetään. OnFailure-määrityksellä määritettiin toiminta virhetilanteessa.

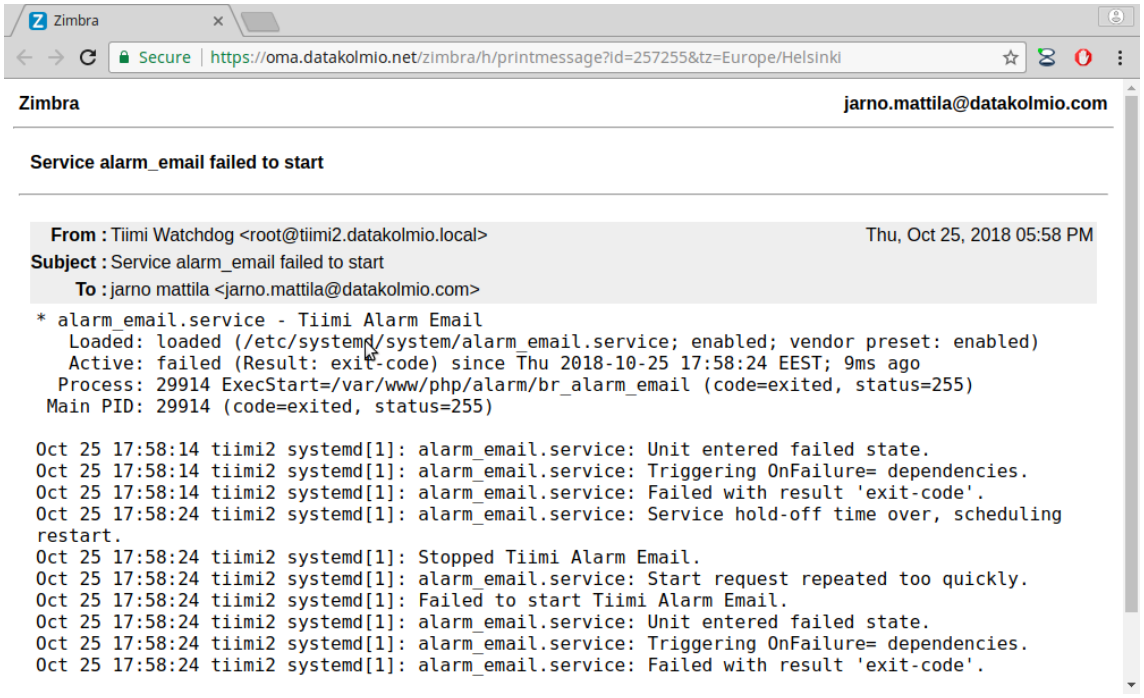
Service-määritysosa määrittelee varsinaisen suoritettavan palvelun. Suoritettava ohjelma määritettiin ExecStart-määriteellä. Ohjelmalle määritettiin myös työhakemisto (WorkingDirectory), joka on järjestelmän kotikansio. Palvelut määritettiin käynnistymään automaattisesti 10 sekunnin viiveellä (Restart ja RestartSec). Virhetilanteessa systemd yrittää käynnistää palvelun kolme kertaa, jonka jälkeen palvelu jää vikatilaan. Jokaisesta käynnistysyrityksestä lähetetään ylläpitäjälle ilmoitus sähköpostilla (OnFailure). SuccessExitStatus-määrite listaa tavat, joilla palvelun voi keskeyttää hallitusti. SIGINT arvo tarkoittaa hallintakonsolista lähetettyä keskeytyskomentoa.

Install-määritysosassa määritetään ajotaso- jolla palvelu käynnistyy. Linux-järjestelmää voidaan käyttää useilla eri ajotasoilla, esimerkiksi siten että käytetään vain

merkkipohjaista konsolia, tai siten, että käytetään graafista käyttöliittymää. Määritys multi-user.target tarkoittaa normaalia ajotasoa, jolloin palvelimelle voi olla kirjautuneena useita samanaikaisia käyttäjiä, ja esimerkiksi verkon palvelut ovat käytössä. (6, s. 185.)

Virhetilanteiden hallinta

Kaikki hälytysjärjestelmäpalvelut suorittavat virhetilanteessa alarm_notifymailer@.service-ilmoituspalvelun. Ilmoituspalvelu lähettää ylläpitäjälle sähköpostiviestin, jos palvelu ei käynnisty. Ilmoituspalvelun nimi sisältää @-merkin. Sen avulla unit-määritystiedostosta voidaan luoda useita ilmentymiä. Ilmentymä nimetään @-merkin ja tiedostotunnisteen välisellä merkkijonolla. Näin käynnistetty palvelu tietää ilmentymänsä nimen ja voi käyttää sitä ajon aikana. Kukin palvelu käynnistää alarm_notifymailler@<palvelun nimi>.service -nimisen ilmentymän, jolloin käynnistetty palvelu voi välittää käynnistäneen palvelun tiedot ylläpitäjälle. (Kuva 31.)



KUVA 31. Systemd-hälytysviesti sisältää vikatilassa olevan palvelun tilatiedot ja viimeisimmät lokimerkinnät

6.5.3 Ajastimen määrittäminen timer-yksiköllä

Palveluiden lisäksi systemd-palvelulla voidaan määrittää myös ajastettuja toimintoja. Toiminto voi olla ajastettu kalenterin ja kellon mukaan tai sekunteina edellisestä määritetystä tapahtumasta. Ajastin määritetään timer-loppuisella unit-tiedostolla (esim. ajastin.timer). (38.)

Hälytysjärjestelmää varten asetettiin ajastin, jolla alarm_queue suoritetaan tunnin välein. Toiminto valvoo ja ylläpitää hälytysjärjestelmän tietokantaa. Ajastimen unit-tiedosto sisältää [Timer] määritysosan, jossa ajastin määritetään. Hälytysjonon valvontaa ylläpitävän palvelun ajastus sisälsi kolme määritystietoa:

1. Palvelu suoritetaan aina 10 minuutin kuluttua siitä, kun palvelin käynnistetään: (OnbootSec=10min).
2. Palvelu suoritetaan aina tunnin kuluttua edellisestä suorituksesta (OnUnitActiveSec=1h).
3. Käynnistettävän palvelun unit-tiedosto on alarm_queue.service (Unit=alarm_queue.service).

Muuten ajastimen unit-tiedosto vastaa tavallisen palvelun määrittävää unit-tiedostoa. (Kuva 32.)

```
[Unit]
Description=Tiimi Alarm Queue Timer

[Timer]
OnBootSec=1min
OnUnitActiveSec=1h
Unit=alarm_queue.service

[Install]
WantedBy=multi-user.target
```

KUVA 32. alarm_queue-palvelun käynnistävän ajastimen määritys

6.5.4 Palveluiden hallinta systemctl-ohjelmalla

Systemd-järjestelmäpalvelun palveluita hallitaan systemctl-ohjelmalla. Palvelu aktivoidaan käynnistymään automaattisesti komennolla: ”systemctl enable <pal-

velun nimi>”. Vastaavasti disable-argumentti poistaa automaattisen käynnistyksen. Palvelu käynnistetään start-argumentilla ja pysäytetään stop-argumentilla. Samoin kuin aktivoinnissa, käynnistys- ja pysäytysargumentteja seuraa aina palvelun nimi. Start- ja stop-argumenttien lisäksi myös restart-argumenttia voi käyttää. Se suorittaa palvelun uudelleenkäynnistyksen. Status-argumenttia käyttämällä saadaan tietoa palvelun tilasta (kuva 33).

```
root@tiimi2:~# systemctl status alarm_sms.service
* alarm_sms.service - Tiimi Alarm SMS
   Loaded: loaded (/etc/systemd/system/alarm_sms.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2018-11-03 14:41:36 EET; 4h 23min ago
 Main PID: 13673 (br_alarm_sms)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/alarm_sms.service
           `-13673 /usr/bin/php /var/www/php/alarm/br_alarm_sms

marras 03 14:41:36 tiimi2 systemd[1]: Stopped Tiimi Alarm SMS.
marras 03 14:41:36 tiimi2 systemd[1]: Started Tiimi Alarm SMS.
```

KUVA 33. Tietoja alarm_sms-palvelun tilasta

6.5.5 Järjestelmä- ja hälytyslokien hallinta

Systemd unit-tiedoston logging-tieto määrittää käytettävän lokipalvelun (syslog). Debian 9 järjestelmäloki käyttää rsyslog-palvelua, joka on kehittynyt monisäikeinen järjestelmälokipalvelu (37). Hälytysjärjestelmää varten rsyslog-palveluun lisättiin oma asetustiedosto, jossa määritetään, että kaikki hälytysjärjestelmäpalveluiden syslog-tapahtumat kirjataan lisäksi myös hälytysjärjestelmän omaan lokitiedostoon (kuva 34).

```
root@tiimi2:/var/www/php/alarm# tail -f /var/log/tiimi_alarm.log
03.11.2018 14:41:04:200312      Alert                   I       Empty Alert File Folder
03.11.2018 14:41:07:654129      Message                 I       No email alerts on the queue.
03.11.2018 14:41:09:304380      Message                 I       No SMS alerts on the queue.
03.11.2018 14:41:14:200830      Alert                   I       Empty Alert File Folder
03.11.2018 14:41:17:655019      Message                 I       No email alerts on the queue.
03.11.2018 14:41:19:306508      Message                 I       No SMS alerts on the queue.
03.11.2018 14:41:24:201175      Alert                   I       Empty Alert File Folder
03.11.2018 14:41:27:655703      Message                 I       No email alerts on the queue.
03.11.2018 14:41:29:307192      Message                 I       No SMS alerts on the queue.
03.11.2018 14:41:34:201537      Alert                   I       Empty Alert File Folder
Nov  3 14:41:36 tiimi2 systemd[1]: Stopping Tiimi Alarm SMS...
Nov  3 14:41:36 tiimi2 systemd[1]: Stopped Tiimi Alarm SMS.
Nov  3 14:41:36 tiimi2 systemd[1]: Started Tiimi Alarm SMS.
03.11.2018 14:41:36:174592      Message                 I       No SMS alerts on the queue.
03.11.2018 14:41:37:656388      Message                 I       No email alerts on the queue.
03.11.2018 14:41:44:201875      Alert                   I       Empty Alert File Folder
```

KUVA 34. SMS-palvelun uudelleenkäynnistys kirjautuu hälytysjärjestelmän lokiin

Lokitietojen käsittely viimeisteltiin määrittämällä logrotate-toiminnolla hälytysjärjestelmän tapahtumalokin säilytyskäytännöt. Käytännöksi määritettiin, että tapahtumalokeista muodostetaan päivittäin erillinen tiedosto, ja järjestelmä säilyttää 10 viimeisintä lokitiedostoa (kuva 35).

```
root@tiimi2:/var/log# ls -lh tiimi_alarm.log*
-rw-r----- 1 root adm 1,3M marra  3 14:38 tiimi_alarm.log
-rw-r----- 1 root adm 2,0M loka  26 00:00 tiimi_alarm.log-20181026
-rw-r----- 1 root adm 2,0M loka  26 23:59 tiimi_alarm.log-20181027
-rw-r----- 1 root adm 2,0M loka  28 00:00 tiimi_alarm.log-20181028
-rw-r----- 1 root adm 2,1M loka  29 00:00 tiimi_alarm.log-20181029
-rw-r----- 1 root adm 2,0M loka  30 00:00 tiimi_alarm.log-20181030
-rw-r----- 1 root adm 2,0M loka  31 00:00 tiimi_alarm.log-20181031
-rw-r----- 1 root adm 2,0M marra  1 00:00 tiimi_alarm.log-20181101
-rw-r----- 1 root adm 2,0M marra  1 23:59 tiimi_alarm.log-20181102
-rw-r----- 1 root adm 2,0M marra  3 00:00 tiimi_alarm.log-20181103
```

KUVA 35. Logrotate kierrättää päivittäin 10 viimeisintä lokitiedostoa

7 TESTAUS

Järjestelmän testaus on laaja, koko kehitysvaiheen kestävä prosessi, jonka tarkoituksena on varmistaa, että järjestelmä on toiminnoiltaan ja laatuvaatimuksiltaan sellainen kuin sen on tarkoituskin olla. Järjestelmää testataan useilla eri tavoilla sekä kehittäjän, että ulkopuolisen testaajan testeillä:

- Yksikkötestaus (unit testing) tarkoittaa luokan, metodin tai tehtäväkohtaisen pienoisohjelman testausta, joka tehdään erillään muusta järjestelmästä. Testauksen tekee yleensä ohjelmoija itse.
- Komponenttitestaus (component testing) tarkoittaa luokan, paketin tai pienen ohjelman testausta, joka tehdään erillään muusta järjestelmästä. Testauksen tekee yleensä ohjelmointitiimi, joka vastaa komponentin toteutuksesta.
- Integraatiotestaus (integration testing) tarkoittaa kahden tai useamman luokan, paketin, komponentin tai alijärjestelmän testausta. Integraatiotestaus on ohjelmointitiimien suorittamaa testausta, joka alkaa heti kun kaksi luokkaa on valmiina testattavaksi. Testaus jatkuu, kunnes koko järjestelmän on valmis.
- Regressiotestaus (regression testing) tarkoittaa jo testattujen kohteiden uudelleentestausta, jonka tarkoituksena on selvittää, onko aikaisemmin testauksen läpäissyt toiminto edelleen toimiva, kun järjestelmään on tehty muutoksia.
- Järjestelmätestaus (system testing) tarkoittaa koko järjestelmän testausta lopullisessa muodossaan, tuotantoympäristöä vastaavalla alustalla ja kaikki ulkopuoliset laite- ja ohjelmarajapinnat käytössä. Testauksen tarkoituksena on tietoturvan, suorituskyvyn, resurssikuormituksen, ajastuksien ja muiden sellaisten ongelmien tunnistaminen, joita ei voida luotettavasti tunnistaa edellä mainituilla testauksilla.

(5, s. 499.)

7.1 Käytetyt testaustavat

Testaus jaetaan yleensä kahteen kategoriaan: sisäinen testaus (white-box testing) ja ulkopuolinen testaus (black-box testing). Black-box tarkoittaa testaustilannetta, jossa testaaja ei näe järjestelmän sisällä olevaa toimintaa, vaan voi havainnoida ärsykkeen aiheuttamaa vaikutusta vain järjestelmän palauttamalla tuloksella. White-box tarkoittaa testausta, jossa testaaja voi seurata, millä tavalla järjestelmä toimii annetulla ärsykkeellä, mitä komentoja se suorittaa ja miten toiminto etenee järjestelmän sisällä. (5, s. 500.)

Tämän työn aikaisemmissa kappaleissa on kerrottu hälytysjärjestelmän alemman tason testauksesta. Lähdekoodin syntaksitarkistusta tehtiin ohjelmoinnin edetessä, käyttämällä lähdekoodieditoriin integroitua PHP-tulkkia (unit testing). Luokkia ja toimintoja testattiin driver-testiohjelmien ja stub-tynkäluokkien avulla (component testing). Toimintokokonaisuuden eli paketin valmistuttua, asennettiin tuotantopalvelinta vastaavaan testiympäristöön testijärjestelmä asiakasyrityksen testaajaa varten. Näin voitiin toteuttaa järjestelmän testausta (system testing) ja regressiotestausta (regression testing) koko järjestelmän kehitystyön ajan.

Järjestelmän valmistumisen, palvelinmääritysten ja käyttöönottomääritysten jälkeen, suoritettiin vielä dokumentoitu järjestelmätestaus, jonka tarkoituksena oli arvioida järjestelmän toimivuutta vaatimusmäärittelyn näkökulmasta.

7.2 Testauksen suunnittelu

Järjestelmätestauksen suoritti järjestelmän kehittäjä. Testausmenetelmänä oli white-box testaus. Testaus suunniteltiin vaatimusmäärittelyn pohjalta, keräämällä määrittelystä järjestelmän ei ominaisuuksia ja toimintoja kuvaavia vaatimuksia. Vaatimusmäärittely oli rakenteeltaan toimintoperusteinen. Jokainen toiminto oli kuvattu sanallisesti, kertomalla miten toiminnon tulee toimia, millaisia tietoja toiminto käsittelee ja mitä vaatimuksia toimintoon kohdistuu. Lisäksi jokaisesta toiminnosta oli laadittu ”Toiminnalliset vaatimukset” -kappale, jossa kuvattiin kriittiset toiminnot, ja miten toiminnon tulee toimia virhetilanteessa.

Testaussuunnitelma laadittiin taulukkomuotoon, kunkin vaatimusmäärittelyn vaatimuksen muodostaessa yhden taulukon rivin. Testaussuunnitelman sarakkeet olivat

1. NUM: laskurikenttä
2. DATE: testausajankohta
3. OPERATOR: testauksen suorittaja
4. MODULE: testattava toiminto (paketti)
5. TEST: kuvaus suoritettavasta testauksesta
6. RESULT: testaustulos (hyväksytty/hylätty).

Testaussuunnitelmaan listattiin yhteensä 19 suoritettavaa testiä. Testauksen valmistuttua testaustulokset kirjattiin samaan taulukkoon. (Kuva 36.)

	A	B	C	D	E	F
1	2018-10-24				Tiimi Alarm Test Summary	
2	NUM	DATE	OPERATOR	MODULE	TEST	RESULT
3	1	2018-10-22	JM	Alert	Alert Directory polling and alert save in DB	PASSED
4	2	2018-10-22	JM	Alert	Alert Directory polling with invalid filename	PASSED
5	3	2018-10-22	JM	Alert	Alert Analyzing – valid data	PASSED
6	4	2018-10-22	JM	Alert	Alert Analyzing – invalid data (line missing, extra line)	PASSED
7	5	2018-10-22	JM	Alert	Alert Analyzing – invalid controller address	PASSED
8	6	2018-10-22	JM	Alert	Getting Alert instructions by Portal Interface function	PASSED
9	7	2018-10-22	JM	Alert	Saving Alert to Queue	PASSED
10	8	2018-10-22	JM	Alert	Saving Alert to Queue without Instructions	PASSED
11	9	2018-10-23	JM	Alert	Test logging and functionality if alert directory or error directory doesn't exist	PASSED
12	10	2018-10-23	JM	Alert	Database connection not established	PASSED
13	11	2018-10-23	JM	Message (ASPMSM)	Alert SMS polling and and sending when no SMS alert in the queue	PASSED
14	12	2018-10-23	JM	Message (ASPMSM)	Alert SMS polling and and sending when SMS alert in the queue, when receiver phones have country code and not	PASSED
15	13	2018-10-23	JM	Message (ASPMSM)	Alert SMS error test. retry_counter value increasing test	PASSED
16	14	2018-10-24	JM	Message (EMAIL)	Alert Email polling and and sending when no Email alert in the queue	PASSED
17	15	2018-10-24	JM	Message (EMAIL)	Alert Email polling and and sending when Portal receiver mail contains a valid email address.	PASSED
18	16	2018-10-24	JM	Message (EMAIL)	Alert email error test: invalid recipient address	PASSED
19	17	2018-10-24	JM	Message (EMAIL)	Alert email error test: failed mail delivery increases the retry counter value	PASSED
20	18	2018-10-24	JM	Message (SystemAlert)	System Alert Email Sending by 10min period.	PASSED
21	19	2018-10-24	JM	Message (Queue)	Sending System Alert Email if retry_counter-values is reached max value	PASSED

KUVA 36. Testaussuunnitelma ja tulosten merkitseminen

7.3 Testauksen tekeminen

Testaukset dokumentointiin niin tarkasti, että dokumentista voitiin yksiselitteisesti päätellä testaustulos. Testausdokumentin runko otettiin testaussuunnitelmasta, mutta siihen lisättiin testaustilannetta tarkentavia tietoja:

- OBJEKTIVE: testin tavoite, mitä on tarkoitus testata
- PRECONDITION: alkutilanteen kuvaus
- TEST DATA: testissä käytettävä data, esim. hälytystiedoston sisältö
- EXPECTED RESULT: odotettu tulos, miten ohjelman pitäisi toimia
- RESULT: toteutunut tulos, miten ohjelma toimi testauksessa
- TEST STATUS: testaustulos: hyväksytty/hylätty.

Testauksen alussa dokumentoitiin alkutilanne ja testattavat asiat (taulukko 4). Testauksen aikana ohjelman toimintaa seurattiin konsolitulosteiden lokitietojen ja käyttöjärjestelmän antaman tiedon avulla. Tavoitteena oli kerätä monipuolisesti ja riittävästi tietoa testaustuloksen arviointia varten.

Testauksen rakenne oli kaikissa testauksissa sama, mutta toimintojen seuranta ja dokumentointi vaihteli eri testien välillä. Esimerkiksi virhetilannetta testattaessa, seurattiin tapahtumalokia vain alimmalla tasolla (error), jolloin toimintaa virhetilanteessa voitiin arvioida tarkemmin. Testattaessa toimintoa yleisesti, voitiin tapahtumaloki asettaa informatiivisemmaksi (informaatio tai debug -taso), jolloin saatiin tarkempia tietoja ohjelman sisäisestä toiminnasta.

TAULUKKO 4. Virhetilanteen testaus, alkutilanteen dokumentaatio. Virhe korostettu värillä

TEST 16	24.10.2018
OPERATOR	J.M.
OBJECTIVE	Alert email error test: invalid recipient address
PRECONDI-TION	Polling started. Empty alert directory, empty database. Log-level 4: Information

(taulukko 4 jatkuu)

(taulukko 4 jatkuu)

TEST	Save valid alert file with valid receiver information in the alert directory. Instructions contain invalid email address Alert file contains sms recipient with country-code prefixed phone number
TEST DATA	Alert File: 351579051088910 AT+CMGS="+358407069642" - / TiiMi 7610 5.00 / Murto halytys / To 03.05.2018 08:50:19 / 101.111.222.001-005.008-001-001 Portal database: MariaDB [portal1]> select controller_address,node_name, receiver_phone, receiver_mail from tree where node_id = 5 or node_parent_id = 5; +------------------+--------------+---------------+------------------------+ | controller_address | node_name | receiver_phone | receiver_mail | +------------------+--------------+---------------+------------------------+ | 101.111.222.1 | IV-Tulo | | | | Jarno Mattila | 0407069642 | jarno.mattila@datakol-miocom | +------------------+--------------+---------------+------------------------+ 2 rows in set (0.00 sec) Queue database: MariaDB [alarm_queue1]> select * from alert_queue; Empty set (0.00 sec)
EXPECTED RESULT	Log Information: New alert found. Alert added to queue with 2 receiver_phone-value. Field receiver_email IS NULL. No email sent.

7.4 Testaustulosten arviointi

Testaustuloksia arvioitiin ensisijaisesti lopputuloksen kannalta. Jos toiminnon täytyy lähettää sähköposti tiettyyn osoitteeseen, niin lähtikö viesti ja tuliko viesti perille? Lisäksi arvioitiin millä tavalla ohjelma toimii, miten se reagoi annettuun ärsykkeeseen, miten dataa käsiteltiin ja mitä vaiheita ohjelma suoritti? Arvioinnissa hyödynnettiin toimintolokia, järjestelmälokeja, tietokannan tietoja ja yleistä toiminnan havainnointia. Jos ohjelman esimerkiksi täytyi poistaa tai siirtää tiedosto kansiosta, toiminto tarkistettiin suorittamalla kansiolistaus ja toteamalla tiedoston sijainti tai poistuminen.

Testaustulokset tallennettiin samaan testauspöytäkirjaan alkutilannetta kuvaavan taulukon jatkeeksi (taulukko 5). Tuloksista korostettiin väreillä ne havainnot, joihin arviointi perustui. Samoja värikorostuksia voitiin käyttää myös arviointilausunnoissa, jolloin asiayhteys arvioitavan tiedon ja arvion välillä oli helpompi huomata. Värikorostuksesta oli hyötyä erityisesti silloin, kun testaustietoa oli dokumentoitava paljon, tai testausarvio perustui useampaan kuin yhteen tekijään.

TAULUKKO 5. Testaustulokset ja arviointi

TEST RESULT	Log: 24.10.2018 18:16:25:127634 Alert I Empty Alert File Folder 24.10.2018 18:16:30:545899 Message I No email alerts on the queue. 24.10.2018 18:16:35:128080 Alert I New alert found! Starting analyse... 24.10.2018 18:16:35:128461 Alert I Analyse done. Message is: 24.10.2018 18:16:35:128572 Alert I - / TiiMi 7610 5.00 / Murto halytys / To 03.05.2018 08:50:19 / 101.111.222.001-005.008-001-001 24.10.2018 18:16:35:128713 Alert I Getting sending instructions from Portal for: 351579051088910, 101.111.222.001, 101.111.222.1 24.10.2018 18:16:35:129294 Alert I Done. Handling reading instructions... 24.10.2018 18:16:35:129440 Alert I Done. Adding alert to Queue... 24.10.2018 18:16:35:135519 Queue W AddToQueue: validation of receiver jarno.mattila@datakolmiocom failed. Using NULL 24.10.2018 18:16:35:138516 Alert I Done. Alert added to Queue 24.10.2018 18:16:35:138699 Alert I AlertFile SMS1_Test1_ValidFile_ValidFilename removed 24.10.2018 18:16:36:139122 Alert I Empty Alert File Folder 24.10.2018 18:16:40:546841 Message I No email alerts on the queue. 24.10.2018 18:16:46:139462 Alert I Empty Alert File Folder 24.10.2018 18:16:50:547624 Message I No email alerts on the queue.

(taulukko 5 jatkuu)

(taulukko 5 jatkuu)

	Queue database: MariaDB [alarm_queue1]> select * from alert_queue; +----------+-------------------+------------------------------+------------- --+--------------------------+---+- --- ---+------------------- --------------+--------------------+-----------+---------------+ | queue_id | connection_address | controller_address | re- ceiver_phone | receiver_mail | controller_prefix | message_body | alert_filename | arrived | delivered | retry_coun- ter | +----------+-------------------+------------------------------+------------- --+--------------------------+---+- | 524 | 351579051088910 | 101.111.222.001-005.008-001-001 | +358407069642 | NULL | / / IV-Tulo | / / IV-Tulo / - / TiiMi 7610 5.00 / Murto halytys / To 03.05.2018 08:50:19 / 101.111.222.001-005.008-001-001 | SMS1_Test1_ValidFile_Valid- Filename | 2018-10-24 18:19:16 | NULL | 0 | | 525 | 351579051088910 | 101.111.222.001-005.008-001-001 | 0407069642 | NULL | / / IV-Tulo | / / IV-Tulo / - / TiiMi 7610 5.00 / Murto halytys / To 03.05.2018 08:50:19 / 101.111.222.001-005.008-001-001 | SMS1_Test1_ValidFile_Valid- Filename | 2018-10-24 18:19:16 | NULL | 0 | | 526 | SystemAlert | Queue | NULL | datakolmio@datakolmio.com | WARNING 24.10.2018 18:19:16 | WARNING: AddToQueue: validation of receiver jarno.mat- tila@datakolmiocom failed. Using NULL | | 2018-10-24 18:19:16 | NULL | 0 | | 527 | 351579051088910 | 101.111.222.001-005.008-001-001 | NULL | NULL | / / IV-Tulo | / IV-Tulo / - / TiiMi 7610 5.00 / Murto halytys / To 03.05.2018 08:50:19 / 101.111.222.001-005.008-001-001 | SMS1_Test1_ValidFile_Valid- Filename | 2018-10-24 18:19:16 | NULL | 0 | +----------+-------------------+------------------------------+------------- --+--------------------------+---+- Results: Log information after new alert: New alert found! Starting analyse… Invalid receiver found: AddToQueue: validation of receiver jarno.mattila@datakolmiocom failed. Using NULL 2 records found with receiver_phone All alert records have NULL receiver_mail-value Extra-behavior: SystemAlert generated for warning level log entry No email alerts on the queue. No email message sent
TEST STATUS	Passed

8 YHTEENVETO JA TULOKSET

8.1 Työn tavoite

Opinnäytetyön tavoitteena oli toteuttaa Tiimi-ohjelmaportaaliin integroitu hälytysten ohjaustoiminto. Kehitettävän järjestelmän tuli välittää kiinteistöautomaatiolaitteelta tuleva hälytys laitteen omistajalle tai LVI-asennusliikkeelle, joka vastaa laitteiston kunnossapidosta ja toiminnasta. Järjestelmän tuli analysoida palvelimelle tekstitiedostona tallennettava hälytystieto, selvittää viestin vastaanottajat ja välittää hälytys eteenpäin vastaanottajatiedoissa määrätyllä tavalla, joko sähköpostilla, tekstiviestillä tai molemmilla tavoilla.

8.2 Työn eteneminen

Esiselvitys

Työ alkoi esiselvityksellä, jossa tutustuttiin lähtötilanteeseen, miten hälytystoiminta toimi sillä hetkellä ja mitä puutteita ja kehittämistarpeita toiminnossa oli. Esiselvitysvaiheessa muotoutui käsitys suoritettavasta tehtävästä, laite- ja sovellusympäristöstä ja kiinteistöautomaatiotoimialasta yleensä, joka ei ollut opinnäytetyön tekijälle entuudestaan kovin tuttu.

Vaatimusmäärittely

Esiselvityksen jälkeen kehitettävästä järjestelmästä tehtiin kirjallinen vaatimusmäärittely, jossa kuvattiin järjestelmän eri toiminnot ja niiden vaatimukset. Vaatimusmäärittely laadittiin toimintoperusteisesti, kuvaamalla toimintojen tehtävät ja tiedot, joita kukin toiminto käsittelee. Tärkeimmistä toiminnallisista vaatimuksista kirjattiin määrittelyyn yksityiskohtaiset vaatimukset, miten ohjelman tulisi toimia tietyssä tilanteessa ja miten tulisi toimia todennäköisessä virhetilanteessa. Vaatimusmäärittely rajasi myös yleisiä vaatimuksia, kuten toimintaympäristö, ulkopuoliset rajapinnat ja käytettävät tekniikat.

Analyysi ja suunnittelu

Vaatimusmäärittelyn jälkeen seurasi analyysi- ja suunnitteluvaihe. Analyysivaiheessa vaatimusmäärittely käytiin tarkasti läpi, etsien tekstistä mahdollisesti tunnistettavia ohjelmarakenteita, kuten luokkia, metodeja ja luokkien ominaisuuksia. Analyysi ja suunnittelu nivoutuivat työn aikana yhteen. Kun ymmärrys kehitettävästä järjestelmästä kasvoi analyysin myötä, alkoi samalla hahmottua ohjelman rakenne, tarvittavat luokat ja miten ohjelman eri komponenttien tulisi toimia ja kommunikoida keskenään. Analyysi- ja suunnitteluvaiheessa hyödynnettiin UML-mallinnusta, jolla kuvattiin tärkeimmät toiminnot (toimintokaavio), ja kaikki luokat ja niiden väliset sidokset (luokkakaavio). Suunnitteluvaiheessa luokat ja ohjelmakomponentit ryhmiteltiin paketeiksi. Paketointimallista jalostui suunnitteluvaiheessa myös perusta järjestelmän konfiguraatiolle ja parametrien ryhmittelylle.

Ohjelmointi

Suunnitteluvaiheesta työ eteni ohjelmointivaiheeseen. Kullekin toiminnolle kirjoitettiin pääluokka ja testiohjelma, jolla luokan toimintoja alettiin testata heti ohjelmointivaiheessa. Kun pääluokka tai testiohjelma tarvitsi jonkun toisen luokan toimintoa, kirjoitettiin apuluokkaa niin pitkälle, että tarvittavaa toimintoa voitiin käyttää. Alkuvaiheessa apuluokan toiminto oli lyhyt tynkäohjelma, joka yksinkertaisesti palautti oletetun arvon. Esimerkiksi boolean-tyyppinen tiedontarkistusmetodi sisälsi vain tosi/epätosi palautusarvon, mikä riitti pääohjelman testaukseen.

Suunnitteluvaiheen UML-mallinnukset ja niihin perustuvat suunnitelmat eivät olleet aluksi tarkkoja, vaan tarkoitus oli hahmottaa ohjelman rakenne ja eri asioiden väliset vuorovaikutukset yleisellä tasolla. Ohjelmoinnin edetessä UML-luokkakaavioita kuitenkin päivitettiin vastaamaan todellista tilannetta. UML-mallinnuksesta oli hyötyä myös ohjelmointivaiheessa, koska kaaviomuodossa esitetyt luokkien väliset riippuvuudet ovat riittävästi yksinkertaistettuja, jotta niissä esiintyvät epäloogisuudet oli helppo havaita.

Ohjelmaa testattiin jo ohjelmointivaiheessa käyttämällä erilaisia testiympäristöjä. Kehitysympäristö itsessään vastasi pitkälti tulevaa tuotantopalvelinta. Kehitysym-

päristön lisäksi käytettiin erillistä testipalvelinta, jonka kokoonpano poikkesi tuotantopalvelimesta sekä Linux-jakeluversion, että PHP-version osalta. Näin haluttiin varmistaa mahdollisimman laaja yhteensopivuus ja toimintavarmuus erilaisten toimintaympäristöjen kanssa. Toimintavarmuutta lisättiin myös erillisellä apuluokalla, joka oli kaikkien luokkien ja toimintojen käytössä, ja jonka avulla vähennettiin PHP-tulkin versiopäivityksistä aiheutuvia mahdollisia ongelmia nyt ja tulevaisuudessa.

Asennus

Järjestelmä asennettiin tuotantopalvelimelle vaiheittain, työn edistymisen mukaan. Toimintoja testattiin tuotantopalvelimella yksitellen, käyttämällä tarkoitusta varten ohjelmoituja testiskriptejä. Järjestelmän valmistuttua kokonaan, se asennettiin palvelimelle vaatimusmäärittelyssä kuvatulla tavalla. Toiminnot määritettiin suoritettavaksi palvelimella automaattisesti käynnistyvinä taustapalveluina. Taustapalveluiden lisäksi järjestelmään ohjelmoitiin selainpohjainen käyttöliittymä, jonka avulla järjestelmän ylläpitäjä voi seurata hälytysten käsittelyjä ja tuottaa hälytyshistoriasta raportteja.

Testaus

Valmiin järjestelmän testaus suunniteltiin vaatimusmäärittelyn pohjalta. Suunnitelma sisälsi 19 testitapausta, joille suunniteltiin tapauskohtaiset testaustilanteet. Testit ja niiden tulokset dokumentoitiin testauspöytäkirjoihin. Testaussuunnitelmasta ja testauspöytäkirjoista koostettiin taulukkomuotoinen testausraportti, josta näki yhdellä silmäyksellä testauskohteet ja -tulokset. Testaus katsottiin suoritetuksi hyväksytysti, kun kaikki suunnitellut testit oli suoritettu vaadituilla kriteereillä.

8.3 Työn lopputulos

Opinnäytetyönä valmistettu laitehälytysten hallinta ja ohjausjärjestelmä on palvelinkeskeinen sovellus, joka käsittelee saapuvat laitehälytykset ja lähettää ne edelleen vastaanottajille. Järjestelmän toiminnot ovat

- laitehälytysten välitys vastaanottajalle tekstiviestillä

- laitehälytysten välitys vastaanottajalle sähköpostilla
- laitehälytyshistorian tallennus ja raportointi.

Järjestelmä on suunniteltu käytettäväksi Linux-palvelimella, PHP-CLI SAPI -ympäristössä. Järjestelmän toiminnot toimivat taustapalveluina, jotka käynnistyvät automaattisesti palvelimen käynnistyksen jälkeen ja suorittavat määritettyjä tehtäviä ajastetusti määrätyillä aikaväleillä. Opinnäytetyön toteutuksessa palveluta hallitaan systemd-järjestelmäpalvelun avulla, käyttäen systemctl-ohjelmaa.

Järjestelmän käyttämät sisäiset ja ulkopuoliset palvelut ja rajapinnat ovat

- MariaDB-tietokantapalvelu
- nginx http-palvelu
- Postfix MTA sähköpostien välityspalvelu
- SMS tekstiviestien välityspalvelu (ulkopuolinen palvelu)
- BaseRelay laitehälytysten välityspalvelu (ulkopuolinen palvelu)
- Tiimi-portaalin asiakastietokanta (ulkopuolinen palvelu).

Järjestelmä sisältää vikasietoisuutta ja ylläpitoa parantavia toimintoja, kuten virheenkäsittelytoiminnot hälytysten käsittelyn yleisimmille virhetilanteille (esim. virheellinen tai puuttuva vastaanottajatieto). Järjestelmä ylläpitää toiminnoistaan tapahtumalokia. Tapahtumalokin tasoa on mahdollisuus muuttaa toimintokohtaisesti. Tapahtumaloki sisältää neljä tasoa, jotka määräytyvät luvun 2 potenssien mukaan:

- $2^0 = 1$: virhe (error)
- $2^1 = 2$: varoitus (warning)
- $2^2 = 4$: informaation (information)
- $2^3 = 8$: testaus (debug).

Alin taso (virhe) kirjaa lokiin vain järjestelmän tuottamat virheilmoitukset. Tasoa voidaan muuttaa pakettikohtaisesti, konfiguraation logLevel-arvoa muuttamalla. Kun tasoa nostetaan ylöspäin, tapahtumalokin informatiivisuus lisääntyy. Ylempi taso sisältää aina kaikki alemman tason lokitapahtumat, joten ylin taso (testaus)

tarkoittaa testausta varten tallennettavien lokitietojen lisäksi kaikkien virhe-, varoitus- ja informaatiotapahtumien kirjaamista tapahtumalokiin.

Alimman virhe-tason tapahtumista muodostuu lokikirjauksen lisäksi sähköpostilla toimitettava järjestelmähälytys. Hälytykset toimitetaan vastaanottajalle saman lähetyskanavan kautta kuin laitehälytyksetkin, sillä erotuksella että järjestelmähälytyksen vastaanottaja on aina sama konfiguraatiossa määritetty ylläpitäjän sähköpostiosoite.

Järjestelmäkonfiguraatio on kahdennettu, ja sisältää ensisijaisen konfiguraation ja oletuskonfiguraation. Ensisijaista konfiguraatiota voidaan muuttaa ylläpitäjän toimesta, pysäyttämättä palveluita. Järjestelmä tarkistaa aina konfiguraation syntaksin ja kunkin parametrin arvon. Jos konfiguraatiotiedosto on viallinen tai parametrin arvo poikkeaa sallitusta, järjestelmä hylkää ensisijaisen arvon ja käyttää oletusarvoa.

Sisäänkirjoitettujen virheenkäsittelyiden ja vikatilanteiden hallinnan lisäksi järjestelmän toimintaa valvoo käyttöjärjestelmän tarjoama systemd palveluidenhallinta -toiminto. Systemd valvoo järjestelmän toimintaa ja käynnistää mahdollisesti pysähtyneen palvelun automaattisesti uudelleen. Kaikki palveluiden pysähtymiset ja uudelleenkäynnistämiset ilmoitetaan ylläpitäjälle sähköpostiviestillä, jolloin vikatilanne huomataan ja vikaselvitys voidaan aloittaa heti.

8.4 Yhteenveto

Opinnäytetyönä toteutettu järjestelmä on monipuolinen palvelu, joka sisältää vaaditut toiminnot laitehälytysten käsittelyyn ja välittämiseen, monipuoliset virheenhallintaominaisuudet ja vikasietoisen konfiguraation. Järjestelmä toimii sulautettuna osana laajempaa kokonaisuutta, Team-Control Oy:n kiinteistöautomaatiolaitteiden kommunikaatiota. Se on määritetty toiminaan Linux-palvelinjärjestelmään integroituna taustapalveluna. Järjestelmä käyttää monipuolisesti sekä paikallisia että ulkopuolisia palveluita.

Järjestelmä sisältää myös selainkäyttöisen käyttöliittymän, jonka avulla ylläpitäjä voi valvoa ja hallita hälytysjonoa.

Järjestelmän palvelut on ohjelmoitu PHP-skriptikielellä, käyttäen olio-ohjelmointitekniikkaa. Järjestelmän suunnittelu perustuu dokumentoituun vaatimusmäärittelyyn ja oliopohjaiseen analyysiin ja suunnitteluun.

Koko järjestelmä määritystiedostoineen sisältää

- 5 palvelua
- 9 Linux-palvelimen määritystiedostoa
- 6 konsoliohjelmaa
- 1 tietokannan
- 1 selainkäyttöliittymän
- 11 luokkaa
- n. 3 500 riviä lähdekoodia.

9 LOPPUSANAT

Opinnäytetyön aihetta mietittäessä lähtökohtana olivat opintoihin liittyneet aikaisemmat yrityskohtaiset kehitysprojektit. Projektien aiheena olivat olleet erilaiset kiinteistöautomaatioon liittyvät kehitystyöt. Ensimmäinen projekti keskittyi säätimen selainkäyttöliittymän kehittämiseen, toisen projektin aikana toteutettiin säätimien keskitettyyn hallintaan tarkoitettu palvelinympäristö ja kolmannen projektin aikana kehitettiin käyttäjäportaalia, jonka avulla IoT-säätimiä voidaan hallita keskitetysti. Kaikki projektityöt oli tehty yhteistyössä Team-Control Oy:n kanssa, joten Team-Control Oy:n ehdottama opinnäytetyön aihe tuntui lähtökohtaisesti mielenkiintoiselta.

Opinnäytetyön aihe, Tiimi-portaalin hälytysten käsittely ja ohjaus, sopi hyvin jatkeeksi jo projektitöiden aikana aloitetulle työlle. Aihe käsitteli samoja teemoja, kiinteistöautomaatioon liittyvien säätimien etähallintaa, mutta näkökulma oli uusi. Tavoitteena oli toteuttaa monipuolinen palvelu, jossa oli useita toimintoa, mutta selkeästi rajattu tehtävä. Aihe myös sivusi aikaisempia projektitöitä, joten vaikka toimiala on minulle entuudestaan tuntematon, käsitteet ja termistöt olivat jo jollain lailla tuttuja.

Kehitettävän järjestelmän toteutustapa oli työn alkuvaiheessa epäselvä. Mietittiin, tulisiko toteutustapa perustua yksinkertaiseen proseduraaliseen ohjelmointiin vai olisiko mielekästä käyttää olio-ohjelmointia. Tulisiko ratkaisun perustua tilakonemalliin vai itsenäisesti toimiviin toimintokohtaisiin palveluihin? Opinnäytetyön lähtötietomuistiossa toteutustapaa ei rajattu, mutta siinä pyydettiin kiinnittämään huomiota toiminnan luotettavuuteen ja vikasietoisuuteen. Työn tekeminen päätettiin aloittaa tekemällä vaatimusmäärittely, johon varsinaisen toteutuksen tulisi perustua. Vaatimusmäärittelystä tuli keskeinen osa opinnäytetyötä. Toteutus perustui vaatimusmäärittelystä tehtyyn oliokeskeiseen analyysiin ja suunnitteluun, ja valmiin järjestelmän testaus oli lähtökohtaisesti vaatimusmäärittelyssä kirjattujen vaatimusten testaamista.

Opinnäytetyö kokonaisuudessaan vastasi mielestäni hyvin sille asetettuja tavoitteita. Toteutettu järjestelmä on vaatimusten mukainen ja toimiva. Kehitystyön aikana pystyin mielestäni hyödyntämään monipuolisesti opittuja ohjelmistosuunnittelun taitoja ja tekniikoita. Lisäksi pystyin hyödyntämään aikaisempaa osaamistani, erityisesti Linux-palvelinjärjestelmän määrittelyssä ja hallinnassa.

LÄHTEET

1. Tekniset esitteet. Team-Control Oy. Saatavissa: https://www.team-control.fi/tiedostot. Haettu 24.6.2017.

2. TiiMi 7120B Ilmanvaihdon säädin. käsikirja. 2013. Saatavissa: https://team-control.fi/uudet/wp-content/uploads/ilman-vaihto/7120B/tiimi_7120B_4.20_kasikirja.pdf. Haettu 24.6.2018.

3. Mattila, Jarno. 2017. Yrityslähtöinen tuotekehitysprojekti - Lämmityssäätimen etäkäyttöliittymän kehittäminen, loppuraportti. Oulun ammattikorkeakoulu. Syksy 2017.

4. TiiMi GSM-Set GSM/GPRS-modeemiterminaali. Team-Control Oy. Saatavissa: https://www.team-control.fi/etakaytto/. Haettu 24.6.2018.

5. McConnell, Steve. 2004. Code Complete 2nd Edition. Redmond, Washington, USA: Microsoft Press.

6. Hertzog, Raphaël –Mas, Roland. 2015. The Debian Administrator's Handbook. Sorbiers, France: Freexian SARL.

7. Linux (ydin). 2018. Wikipedia. Saatavissa: https://fi.wikipedia.org/wiki/Linux_(ydin). Haettu 21.7.2018.

8. Systemd – system and service manager. 2018. Debian Wiki. Saatavissa: https://wiki.debian.org/systemd. Haettu 5.8.2018.

9. Logrotate(8). 2002. Linux manuaali. Saatavissa: https://linux.die.net/man/8/logrotate. Haettu 5.8.2018.

10. Dent, Kyle D. 2003. Postfix – The Definitive Guide. Sebastopol, California, USA: O'Reilly.

11. About Us. 2018. MariaDB. Saatavissa: https://mariadb.com/about-us. Haettu: 7.8.2018.

12. Beaulieu, Alan. 2009. Learning SQL, Second Edition. Sebastopol, California, USA: O'Reilly.

13. PHP: Hypertext Preprocessor. Dokumentaatio. 2018. Saatavissa: http://fi2.php.net/get/php_manual_en.html.gz/from/this/mirror. Haettu 16.8.2018.

14. Usage of server-side programming languages for websites. 2018. W3Techs. Saatavissa: https://w3techs.com/technologies/overview/programming_language/all/. Haettu: 16.8.2018.

15. Aley, Rob. 2016. PHP CLI: Create Command Line Interface Scripts with PHP. Oxford, United Kingdom: Apress.

16. IEEE Recommended Practice for Software Requirements Specifications. 1993. IEEE Standards Board. NewYork, USA: The Institute of Electrical and Electronics Engineers, Inc.

17. Larman, Craig. 2004. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition: Addison Wesley Professional.

18. Fowler, Martin – Scott, Kendal. 2002. UML. Jyväskylä: Docendo Finland Oy.

19. About Geany. 2010. Geany. Saatavissa: https://www.geany.org/Main/About. Haettu: 26.10.2018.

20. A Quick Start Guide to TortoiseHg. 2018. TortoiseHg. Saatavissa: https://tortoisehg.readthedocs.io/en/latest/quick.html. Haettu: 26.10.2018.

21. Introducing JSON - JSON, Saatavissa: https://www.json.org. Haettu: 27.10.2018.

22. The JSON Data Interchange Syntax. 2017. Standard ECMA-404 2nd Edition: Ecma International.

23. Inside DocBlocks - phpDocumentor. Saatavissa: https://docs.phpdoc.org/guides/docblocks.html. Haettu: 27.10.2018.

24. Supported Versions . 2018. PHP. Saatavissa: https://secure.php.net/supported-versions.php. Haettu: 28.10.2018.

25. Migrating from PHP 5.6.x to 7.0.x. PHP Manual, Appendices. Saatavissa: https://secure.php.net/manual/migration70.php. Haettu: 28.10.2018.

26. Variables. 2018. PHP Manual, Language Reference. Saatavissa: https://secure.php.net/manual/en/language.variables.basics.php. Haettu: 29.10.2018.

27. Type Juggling. 2018. PHP Manual, Language Reference, Types. Saatavissa: https://secure.php.net/manual/en/language.types.type-juggling.php. Haettu: 29.10.2018.

28. Class Abstraction. 2018. PHP Manual, Language Referense, Classes and Objects. Saatavissa: https://secure.php.net/manual/en/language.oop5.abstract.php. Haettu: 29.10.2018.

29. Yadav, Rahul. 2015. What is the difference between stubs and drivers in software testing? Saatavissa: https://www.quora.com/What-is-the-difference-between-stubs-and-drivers-in-software-testing. Haettu: 29.10.2018.

30. Pressman, Roger S. 2001. Software Engineering - a Practitioner's Approach, 5th Edition. New York, USA: The McGraw-Hill Companies, Inc.

31. Mail. 2018. PHP Manual, Function Reference, Mail Related Extensions, Mail, Mail Functions. Saatavissa: https://secure.php.net/manual/en/function.mail.php. Haettu: 29.10.2018.

32. Definition of the CSV Format. 2005. IETF RFC 4180. Saatavissa: https://tools.ietf.org/html/rfc4180#section-2. Haettu: 1.11.2018.

33. What is jQuery? 2018. jQuery. Saatavissa: https://jquery.com. Haettu: 1.11.2018.

34. Bootstrap - The most popular HTML, CSS, and JS library in the word. Saatavissa: https://getbootstrap.com. Haettu: 1.11.2018.

35. DataTables - Table plug-in for jQuery. 2018. SpryMedia Ltd. Scotland. Saatavissa: https://datatables.net. Haettu: 1.11.2018.

36. Asynchronous JavaScript + XML - Ajax. 2018. MDN web docs. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX. Haettu: 1.11.2018.

37. Rsyslog. 2018. Debian Wiki. Saatavissa: https://wiki.debian.org/Rsyslog. Haettu: 3.11.2018.

38. Systemd.timer - Timer unit configuration. 2018. Debian manual. Saatavissa: https://manpages.debian.org/strech/systemd/systemd.timer.5.en.html. Haettu: 3.11.2018.