Teemu Herttua

Skaalautuva Hälytysjärjestelmä

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikan koulutusohjelma
Toukokuu 2020

Centria AMMATTIKORKEAKOULU

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Toukokuu 2020	**Tekijä/tekijät** Teemu Herttua
Koulutusohjelma Tieto- ja viestintätekniikka		
Työn nimi Skaalautuva Hälytysjärjestelmä		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 66
Työelämäohjaaja Juho Jaakola		

Tämä opinnäytetyö on tehty Kyynel Oy:lle Oulussa. Yritys tarjoaa tiedonsiirtoratkaisuja merenkulkuun. Opinnäytetyö koostuu järjestelmän suunnittelusta, sen kehityksestä ja testauksesta. Työ tehtiin tuotevaatimuksen pohjalta, ja tavoitteena oli kehittää jo olemassa olevaa hälytysjärjestelmää eteenpäin Kyynel Oy:n tarpeiden mukaiseksi.

Kehitettävän hälytysjärjestelmän tuli seurata radion laitteiston arvoja ja palveluiden tilaa moniajoa hyväksikäyttäen. Mahdollisista virheistä tuli luoda hälytysviesti ja tallentaa näiden virhesanomien tiedot. Hälytysten tiedot tuli myös tallentaa tietokantaan.

Kehitettävän hälytysjärjestelmän tuli myös olla skaalautuva niin, että uusia seurattavia palveluita ja arvoja voitaisiin ottaa käyttöön ilman koodin muuttamista. Järjestelmä tuli myös kehittää niin, että täysin uuden tyyppisten ominaisuuksien seuraamisen lisääminen hälytysjärjestelmään onnistuisi ilman suurta muutosta koodiin.

Opinnäytetyössä tehty ohjelmisto on yrityksen verkon laadun ja sen hallinnan kannalta hyvä lisä. Vaatimusmäärittelyssä esitetyt asiat tulivat toteutettua, ja työ onnistui hyvin. Ohjelmistoon tuli uusia ominaisuuksia, kuten laitteiston eri arvojen seuraaminen. Työ mahdollistaa radioiden paremman etävalvonnan, ja ohjelmiston käyttämisestä tuli helpompaa.

Asiasanat
Docker, hälytysjärjestelmä, korkeataajuus, Linux, moniajo, ohjelmistosuunnittelu, ohjelmistotestaus, Postgresql, Python, rengaspuskuri, skaalautuvuus, verkonlaatu, virtuaalinen ympäristö

Centria UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Centria University of Applied Sciences	**Date** May 2020	**Author** Teemu Herttua
Degree programme Information technology		
Name of thesis Scalable Alert System.		
Instructor Sakari Männistö		**Pages** 66
Supervisor Juho Jaakola		

This thesis was commissioned by Kyynel Ltd. and made in Oulu. The company provides data transfer solutions for the maritime industry. The thesis consists of the design, development and testing of a system. The project was made based on a product requirement and the goal was to further develop an existing alert system for Kyynel Ltd's needs.

The new alert system was required to monitor hardware-related values and the state of various services concurrently. The system was also required to generate an alert for any abnormality found in the monitored values or services. The details of these alerts were required to be stored in a database as well.

The new alert system was also required to be scalable so that it would support various new features which would need monitoring with as little modifications of the code as possible. Adding the means to monitor a new hardware-related value or service state was required to be possible without any change in the code.

The program produced for this thesis improves network management and network quality. The product requirement was met and the outcome of this thesis was successful. New features such as monitoring hardware values were added. The program futher improves supervision of the radios and is now easier to use.

Key words
alert service, concurrency, Docker, high frequency, Linux, network quality, Postgresql, Python, ring buffer, scalability, software development, software testing, virtual environment

KÄSITTEIDEN MÄÄRITTELY

AIS	Eräs merenkulkuun tarkoitettu seurantajärjestelmä
CALLBACK	Vastakutsuna tapahtuva metodi tai funktio
CRON	Määritetyn ajan välein tapahtuva toistorutiini
CSV	Tiedostomuoto, jossa arvot esitetään pilkuilla eroteltuina
DAEMON	Taustaohjelma
DOCKER	Virtuaalisten säiliöiden hallintaohjelma
DOCKER-SÄILIÖ	(Docker container) Virtuaalinen säiliö, joka sisältää vain ne käyttöjärjestelmän osat, joita tarvitaan sovelluksen ylläpitoon
DOCKER-COMPOSE	Monen Docker-säiliön hallinnointiohjelma
ERP	(Enterprise Resource Plannin) Toiminnanohjausjärjestelmä
GENERAATTORI	Python-ohjelmointikielen iteraattorimetodi
GREENLET	Vuorottaisrutiini
HF	(High Frequency) Korkeataajuus
HTTP	(Hyper Text Transfer Protocol) Yhteyskäytäntö
JSON	(JavaScript Object Notation) Dataformaatti
JQUERY	Javascript-ohjelmointikielen kirjasto
IOT	(Internet of Things) Esineiden internet
LINUX	Käyttöjärjestelmä
METODI	Luokkaan kuuluva aliohjelma tai proseduuri
MODUULI	Määritelmät ja lausekkeet sisältävä kooditiedosto
POSTGRESQL	Relaatiotietokannan hallintaohjelma
PYTHON	Yleisohjelmointikieli
PSYCOPG2	Python-ohjelmointikielen sovitin PostgreSQL-relaatiotietokantaa varten
SEMAFORI	Muuttuja, joka osoittaa resurssinkäyttötilaa
SKRIPTI	Sarja sekvenssinä suoritettavia komentoja
SWR	(Standing wave ratio) Radioaallon takaisinheijastuksen määrä
TARKKALIJA	Tässä opinnäytetyössä käytetty nimitys ”observer”-luokan olioinstanssista

TIIVISTELMÄ
ABSTRACT
KÄSITTEIDEN MÄÄRITTELY
SISÄLLYS

1 JOHDANTO ... 1

2 KYYNEL OY ... 3
2.1 Yritys ... 3
2.2 Radio ja verkko ... 4
2.3 Tuotteet ja palvelut ... 6
2.4 Ylläpito ... 9

3 PROJEKTI ... 10
3.1 Vaatimusmäärittely ... 10
3.2 Tavoitteet ja rajaus ... 11
3.3 Kehitysympäristö ja työkalut ... 12

4 LÄHTÖTILANNE ... 14
4.1 Toiminta ... 14
4.2 Hälytyksen luonti ja lähetys ... 15
4.3 Hälytyksen vastaanotto ... 16
4.4 Koodin uudelleenkäytettävyys ... 17
4.5 Uudet kokonaisuudet ... 17

5 TOTEUTUS ... 18
5.1 Arkkitehtuuri ... 18
5.1.1 Alert Service 2 esittely ... 19
5.1.2 Tietokanta ... 21
5.1.3 Tiedostohierarkia ... 22
5.2 Ratkaisut ... 24
5.2.1 Palveluiden ja levytilan seuraaminen ... 24
5.2.2 HW-arvojen seuraaminen ... 25
5.2.3 Takaisinheijastuksen seuraaminen ... 26
5.2.4 ”Collect”-palvelun seuraaminen ... 26
5.2.5 Rengaspuskuri ... 27
5.2.6 Rengaspuskurin toiminta ... 28
5.2.7 Sarjallistaminen ... 29
5.3 Turvallisuuden huomiointi ... 32

6 TOIMINTA ... 34
6.1 Lähetyskomponentti ... 35
6.1.1 Alert Sender -moduuli ... 36
6.1.2 Scheduler-moduuli ... 37
6.1.3 Observer-moduuli ... 38
6.1.4 Alert generator-moduuli ... 44
6.2 Vastaanotto palvelimella ... 45
6.2.1 Alert receiver-moduuli ... 45
6.2.2 Alert handler-moduuli ... 46

6.2.3 Database handler-moduuli ... 46
6.3 Kokoonpanoasetukset ... 48

7 TESTAUS ... 49
7.1 Yksikkötestit ... 50
7.1.1 Kokoonpanoasetukset ja niiden hallinta ... 50
7.1.2 Hälytysviestin käsittely ... 51
7.1.3 Tarkkailija ... 54
7.2 Integraatiotestit ... 56
7.2.1 Aloitusasettelu ja purkaminen ... 57
7.2.2 End-to-end ... 58
7.2.3 Ei hälytyksiä ... 58
7.2.4 Epämääräinen kokoonpanoasetus ... 58
7.2.5 Vähäinen levytila ... 59
7.2.6 Yhteys katkeaa ... 59

8 YLLÄPITO ... 61

9 JATKOKEHITYS ... 62
9.1 Uudet hälytykset ... 62
9.2 Uuden tyyppiset hälytykset ... 63
9.3 Uudet ominaisuudet ... 63

10 YHTEENVETO ... 65

LÄHTEET ... 67

KUVAT

KUVA 1. Mesh Network ... 5
KUVA 2. KNL Mail ... 7
KUVA 3. KNL FILE ... 7
KUVA 4. KNL Track ... 8
KUVA 5. KNL WaveAccess COLLECT ... 9
KUVA 6. Alert Service 1 ... 14
KUVA 7. Alert Service 2 ... 20
KUVA 8. Tiedostohierarkia ... 22
KUVA 9. Testien tiedostohierarkia ... 23
KUVA 10. Rengaspuskuri ... 29
KUVA 11. Viestin rakenne ... 30
KUVA 12. Sarjallistamisen vaiheet ... 31
KUVA 13. Sarjallistamisen poistaminen ... 32
KUVA 14. Scheduler-luokan toiminta ... 38
KUVA 15. Levytilan tarkkailu ... 39
KUVA 16. Palvelun tarkkailu ... 39
KUVA 17. Laitteiston tarkkailu ... 40
KUVA 18. SWR-arvojen tarkkailu ... 41
KUVA 19. Collect-palvelun tarkkailu ... 42
KUVA 20. Hälytysten tallentaminen ja historian kopiointi ... 44
KUVA 21. Tietokantaan upottaminen ... 47

TAULUKOT

TAULUKKO 1. Moduulien kuvaus....15
TAULUKKO 2. Uusien moduulien kuvaus....21
TAULUKKO 3. Tietokannan esittely....21
TAULUKKO 4. Tarkkailijoiden pääluokat....24
TAULUKKO 5. Tarkkailijoiden asetukset....35
TAULUKKO 6. Hälytyskoodit....43
TAULUKKO 7. Testiviestit 1....51
TAULUKKO 8. Testiviestit 2....52
TAULUKKO 9. Testiviestit 3....53
TAULUKKO 10. Testitarkkailijoiden määritykset 1....54
TAULUKKO 11. Testitarkkailijoiden määritykset 2....57

1 JOHDANTO

Tämä opinnäytetyö on tehty Oulussa Kyynel Oy:lle vaatimusmäärittelyn pohjalta. Työn tarkoituksena on edistää verkonhallinnan automatisointia. Opinnäytetyön tarkoituksena oli tutkia, miten jo olemassa olevaa hälytysjärjestelmää voisi parantaa vaatimusmäärittelyn mukaiseksi. Koska hälytysjärjestelmän toiminta muuttui paljon, koko arkkitehtuuria täytyi muuttaa ja kehittää uusiksi.

Opinnäytetyössä esitellään aluksi yritys ja sen tuoma lisäarvo merenkulkua ajatellen. Sitten avataan radioaaltoteknologiaa ja yrityksen käyttämää korkeataajuusaaltoa (HF). Seuraavaksi esitellään, miten yritys käyttää tätä HF-teknologiaa, ja esitellään yrityksen kehittämiä palveluita. Verkonlaadun parantamisen lähtökohdat käydään läpi, mistä päästään tämän työn olemassaolon oikeutukseen. Työ on tärkeä verkonlaadun edistämisen kannalta, ja etävalvonta on hyvä seikka myös markkinoinnin kannalta.

Työssä tarkastellaan tuotteen koko elinkaarta, joten siitä tuli laaja. Työ sisältää suunnitteluvaiheen, ratkaisujen etsimisen, tuotteen kehittämisen, tuotteen testaamisen sekä jatkokehityksen pohdinnan. Työn keskeisiä tavoitteita päätoiminnallisuuden lisäksi oli löytää keinot ja suunnitella helposti laajennettava ohjelma, joka toimii rinnan ajona. Sovelluksen päätoiminnallisuus on seurata tiettyjä radion toimintaan liittyviä parametrejä ja palveluita ja kirjoittaa rengaspuskurin tapaista historiaa. Toiminnallisuuteen kuului hälytyksien luominen, lähetys ja käsittely. Nämä hälytyksiin liittyvät asiat käydään tarkemmin läpi myöhemmin tässä opinnäytetyössä. Kehityksessä tuli myös ottaa huomioon ohjelman turvallisuus.

Työn ensimmäisissä luvuissa uuteen hälytysjärjestelmään liittyviä ratkaisuja pohditaan ja käydään läpi. Ohjelman uusi arkkitehtuuri ja muu toteutus esitellään, jonka jälkeen käydään läpi, miten uuden järjestelmän lähetys ja vastaanotto tapahtuvat. Järjestelmän eri osien toiminta käydään syvällisemmin läpi.

Myös tuotteen testausvaihe on kuvattu. Tuotteelle kehitettiin yksikkö- ja integraatiotestit havaitsemaan mahdolliset virheet. Testausta käsittelevässä kappaleessa testausperiaatteet tuodaan esille ja esitellään mitä testataan, minkä jälkeen kehitettyjen testien toiminnallinen kulku avataan.

Ylläpitoa koskevassa luvussa on esitetty tuotteen hallinnointiin liittyvät asiat. Näitä ovat esimerkiksi tuotteen käyttö ja ne asiat, jotka vaativat toimenpiteitä tuotteen käyttöönoton jälkeen. Jatkokehitys on otettu esille opinnäytetyön lopussa, ja siinä esitellään, kuinka uusia palveluita, tai muita

tarkkailutoimenpiteitä tarvitsevia asioita, otetaan järjestelmän seurantaan. Luvussa pohditaan myös muita mahdollisia ominaisuuksia, joita järjestelmään voi tulevaisuudessa kehittää.

Viimeiseksi yhteenvedossa pohditaan, missä määrin työn tavoitteet täytettiin, ja käydään läpi työn puutteet. Yhteenvedossa myös pohditaan, miten työ eteni, mitä ongelmia vastaan tuli ja miten ne selvitettiin.

Kirjallisuutena tässä työssä käytettiin eri teknologioiden dokumentointeja. Muita kirjallisuuksia olivat esimerkiksi eri tahojen tekemät raportit ja tutkimukset. Kyynel Oy:n sisäisiä materiaaleja käytettiin myös paljon hyödyksi. Lähteinä olivat myös tietotekniikan alan kirjallisuus ja alaan liittyvät artikkelit. Suurin osa työssä käytetystä kirjallisuudesta oli englanninkielistä.

2 KYYNEL OY

Merenkulku on tärkeä osa maailmantaloutta, ja laivakuljetuksen osuus on 90 % kaikesta maailman logistiikasta. Koska ala on suuri, voisi kuvitella sen olevan pitkälle kehittynyt kaikilla osa-alueilla, mutta asia on toisin; merenkulku ottaa hitaasti käyttöönsä uusia menetelmiä, minkä vuoksi sen modernisointi on hidas prosessi (Berg et al. 2019).

Merenkulun modernisointi on kuitenkin tärkeä askel niin laivojen henkilökunnan, yritysten kuin ympäristönkin kannalta. Suurimmat innovaatiota eteenpäin vievät seikat ovat turvallisuus, tehokkuus ja ympäristöystävällisyys (Berg et al. 2019). Jokaisen mainitun alueen edistämiseksi ajantasaisen tiedon saaminen on tärkeä asia.

Ympäristöystävällisyyden parantamiseksi merenkulkuun liittyvä data pitää saada helposti saatavaksi, jotta se voidaan analysoida nopeasti. Laivat voivat olla satamassa pitkäänkin toimettomana (Berg et al. 2019). Laivojen tullessa satamaan ankkurointiaikaa ja ankkuripaikkojen varauksien hallintaa edistää tieto siitä, missä laiva kulkee. Kun laivojen kommunikoinnista tulee nopeampaa tukikohdan ja muiden laivojen välillä, tehokkuuden ohella myös turvallisuus paranee.

Yritykselle on ajantasaisen tiedon saatavutta ajatellen kehitetty erilaisia palveluja, jotka toimivat hyödyntäen yrityksen korkeataajuusradiota. Yrityksellä on vikasietoinen verkko, jota kehitetyt palvelut käyttävät ja jonka tuloksena merenkulun tietoliikenne saadaan digitalisoitua.

2.1 Yritys

Yritys sai alkunsa vuonna 2011, kun yrityksen kolme perustajaa etsivät uusia käyttökohteita HF-taajuuksille. Markkinapotentiaalia nähtiin ensin tuvallisuuspuolella, jonka jälkeen merenkulun tietoliikenteessä nähtiin myös potentiaalia HF-taajuuksille.

Satelliittiyhteys on yleisin tietoliikenneväylä merenkulussa (Berg et al. 2019). Satelliittiyhteydessä on kuitenkin joitain huonoja puolia, joista esimerkkeinä on sen hitaus tietyissä paikoissa ja hinta (Mällinen 2018). Yleisyys on myös huono asia, sillä satelliittitietoliikenteellä on riski ruuhkautua.

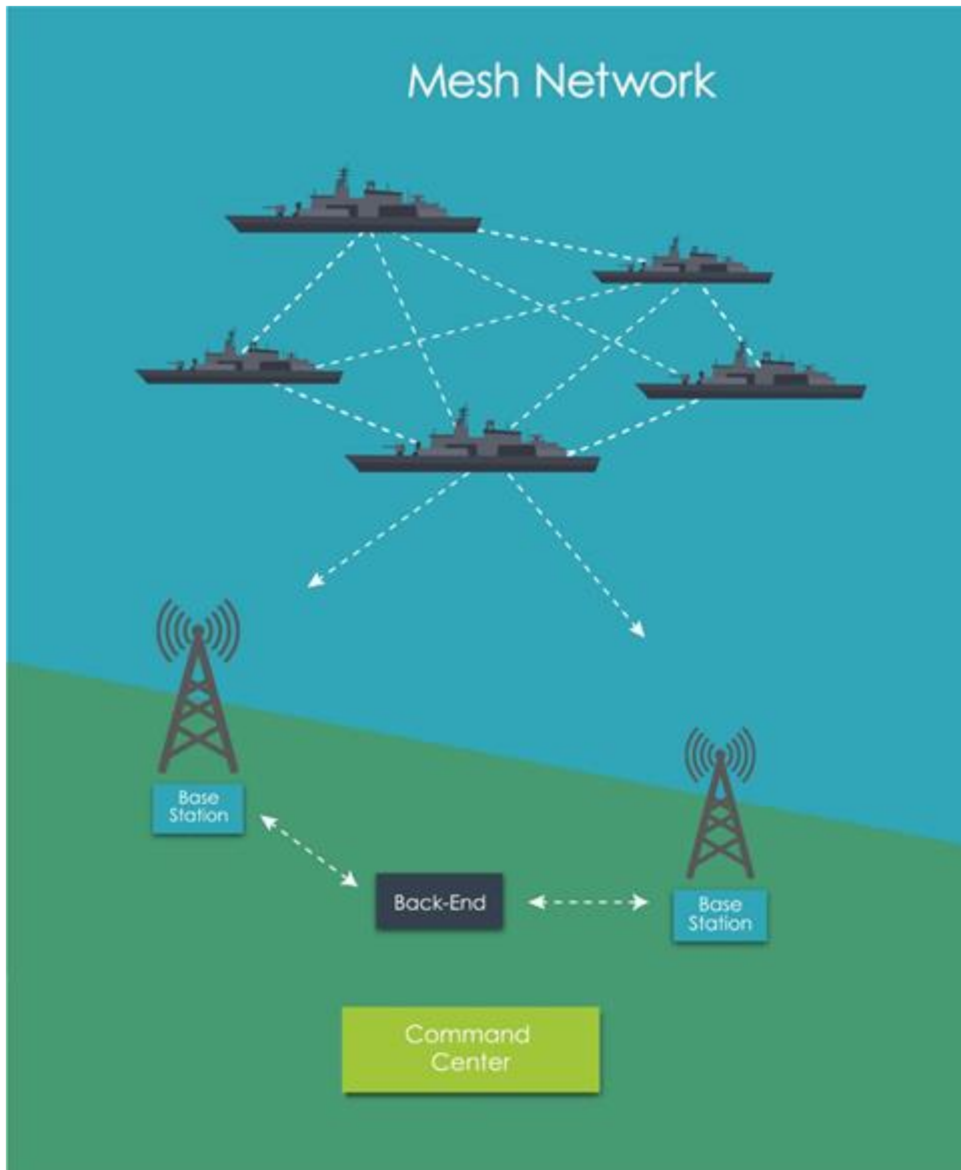
Kyynel Oy tuo innovaationsa merenkulun viestintään hyödyntäen omaa kolmansista osapuolista riippumatonta HF-radiota. Kyynel Oy:n kehittämän radion tarkoitus ei siis ole syrjäyttää käytössä olevaa satelliittiviestintää vaan nykyaikaistaa merenkulun viestinvälitys tuomalla satelliittien rinnalle toinen tiedonvälityskanava erilaisine ominaisuuksineen. (KNL Networks 2019a).

Kyynel Oy:n pääasialliset asiakaskohderyhmät ovat merenkulkualan yritykset. Kuten aiemmin mainittu, merenkulkualan suurimpina haasteina ovat kulujen minimointi, päästöjen vähentäminen ja turvallisuuden lisääminen. Aluksen suorituskyvyn datan saaminen analysoitavaksi voi viedä pitkään, sillä suurta määrää dataa on vaikea siirtää ja se voi olla mahdoton siirtää ilman henkilöstöä. Kyynel Oy pyrkii ratkaisemaan ongelman tuomalla automaattisen ja nopean tavan siirtää esimerikiksi aluksien suorituskykyyn liittyvää dataa alukselta tukiasemalle reaaliajassa.

2.2 Radio ja verkko

HF-signaalin käyttö ei itsessään ole uusi keksintö, ja sitä on käytetty esimerkiksi horisontin taakse katsovissa tutkissa (engl. Over The Horizon) (Fabrizio 2013). Kyynel Oy:n HF-verkko eroaa siten aiemmista HF-verkon käyttötavoista, että Kyynel Oy:n verkko on häiriösietokykyisempi ja kaikki välitetty data on salattua.

Kyynel Oy:n tiedonsiirto toimii mesh-verkko periaatteella, eli monen laivan radiot yhdessä muodostavat kattavan verkon. Tämä verkko yhdistyy sitten tukiasemien kautta palvelimeen, tietokantoihin ja muuhun infrastruktuuriin. Radio voi toimia tukiasemana, jos se on yhteydessä matkapuhelinverkkoon. Kun radiota kantava alus on mobiiliverkon ulottumattomissa, esimerkiksi merellä, radio toimii vain päätepisteenä huolehtien vain omasta tietoliikenteestään. Alla oleva kuva havainnollistaa mesh-verkkoa.



Kuva 1: Mesh Network (Mukailtu lähteen KNL Networks 2019a mukaan)

Korkeataajuussignaaliksi (engl. high frequency) luokitellaan radioaallot väliltä 3 ja 30 MHz. Tätä pienemmät taajuudet, eli alle kolmen megahertsin taajuudet, luokitellaan matalataajuisiksi (engl. low frequency) ja tästä suurempia taajuuksia kutsutaan erittäin korkeiksi taajuuksiksi (engl. very high frequency). Matalataajuudet, jotka ovat taajuudeltaan 1,5 megahertsiä ja alle, ovat varattu hätäviestintään (Giesbrect, 2008). Kyynel Oy:n radio toimii 1.5:n ja 30 MHz:n välillä, eli spektrin matalimmat taajuudet voidaan luokitella matalataajuuksiksi, mutta radio toimii pääasiallisesti korkeataajuuden, eli HF:n, rajoissa, ja hätäviestinnälle varattuja taajuuksia ei käytetä lainkaan.

Matalataajuussignaalit kantavat kaikkein pisimmälle kaartuen maapallon mukaan, ja ne yltävät lähes maapallon toiselle puolelle ennen liiallista vaimenemista. Erittäin korkean taajuuden signaalit puolestaan

eivät kaarru ollenkaan ja menevät ilmakehän läpi. Korkeataajuussignaalit voivat kimpoilla ilmakehän ionosfääristä, jonka vuoksi niiden kantama vaihtelee paljon. Kantama voi olla viidestä kilometristä tuhansiin kilometreihin (Giesbrect, 2008). Ionosfääri taittaa korkeataajuus signaaleita riippuen auringon säteilyn voimakkuudesta, minkä takia korkeataajuuden eri taajuuksien kantamiin vaikuttaa myös olosuhteet (Space Weather Services, 2016)

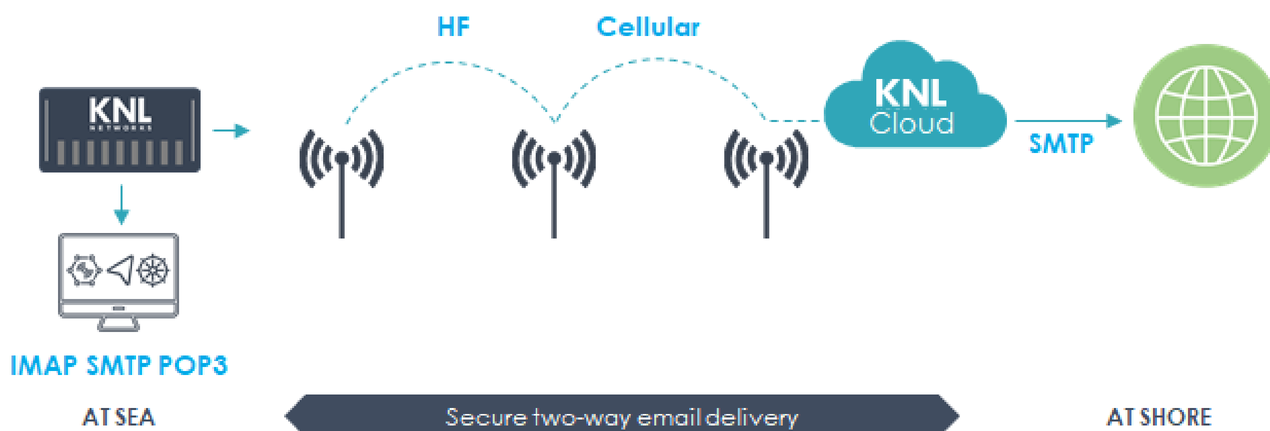
Cognitive Networked HF-radio (CNHF) on Kyynel Oy:n kehittämä radio, joka valitsee optimaalisen taajuuden olosuhteiden mukaan. Oikean taajuuden valitseminen parantaa radion kantamaa, sillä ionosfäärin kykyyn taittaa korkeataajuusaaltoja vaikuttavat gamma-, röntgen- ja ultraviolettisäteily (Space Weather Services, 2016). KNL Networks on Kyynel Oy:n aputoiminimi.

2.3 Tuotteet ja palvelut

Yrityksen päätuotteet ovat HF-radion mahdollistamat palvelut, joita on kehitetty kattamaan asiakkaiden tarpeet. Pääpalvelut ovat KNL Mail, WaveAccess File, WaveAccess Track ja WaveAccess Collect. Yritys myös kehittää ratkaisuja asiakaskohtaisiin ongelmiin käyttäen radion toiminnallisuutta. Asiakkaalle toimitetaan laitteet, ja palveluita myydään kuukausihintaan. Tiedonsiirrosta yritys laskuttaa käytön mukaan.

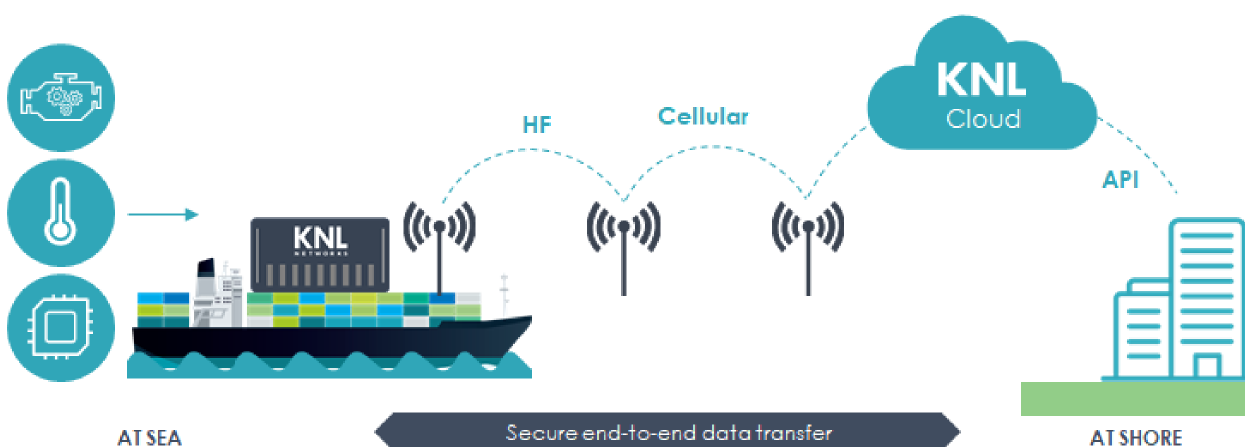
Yritys tarjoaa sähköpostipalvelua sähköpostien lähetystä varten, ja se toimii HF-signaalien ja mobiiliverkon avulla. Sähköpostipalvelun avulla voidaan lähettää sähköposteja, ja se käyttää tavallisimpia sähköpostiprotokollia, kuten SMTP, POP3 ja IMAP (KNL Mail Product Sheet, 2019). Sähköpostit ovat oletusarvoltaan KNL Networksin oman verkkotunnuksen alla, mutta asiakas voi halutessaan itse määrittää haluamansa verkkotunnuksen.

Kuten kaikki tietoliikenne, myös sähköpostit on salattu AES-256 salauksella. (KNL Networks 2019c) Kuvassa 2 on mallinnettu sähköpostipalvelun toiminnan kulku.



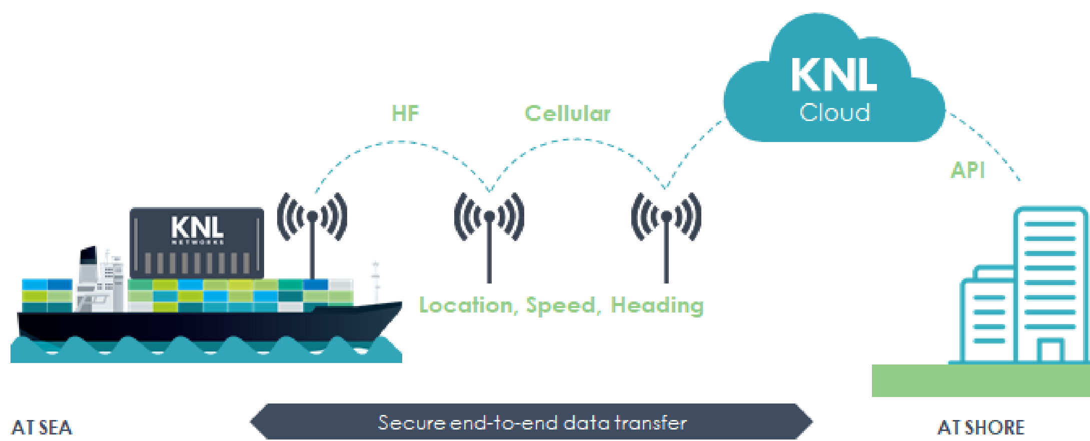
Kuva 2: KNL Mail (KNL Networks 2019c mukaan)

Data-agnostiseen tiedonsiirtoon on kehitetty WaveAccess FILE-palvelu, jonka avulla voidaan dataa lähettää salattuna kaksisuuntaisesti HF-verkon yli. Palvelun avulla voidaan lähettää tiedostot alukselta tukiasemalle sijainnista huolimatta. Palvelussa on myös mahdollisuus määrittää prioriteetti, jonka avulla kriittinen data voidaan lähettää ensimmäisenä. (KNL Networks 2019b) Kuvassa 3 on esitetty tiedostonlähetyspalvelun toiminnan kulku.



Kuva 3: KNL FILE (KNL Networks 2019b mukaan)

Aluksia voidaan seurata reaaliajassa KNL Networksin WaveAccess Track-paikannusjärjestelmän avulla, joka toimii myös siellä, johon Automatic Identification System (AIS) ei ulotu. Paikannusjärjestelmä voi lähettää dataa tiheimmillään minuutin tarkkuudella KNL Networksin pilvipalveluun, josta viimeisimmät sijaintitiedot voidaan hakea (KNL Track ProductSheet). Palvelu tallentaa myös viimeaikaisen datan, jonka avulla voidaan tarkasti seurata aluksen kulkemista. Sijaintitietojen raakadata voidaan ohjelmarajapinnan kautta ottaa käyttöön, tai sijaintia voidaan tarkastella KNL Networksin käyttäjäportaalissa, jossa data on visualisoitu kartan muotoon (KNL Networks 2019e). Palvelun toimintaa on kuvattu kuvassa 4.



Kuva 4: KNL Track (KNL Networks 2019e mukaan)

”WaveAcces COLLECT” on suunniteltu lähettämään IOT-dataa HF-verkon ylitse. Datan keräämiseen voidaan käyttää aluksen omia laitteita, jotka ovat yhteydessä Collect-palveluun. Näiden palveluiden data lähetetään KNL Networksin pilvipalveluun, ja asiakas pääsee tärkeään dataan käsiksi reaaliajassa. Data lähetetään mesh-verkon yli ja tiedonsiirrossa käytetään tavallisimpia tiedonsiirtoprotokollia sekä yrityksen omaa KNL File protokollaa (KNL Networks 2019f). Alla oleva kuva esittää palvelun toiminnan kulkua.

Kuva 5: KNL WaveAccess COLLECT (KNL Networks 2019f mukaan)

2.4 Ylläpito

Asiakkaiden tyytyväisyyden takaamiseksi ja tarpeiden täyttämiseksi verkkoa ja itse radiota parannetaan, kehitetään ja optimoidaan jatkuvasti jokaisella osa-alueella. Verkon ja sen hallinankehittämistä varten radio tallettaa ja lähettää dataa eri toiminnoistaan ja suorituskyvystään KNL Networksin tietokantaan analysoitavaksi. Laaduntarkkailua varten signaalin ja tiedonsiirtoon liittyvän datan lisäksi myös radion ohjelmiston toimivuudesta kerätään tietoa.

Verkonhallinnan ja laaduntarkkailun automatisointi on tärkeää, jotta palveluita ja verkkoa voidaan kehittää eteenpäin. Radion ohjelmiston toimivuutta valvotaan hälytysjärjestelmällä, joka ilmoittaa vähäisestä levytilasta ja mahdollisesta palveluiden toimimattomuudesta tekemällä niistä KNL Networksin ERP-järjestelmään julkaisun. Vaikka hälytysjärjestelmä on tehokas, se on kuitenkin yksinkertainen eikä täytä kaikkia yrityksen tarpeita. Esimerkiksi nykyinen hälytysjärjestelmä ei huomioi radion laitteiston arvojen seuraamista, kuten lämpötilojen ja jännitteiden seuraamista.

3 PROJEKTI

Tämän opinnäytetyön tarkoitus on laajentaa hälytysjärjestelmän ominaisuuksia verkonhallinnan laadun ja tarkkailun parantamiseksi. Työn aloitushetkellä ”Alert Service” nimeä kantava hälytysjärjestelmä generoi vain hyvin yksinkertaisia hälytyksiä kaikista radion eri toiminnosta, joita ovat levytilan kokonaismäärän ja eri palveluiden tarkkailu. Palveluita ovat muun muassa ”File service”, joka esiteltiin aiemmin ensimmäisessä luvussa, ja ”connection manager”, joka vastaa radion yhteyksien hallinnasta.

Järjestelmän hallintaan liittyi myös ongelmia. Esimerkiksi osalla radioista kaikkia palveluita ei ollut otettu käyttöön, mutta hälytysjärjestelmä silti generoi kyseisten palvelujen toimimattomuudesta hälytyksiä.

3.1 Vaatimusmäärittely

Vaatimusmäärittelyssä oli määritelty hälytyksen luonnin ja järjestelmän käyttäytymisen vaatimukset. Järjestelmän tulisi tunnistaa radion viat ja toimintahäiriöt sekä luoda niistä hälytykset. Luotu hälytysviesti tulee myös tallentaa odottamaan sen lähetystä. Hälytyksen tapahtuessa lokihistoria ja asiaankuuluvien, tai muuten hälytykselle oleellisten, arvojen lokihistoriat tuli myös ottaa talteen.

Kehitettävän järjestelmän täytyy suorittaa valvominen rinnakkaisajona. Eri asioille tulee olla säädettävissä olevat seuraamisaikavälit, ja luettu otosdata tallennetaan levylle. Levylle tallentamista varten täytyi kehittää tapa, jolla vanhimman tiedoston päälle kirjoitetaan uusi, kun hakemiston koko saavuttaa tietyn koon. Kehitettävä järjestelmä tulee myös testata hyvin yksikkö- ja integraatiotesteillä, joissa koodin turvallisuuteen ja virhetiloista selviämiseen tuli kiinnittää huomiota.

Laitteiston tarkkailu tuli laajentaa lukemaan laitteiston toiminnan arvoja, kuten jännitettä ja lämpötilaa. Luetuillla arvoilla voi olla eri prioriteetin ylä- ja alaraja. Esimerkiksi lämpötilan varoitustason ylittämisestä täytyi luoda eri hälytys kuin kriittisen tason ylittämisestä. Myös liian nopeasta noususta ohjelman tuli luoda eri hälytys.

Hälytyksen laukeamisen yhteydessä tuli olla ratkaisu estämään uuden hälytyksen luominen samasta aiheesta tietyn aikavälin aikana. Laitteiston arvojen nopean muutoksen tarkkailussa täytyi olla

määritettävissä oleva viive, jolla voi säätää, mistä lähtien arvon muutosta aletaan tarkkailemaan. Viiveiden tarkotuksena on karsia turhia hälytyksiä pois. Esimerkiksi jos radio käynnistetään kylmässä tilassa, laitteisto voi lämmetä sallittua muutosrajaa nopeammin, minkä jälkeen lämpötila voi kuitenkin pysyä sallittujen rajojen sisällä.

Laitteiston toiminnan arvojen lukemisen yhteydessä järjestelmän tuli kirjoittaa historialokia, johon parametrin arvon lisäksi päivämäärä ja aika kirjoitetaan. Historialokin pituuden täytyy olla säädettävissä eri mittaiseksi eri asioille, ja historiassa täytyi olla kiertovuorottelu vanhimman merkinnän pois pudottamiseksi uuden tieltä. Hälytyksen tapahtuessa sen hetkinen historialoki ja olennaisten laitteiston arvojen historialokit tuli tallentaa levylle. Koska ohjelma tallentaa levylle dataa, datan ja historialokien tallentamiseen täytyi myös kehittää ratkaisu, joka estää levytilan liiallisen täyttymisen.

Itse hälytysviestissä tuli olla tarkkailtavan asian arvo, nimi, päivämäärä ja aika, radion id-tunnus sekä lyhyt kuvaus hälytyksestä. Hälytysviestit tulee sarjallistaa ja viesti tulee tallettaa levylle talteen, kunnes se on lähetetty onnistuneesti. Vastaanottopuolella sarjallistaminen täytyi poistaa ja viestin tiedot tuli tallentaa tietokantaan.

Kokoonpanoasetuksissa piti pystyä määrittämään yksityiskohtaisesti tärkeimmät hälytysjärjestelmän toimintaa määrittävät asiat, ja niiden muokkaamisen tuli olla helppoa. Historialokin ja luotujen viestin tallennuspolkujen tuli myös olla määritettävissä. Tarkkailun aikavälin ja hälytyksen aikaviiveiden täytyi myös olla muokattavissa kokoonpanoasetuksissa.

3.2 Tavoitteet ja rajaus

Tässä opinnäytetyössä vastattiin projektin koko elinkaaresta suunnitteluvaiheesta käyttöönottoon asti. Projektin tavoitteena oli tutkia ja suunnitella, miten edellisessä vaatimusmäärittelyssä mainitut ominaisuudet ja toiminnot voitiin toteuttaa. Kartoitetun toteutussuunnitelman pohjalta kehitetään, toteutetaan ja testataan valmis tuote tuotantoa varten.

Hälytysjärjestelmästä tuli luoda vikasietokykyinen ja helposti laajennettava ohjelma. Tavoitteena oli myös luoda järjestelmä niin, että sen ylläpito on yksinkertaista. Projekti suunnitellaan vaatimusmäärittelyn pohjalta, mutta suunnitteluvaiheessa selvitetään myös, jos jokin lisäominaisuus

voisi tuoda järjestelmään lisäarvoa. Projektissa otettiin huomioon turvallisuus ja mahdolliset tietoturvariskit.

Ohjelmaan tuli kaksi pääosaa, jotka ovat lähetys ja vastaanotto. Kummallekin osalle tehtiin asennusskripti. Vastaanottokomponenttia varten tehtiin Docker-tiedosto, joka määrittää kokoonpanon Docker-säiliön sisällön, ja itse vastaanotto-ohjelma ajetaan Docker-säiliössä. Hälytysjärjestelmän määritysasetukset kirjoitetaan jo olemassa olevaan Docker compose-tiedostoon, jossa on määritetty ohjelman riippuvuudet muihin Docker-säiliöihin. Lähetyskomponentti tuli olemaan osa Yocto-käyttöjärjestelmän asennuskuvaa, ja se jää tämän työn ulkopuolelle.

Projektin ulkopuolelle jää ensimmäiseen hälytysjärjestelmään liittyvä ERP-julkaisujen luominen. Hälytyksen lähettämistä varten KNL Networksilla oli jo lähettämistä ja vastaanottamista varten ohjelmarajapinta, jota käytetään tässä projektissa pienillä muutoksilla. Hälytysjärjestelmästä ei myöskään tehty itsenäistä taustalla toimivaa prosessia, sillä radiolla on käytössä taustaprosessien hallinnointiohjelma, johon kehitettävä hälytysjärjestelmä liitettiin.

3.3 Kehitysympäristö ja työkalut

Alkuperäinen hälytysjärjestelmä on kehitetty Linux-ympäristössä ja Linux-käyttöjärjestelmää varten. Uudesta hälytysjärjestelmästä ei tule käyttöjärjestelmä agnostista, sillä kehitettävä järjestelmä tulee vanhan tilalle juuri samaan Linux-pohjaiseen käyttöjärjestelmään, jossa aiempikin hälytysjärjestelmä toimi. Aiemman hälytysjärjestelmän ohjelmointikieli on Python 2.7, ja tällä ohjelmointikielellä toteutettiin myös uusi järjestelmä.

Kehityksessä käytettiin ”Visual Studio Code”-ohjelmaa, joka on Microsoftin kehittämä kevyt lähdekoodieditori Linux, Windows ja macOS käyttöjärjestelmille (Microsoft 2019). Version hallinnassa käytettiin Gitiä, joka on suosittu ja laajasti käytössä oleva versionhallintaohjelma (Git 2019).

Alkuperäisessä hälytysjärjestelmässä käytettiin avolähdekoodista PostgresSQL-relaatiotietokannanhallintajärjestelmää. Koska edellä mainittu tietokannanhallintajärjestelmä on yrityksen yleisessä käytössä muutenkin, sitä käytetään myös uudessa järjestelmässä. Muina tietokannan käyttöön liittyvinä työkaluina käytettiin psql-, pgadmin- ja pgmodeler-ohjelmia. Pgadmin4 on Python- ja jQuery-kielillä kirjoitettu postgresql-hallintajärjestelmän graafinen käyttöliittymä (pgAdmin 2019).

Psql puolestaan on terminaalipohjainen käyttöliittymä tietokannanhallinointiin, ja pgmodeler on PostgreSQL-ohjelmaa varten tehty suunnitteluohjelma. Projektissa käytettiin Psycopg2 Python-moduulia Postgresql-tietokannan adapterina.

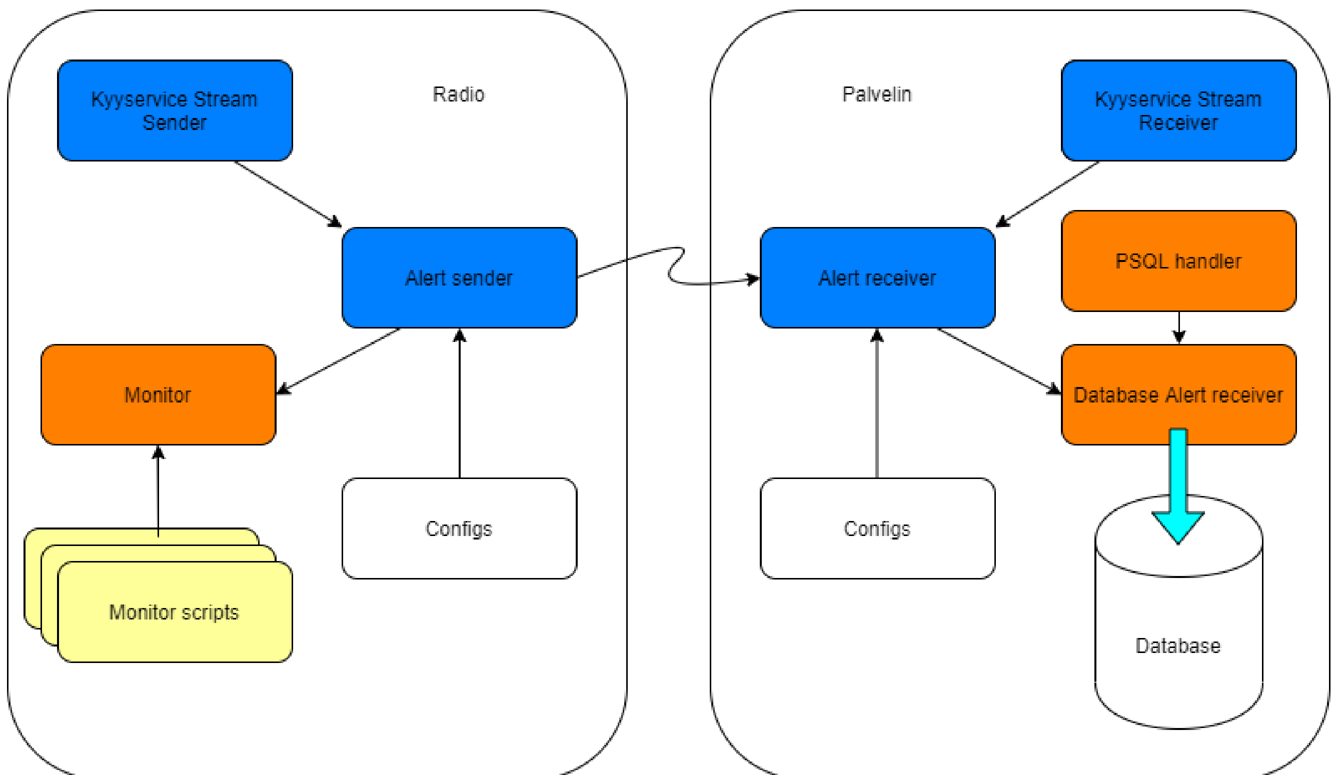
Kuten aiemmassa luvussa jo sivuttiin, tässä projektissa käytetään Docker-ohjelmistoa, jolla voidaan luoda ja hallinnoida Docker-säiliöitä. Näihin Docker-säiliöihin rakennetaan käyttöjärjestelmästä riisuttu virtuaalinen versio, joissa voidaan ajaa muita ohjelmia. Näiden Docker-säiliöiden sisällä pyörivät ohjelmat voivat vaikuttaa toisiin säiliöihin, ja niistä voidaan luoda kokonaisuuksia, tai ne voivat olla niin eristettyjä kuin niistä tehdään

4 LÄHTÖTILANNE

Tässä luvussa käydään läpi ensimmäisen hälytysjärjestelmän toiminta tarkemmin, tarkastellaan rajoitteita ja selvitetään osuus koodista, jota voidaan uudelleen käyttää kehitettävässä hälytysjärjestelmässä.

4.1 Toiminta

Alkuperäinen hälytysjärjestelmä on yksinkertainen ja tehokas. Se on rakennettu toimimaan skriptipohjaisesti ja ajastettu cron-taustaprosessilla, joka on Linux-pohjaisille käyttöjärjestelmille tehty toistorutiinien ajojärjestelmä (Admin's Choice 2019).



Kuva 6: Alert Service 1

Kuvasta 6 voidaan nähdä hälytysjärjestelmän moduulien suhteet toisiinsa. " Alert sender" alustetaan kokoonpanoasetusten mukaisesti, ja cron-taustaprosessi ajaa sen tunnin välein. "Monitor"-moduulissa on määritetty tarkkailuskriptien ajaminen, hälytysviestien sarjallistaminen ja sarjallistuksen poistaminen. "Monitor"-skriptejä on yksitoista, ja Alert sender ajaa kaikki hakemistosta löytyvät skriptit "Monitor"-moduulia käyttäen.

Mahdollinen hälytysviesti lähetetään KNL Networksin kehittämän ”Stream Sender” -lähetyskirjaston avulla vastaanottokomponentille, joka on hallinnointipalvelimella. Tämä vastaanottaja alustetaan käyttäen sille tehtyjä kokoonpanoasetuksia. Vastaanotettu viesti kirjoitetaan tietokantaan ”database alert receiver” -moduulin metodeilla, jotka käyttävät ”Monitor”-moduulia sarjallistamisen poistamiseen. Moduuli ”PSQL Handler” muodostaa viestistä SQL-komennon, jota käytetään viestin tietojen upottamisessa tietokantaan. Taulukossa 1 ovat alkuperäisen hälytysjärjestelmän moduulit listattuna.

Taulukko 1: Moduulien kuvaus

Moduuli	Kuvaus	Missä käytössä
Alert Sender	Järjestelmän lähetysproseduurin alkuskripti, jonka ajamalla muut järjestelmän osat käynnistyvät.	Radio
Sender configs	Lähetyskomponentin alustamisen kokoonpanoasetukset.	Radio
Monitor	Hoitaa tarkkailuskriptien ajon ja hälytysviestien käsittelyn.	Radio ja pavelin
Monitor script	Määrittää palvelun tarkkailun.	Radio
Stream Sender	KNL:n lähetysmoduuli. Lähettää dataa HF:n yli.	Radio
Stream Receiver	KNL:n vastaanottomoduuli. Vastaanottaa HF:n yli lähetetyn datan	Palvelin
Alert Receiver	Järjestelmän vastaanottoproseduurin alkuskripti, jonka ajamalla muut järjestelmän osat käynnistyvät.	Palvelin
Receiver Configs	Lähetyskomponentin alustuksen kokoonpanoasetukset	Palvelin
Database Alert Receiver	Liittää tietokannanhallintamoduulin ”PSQL handler” vastaanottokomponenttiin.	Palvelin
PSQL handler	Muodostaa yhteyden tietokantaan, muotoilee viestin ja kirjoittaa viestit tietokantaan	Palvelin
Alert Service Database Handler	Käy läpi hälytystietokannan merkinnät ja luo niistä tarvittaessa julkaisun.	Palvelin

4.2 Hälytyksen luonti ja lähetys

Lähetyksestä vastaa ”Alert Sender” -moduuli, joka suorittaa tarkkailun, hälytysviestin luomisen ja viestien lähettämisen. Palvelujen tilat selvitetään skripteillä, jotka tarkastavat taustaprosessien hallintaohjelmalta tarkkailtavan palvelun tilan palauttaen totuusarvon. Tarkastus on tietyn palvelun osalta epäonnistunut, jos totuusarvo on epätosi, eli ”False”.

Kaikkien epäonnistuneiden tarkastuksien tulokset kapsuloidaan nimettyyn monikkoon (engl. named tuple), joista muodostetaan lista. Listan monikoiden arvot sarjallistetaan ja lähetetään. Mikäli minkään

skriptin tuloksena ei luotu listaan nimettyä monikkoa, eli kyselyt onnistuvat, lähetysten sijaan lähetetään ”heartbeat”-viesti, joka osoittaa, että radio toimii oikein.

Sarjallistaminen tapahtuu ”monitors”-moduulia käyttäen. Viestiin sarjallistetaan siirrännän versio, palvelun id, palautuskoodi sekä viestin vapaamuotoinen osa, joka on rajoitettu 60 merkkiin. Sarjallistetun hälytysviestin pakkauksen muotoilumerkkiyhdistelmän on pieni, sillä kolme kokonaislukumuotoilumerkkiä ’B’ riittävät. Pakattuna versio, id ja hälytyskoodi ovat siis vain kolme tavua.

4.3 Hälytyksen vastaanotto

Vastaanottopuolella ”Alert Receiver” -moduuli ottaa viestin käsittelyynsä. Moduuli poistaa sarjallistuksen samaa ”monitors”-moduulia käyttäen, jota käytettiin pakkaamisessa. Viestin pohjalta tehdään SQL-kielinen upotuskomento. Lähettäjäradion tunnus ja luomisajankohta otetaan mukaan tässä vaiheessa ja merkitään upotuskomentoon. Samalla tarkistetaan, onko viestin tyyppi ”alert” vai ”heartbeat”. Tietokantaan otetaan yhteys ja hälytysviestin arvot tallennetaan tauluun suorittamalla SQL-komento.

Hälytysjärjestelmään liittyy myös toinen vastaanotettujen hälytysten käsittelytapa. Tässä toisessa käsittelyssä viesteistä muodostetaan saman tietokannan erilliseen tauluun hälytys, josta automaattisesti muodostetaan JIRA-julkaisu ylläpitotukihenkilöiden tiedoksi. Jos”heartbeat”-viestiä tai hälytysviestiä ei saada palvelimella tiettyyn aikaan mennessä, hälytys generoidaan ilmoittamaan radion toimimattomuudesta. Cron-tab ajastuksesta vuoksi tarkkailuskriptit lukevat palveluiden tilat ja muodostavat viestin, vaikka hälytystä ei olisi vielä kuitattu. Kuitenkaan uutta julkaisua ERP-järjestelmään ei luoda, jos edellistä ei ole kuitattu.

4.4 Koodin uudelleenkäytettävyys

Uuden hälytysjärjestelmän tapa seurata muita palveluita, ja hälytysten luomistapa tulevat poikkeamaan niin paljon alkuperäisestä järjestelmästä, että alkuperäisen päälle olisi hankala rakentaa uutta järjestelmää. Alkuperäisessä hälytysjärjestelmässä on kuitenkin hyvin yksinkertaisia ja tehokkaita ratkaisuja palvelun ja levytilan selvittämiseksi, joten niitä voidaan mukailla ja kehittää niitä eteenpäin eikä niitä tarvitse tehdä alusta asti.

Tarkkailuskriptien ydin voidaan pienin muutoksin ottaa uuteen hälytysjärjestelmään mukaan. Tietokantaan yhteyden luominen voidaan myös ottaa käyttöön pienin muutoksin ja viestin muotoilu SQL-lauseeksi toimii samalla periaatteella, eli ”PSQL handler” -moduulin metodeja voidaan uudelleen käyttää muutosten jälkeen.

4.5 Uudet kokonaisuudet

Uuteen hälytysjärjestelmään tulee tehdä tarkkailijamoduuli, joka voidaan alustaa eri parametreillä seuraamaan erilaisia asioita. Koska tämän uuden moduulin toiminta tulee olemaan hyvin erilainen riippuen siitä, mitä tarkkaillaan, täytyy luoda luokka, jota voi periyttää tarpeen mukaan.

”Alert sender” -moduuli muuttuu niin, että se toimii alkupisteenä, joka hallitsee tarkkailijaolioiden rinnakkaisajon. Tarkkailijaolioiden alustamiseen täytyy luoda oma moduuli, joka määrittää, miten tarkkailijat alustetaan. Kokoonpanoasetuksia varten luodaan oma moduulinsa, joka lukee asetukset ja jonka pohjalta rinnakkain ajettaville prosesseille voidaan antaa kokoonpanoasetukset. Kokoonpanoasetusten hallintamoduulin avulla kokoonpanoasetuksia voidaan muuttaa komentoriviltä.

Koska viesteihin tullaan pakkaamaan enemmän tietoa, sarjallistusta ja sen poistoa varten täytyy tehdä omat kokonaisuutensa. Samasta syystä tietokantaan tulee luoda uusi taulu näitä uudenlaisia viestejä varten.

5 TOTEUTUS

Kehitystarve keskittyi hälytysjärjestelmän uusien ominaisuuksien, järjestelmän skaalautumiskyvyn ja rinnakkaisajon toteuttamiseen. Uusiin ominaisuuksiin kuuluvat laitteiston arvojen luku, historian kirjoitus ja sen hallinta. Hälytysten luominen ja niiden hallinta sekä tietokantaan liittyvä hallinointi ovat ominaisuuksia, joita tuli jatkokehittää. Koska kehityshaasteisiin kuuluvat myös koodin helppo ylläpito, muokattavuus ja kokoonpanoasetusten helppo mukauttaminen, ohjelmiston arkkitehtuuri muuttui huomattavasti. Tässä luvussa tutkitaan edellä mainittujen ominaisuuksien toteutustapoja ja vertaillaan niiden erilaisia toteutusvaihtoehtoja.

5.1 Arkkitehtuuri

Rinnakkaisuutta varten Pythonille on olemassa kirjastoja asynkronisen rakenteen toteuttamiseksi, joko monisäieajolla (engl. multithreading) tai prosessien moniajolla (engl. multiprocessing). Säiepohjaiset moduulit käyttävät nimensä mukaan säikeitä, jotka ovat samassa muistiympäristössä, kun taas jokaisella prosessilla on omat muistiympäristönsä (Watson, 2005).

Prosessien moniajon suurimpia vahvuuksia ovat erillinen muistiympäristö, suoraviivainen koodi ja sen kyky käyttää hyväksi monta prosessorin ydintä (Watson, 2005). Suurimpina haittoina moniajolle on sille ominainen suuri muistin tarve. Moniajossa täytyy myös synnyttää prosessit, minkä vuoksi moniajo on hitaampi monisäieajoon verrattuna (Watson, 2005).

Monisäieajon vahvuuksiin kuuluu sen hyvin pieni järjestelmän kuormittavuus. Eri säikeiden välillä on helppo jakaa dataa, mutta oikea toiminta täytyy varmistaa kilpatilanteiden estämiseksi (engl. race condition), sillä monisäikeisessä toteutuksessa kaksi eri säiettä voivat ylikirjoittaa toisensa käsitellessään samaa tietoa (Watson, 2005). Tämän vaihtoehdon huonoihin puoliin kuuluu aiemmin mainitun seikan takia se, että ohjelmoijan tulee tahdistaa säikeet.

Prosessien moniajo on siis hitaampi, mutta sen hallitseminen ja toteuttaminen ovat suoraviivaisempaa. Kunhan mahdolliset kilpatilanteet ja umpikujatilat (engl. dead lock) otetaan huomioon, monisäieajo on nopeampi ja nopeammin käyttöönotettava. Rinnakkainajon toteutustavaksi valitaan siis monisäieajo, johon valitaan Pythonkirjasto ”Gevent”, joka on yrityksen käytössä ollut aiemminkin.

Gevent on vuorottaisrutiiniin pohjautuva Python-kirjasto, joka käyttää prosessin sisäisiä greenlet-säikeitä synkronisuuden saavuttamiseksi (Bilenko, 2019). Greenlet-säikeisiin luodut prosessit voidaan sitten koota säievarantoon, jonka sisällä prosessit vuorottelevat sitä mukaan, kun valmistuvat (Bilenko, 2019).

Jokaista seurattavaa asiaa varten siis muodostetaan uudet prosessit, jotka kootaan säievarantoon. Osa prosesseista lukee tietoa tiedostosta tai kirjoittaa sitä tiedostoon, joten loogisen eheyden säilyttämiseksi ja datan varastoinnin yhdenmukaisuuden vuoksi prosessien tulee käyttää semaforia. Semaforin tarkoitus on tahdistaa tiedosto-operaatiot, jotka käyttävät yhteisiä tiedostoja.

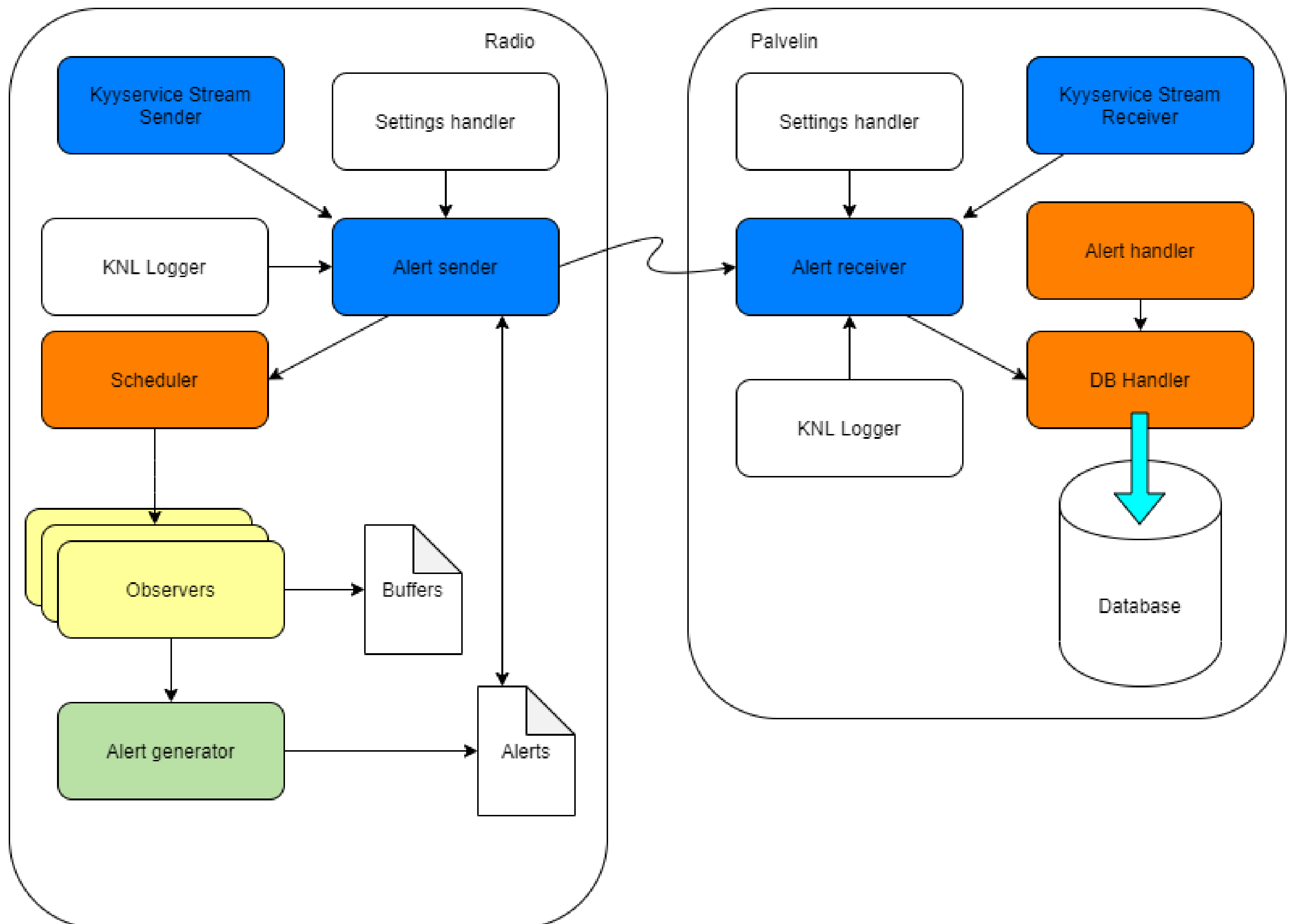
Tätä ongelmaa varten Gevent-kirjastossa on semaforin tavoin käyttäytyvä lukko, josta voidaan tehdä yhteinen kaikkien moduulien ja prosessien kesken. Lukkoa käyttävien metodien täytyy, joko saman tien tuottaa tulos tai suorittaa aikakatkaisu ehkäisemään tilanne, jossa lukkoa ei enää palauteta muiden metodien käytettäväksi.

5.1.1 Alert Service 2 esittely

Radiossa toimivaan lähetyspuoleen kuuluvat kuvassa 7 vasemmalla puolella olevat moduulit Alert Sender, Scheduler, Observer ja Stream Sender. Kuvan oikealla puolella olevat Alert Receiver, DB Handler, Alert Handler ja Stream Receiver kuuluvat palvelimen puolella toimivaan vastaanottopuoleen. Kummallekin kokonaisuudelle yhteisiä moduuleja ovat Alerts Logging, joka hoitaa lokin kirjoittamisen, ja Settings Handler, jolla käsitellään kokoonpanoasetuksia.

Kuva 7 ja taulukko 2 havainnollistavat tätä. ”Alert sender” alustetaan ”Settings handler” -moduulin syöttämien kokoonpanotiedostojen avulla, ja ”Stream Sender” -moduulin luokka alustetaan samalla. ”Alert sender” käynnistää ”Scheduler”-moduulin, joka luo ”Observer”-tarkkailijaoliot. Jokainen tarkkailja alustaa oman viestienkäsittelijän ”Alert Generator” -moduulin pohjalta. Tarkkailijat luovat ”Buffers”-rengaspuskureita, ja tarkkailijat voivat kutsua viestinkäsittelijää luomaan hälytyksiä. ”Alert sender” tarkistaa tietyn väliajan välein kansion hälytyksien varalta ja lähettää löytämänsä hälytysviestit vastaanottajalle ”Alert receiver”.

Vastaanottoja ”Alert receiver” alustetaan ”Settings handler” -moduulin syöttämien kokoonpanoasetusten mukaisesti, ja ”Stream Receiver” -moduulin luokka on se osa, joka ottaa itse asiassa ottaa viestin vastaan. ”DB Handler” muodostaa yhteyden tietokantaan ja tallentaa tietokantaan viestin käyttäen viestin purkamiseen ”Alert handler” -moduulin luokkaa.



Kuva 7: Alert Service 2

Taulukko 2: Uusien moduulien kuvaus

Moduuli	Kuvaus	Missä käytössä
Stream Sender	KNL:n lähetysmoduuli. Lähettää dataa HF yli.	Radio
Alert Sender	Stream Sendermoduulista periytetty hälytysten lähetysmoduuli.	Radio
Scheduler	Tarkkailijaolioiden alustamismoduuli.	Radio
Observer	Tarkkailijoiden pääluokka. Määrittää toiminnan ja käyttäytymisen.	Radio
Alert Generator	Hälytysviestin käsittelijä radiossa. Sarjallistaminen ja tallennus.	Radio
KNL Logger	Järjestelmän toiminnan lokimerkitsijä.	Radio ja Palvelin
Settings Handler	Järjestelmän kokoonpanoasetusten käsittelijä	Radio ja Palvelin
Stream Receiver	KNL:n vastaanottomoduuli. Vastaanottaa HF yli lähetetyn datan	Palvelin
Alert Receiver	Stream Receivermoduuli periytetty hälytysten vastaanottomoduuli.	Palvelin
DB Handler	Tietokannan käsittelyn toimintojen sisältävä luokka.	Palvelin
Alert Handler	Viestin sarjallistamisen purkamismoduuli.	Palvelin

5.1.2 Tietokanta

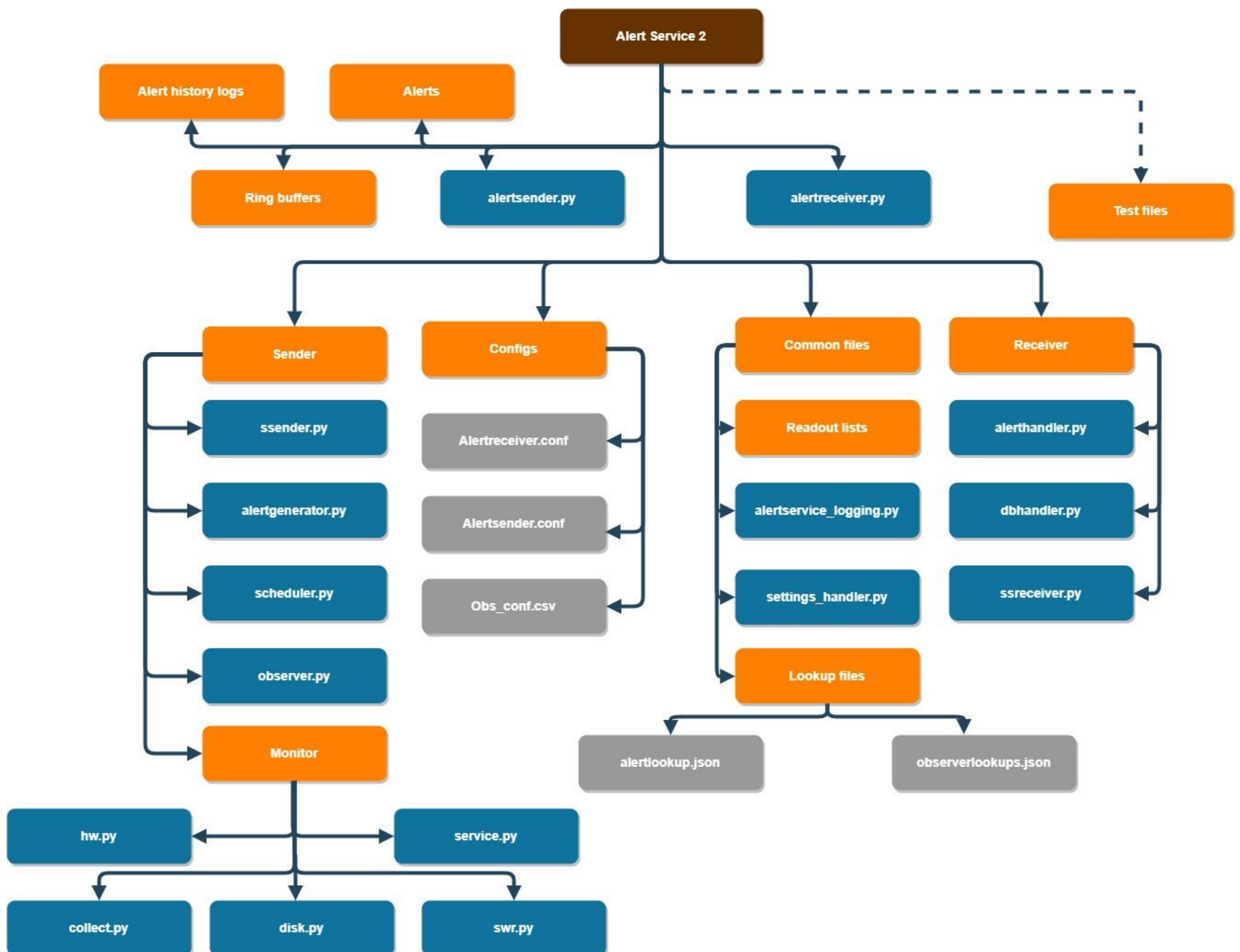
Kehitetyn hälytysjärjestelmän hälytykset poikkeavat ensimmäisen hälytysjärjestelmän hälytyksistä, joten uusia hälytyksiä varten täytyi myös luoda uusi taulu KNL Networksin tietokantaan. Taulukossa 3 on esitetty uuden taulun sarakkeet, sarakkeen arvon tyyppi ja muut mainittavat asiat.

Taulukko 3: Tietokannan esittely

Sarake	Datatyyppi	Muuta
id	int	primary key
generated	timestamp	not null
received	timestamp	not null
radio_id	int	not null
alert_value	int	
alert_text	text	
alert_reason	text	
parameter	text	

5.1.3 Tiedostohierarkia

Kuvassa 8 esitetään järjestelmän tiedostohierarkia. Kaavio on värikoodattu niin, että oranssi tarkoittaa kansiota, sininen Python-tiedostoa ja harmaa muuta tiedostoa, kuten kokoonpanotiedostoa tai tarkistustiedostoa.

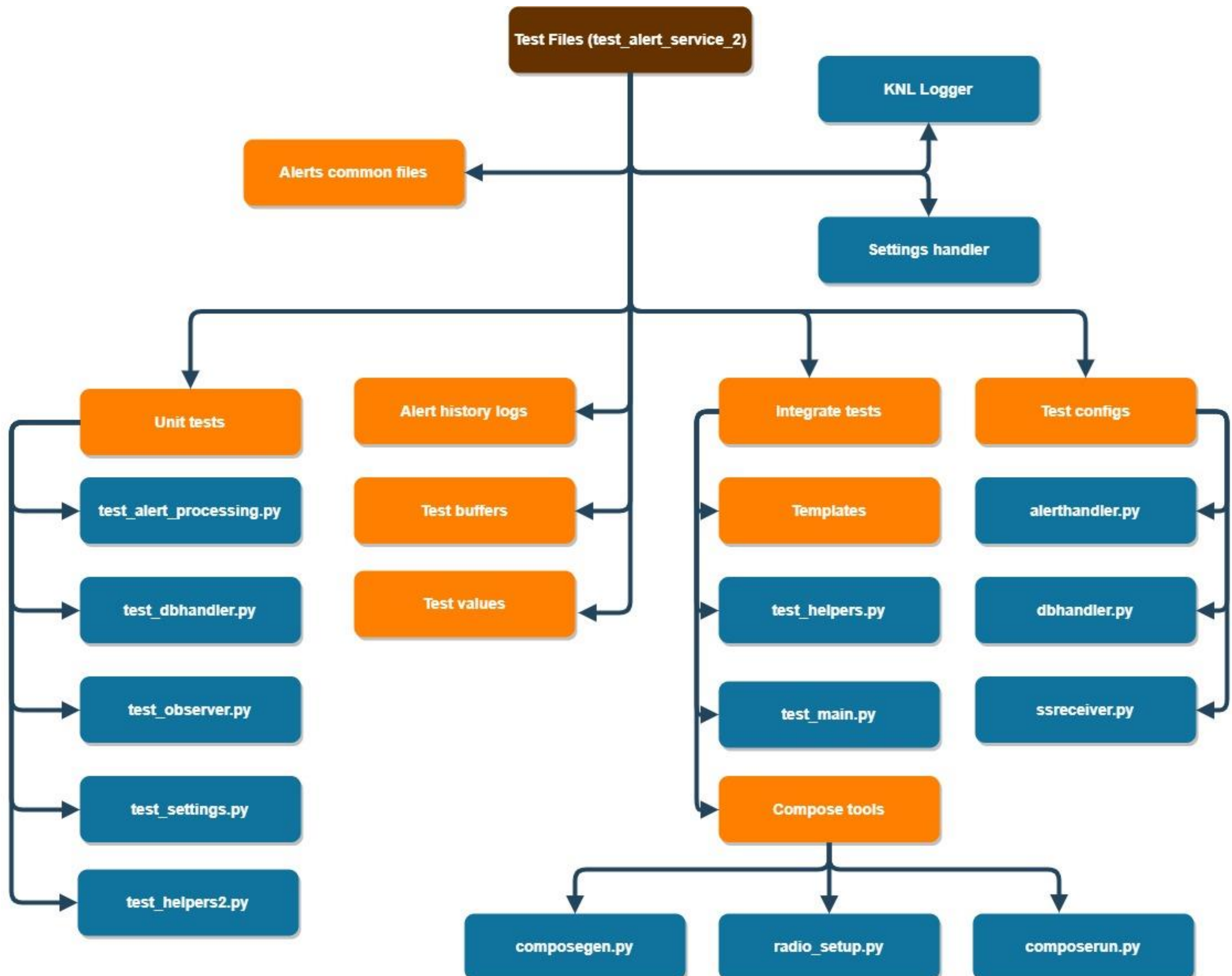


Kuva 8: Tiedostohierarkia

Hakemistot ”Alerts”, ”Ring buffers” ja ”Alert history logs” on tarkoitettu väliaikaisten tiedostojen säilyttämiseen, ja itse radiolla ne tulevat olemaan Linux-käyttöjärjestelmille ominaisessa ”/var/”-hakemistossa. Lähetys- ja vastaanottokomponenttien yhteiset tiedostot löytyvät ”Common files” hakemiston alta, jossa sijaitsevat alihakemistot tarkistustiedostoille ja listat, joissa on määritetty hälytyksien luontia varten muut oleelliset asetukset ja lukemat. Tiedosto ”alertLookup.json” sisältää hälytykset ja niiden koodit numeroina. Vastaavaan tapaan ”observerlookups.json” sisältää kaikkien

tarkkailijoiden nimet ja niiden koodit numeroina. Kokoonpanoasetukset sijaitsevat ”Configs”-hakemiston alla. Kokoonpanoasetustiedostot ja tarkistustiedostot sijaitsevat Linux-käyttöjärjestelmälle ominaisen ”/etc/”-hakemiston alla itse radiolla.

Kuvassa 9 on esitetty testitiedostojen hierarkia aiemmin esitetyllä värikoodauksella.



Kuva 9: Testien tiedostohierarkia

Testit ovat erillään muusta järjestelmästä ja testeille on luotu omat kokoonpanoasetukset. Testeissä kaikki tarvittavat arvot luetaan testien omasta ”Test values” -hakemistosta. Rengaspuskurit kirjoitetaan myös erilleen itse hälytysjärjestelmästä. Testien luomat hälytykset tallennetaan omaan hakemistoonsa, kuten myös hälytyksistä aiheutuvan lokien tallennuskin. Kuitenkin testit käyttävät hälytysjärjestelmän tarkistustiedostoja, jotka löytyvät ”Common files” alta, kuten kuvassa 9 esitetty.

Hakemiston ”Unit tests” alla on moduulien metodien toiminnan testaamiseen tarkoitettuja Python-tiedostoja. Moduuli ”test_helpers.py” sisältää metodeja, jotka auttavat testien ajossa. Esimerkiksi aina testin aluksi ajetaan metodi, joka puhdistaa väliaikaiset tiedostot. Näitä tiedostoja ovat esimerkiksi epäonnistuneiden testien takia jääneet tiedostot, jotka estäisivät alkavaa testiohjelmaa toimimasta oikein.

Integraatiotestit ja niihin liittyvät tiedostot sijaitsevat hakemiston ”Integrate tests” alla. Kaikki integraatiotiestit on kirjoitettu ”test_main.py” Python-tiedostoon. Kuten yksikkötesteissäkin, integraatiotestejä varten on myös tehty avustajametodit testien ajon tueksi. Näiden lisäksi hakemistossa on KNL Networksin kehittämät ”Template” ja ”Compose tools”, jotka sisältävät kaiken tarvittavan koodin radion ja hallinnointipalvelimen simulaation pystyttämiseen ja ajamiseen.

5.2 Ratkaisut

Vaatimusmäärittelyssä oli eritelty kaikki seurattavat asiat, joiden seuraaminen vaihtelee tapauskohtaisesti. Kuitenkin jokaisen asian häiriön ilmaantuessa tulee luoda yhdenmukainen hälytys. Seurattavat asiat voidaan jakaa viiteen pääluokkaan, kuten taulukossa 4 on esitetty.

Taulukko 4: Tarkkailijoiden pääluokat

Nimi	Kuvaus
Service	Radiolla pyörivät prosessit, kuten ”fileservice”, ”mailservice” ja ”connec-tionmanager”
Disk	Radion järjestelmän vapaana oleva tila
HW	Radion laitteiston, eli ”Hardware”-lukemat, kuten jännite ja virta. Lyhennettynä ”HW”
SWR	”Standing Wave Ratio”, eli arvo, joka osoittaa takaisinheijastuksen osuutta lähetetystä signaalista.
Collect	IoT-datankeräyspalvelu.

5.2.1 Palveluiden ja levytilan seuraaminen

Palveluiden ja radion järjestelmän vapaan tilan tarkastaminen tapahtuu samalla periaatteella kuin se tapahtui ensimmäisessä versiossa. Vapaana oleva levytila lasketaan suorittamalla ”statvfs”-komento, joka on osa Pythonin ”os”-moduulia. Komennolla saadaan tietoa käytössä olevasta tiedostojärjestelmästä (Brouwer, 2003). Vapaa tila jaetaan tiedostojärjestelmän koolla, minkä tuloksena jäljellä oleva tila voidaan ilmaista desimaalimuodossa. Tulosta verrataan sitten tarkkailijaolion määritettyyn arvoon, joka merkitsee pienimmän sallitun arvon.

Radiolla palveluita pyörittää taustaprosessienhallinnointiohjelma, jonka ”status”-komento yhdistettynä halutun palvelun nimeen palauttaa palvelun tilan. Yksinkertaisuudessaan hallinnointiohjelmalle tehdään kysely, jonka tulosteen perusteella luodaan mahdollinen hälytys. Jokainen palvelu määritetään erikseen tarkkailijan kokoonpanoasetuksiin, joka mahdollistaa vaihtoehdon jättää jokin palvelu huomioimatta, jos palvelu on tarkoituksella poistettu käytöstä radiolla.

5.2.2 HW-arvojen seuraaminen

Hälytysjärjestelmän HW-arvojen lukeminen perustuu radion kirjoittaman tiedoston lukemiseen. Radio lukee laitteiston arvot tiedostoon, joka ylikirjoitetaan aina, kun radio saa uuden lukeman. Havaittu arvo kirjoitetaan tiedostoon, jolla on sama nimi, kuin itse tarkkailtavalla arvolla. Radio ei kirjoita lokia tai historiaa luetuista arvoista radion tiedostojärjestelmään, minkä vuoksi hälytysjärjestelmä itse kirjoittaa lokia. Lokin koko voidaan määrittää tarkkailijoiden kokoonpanoasetuksissa. Hälytysjärjestelmä lukee parametrin arvon paikalliseen muuttujaan ja kirjoittaa sen rengaspuskurin tavoin toimivaan lokiin. Samalla lokiin kirjoitetaan myös kirjoitusajankohta.

Seuraavaksi lukemaa verrataan asetettuihin ylä- ja alarajoihin, joista kummastakin on varoitus- ja vaaratasot. Vertailut tapahtuvat järjestyksessä: ensin vaaratasot, sitten varoitustasot, ja hälytyksen laukaisee ensimmäisenä rajan ylittänyt arvo.

Lukeman äkillinen nousu selvitetään kirjoitetusta historialokista. Historia käydään läpi silmukalla ja sen sisäisellä silmukalla. Kaikkia niitä historiaan kirjoitettuja arvoja, jotka on kirjoitettu määritetyn aikavälin sisällä, verrataan toisiin samat ehdot täyttäviin arvoihin, ja jokaisen arvon kohdalla lasketaan arvojen erotuksen itseisarvo, joista suurin ero otetaan talteen. Lopuksi historiasta löydettyä suurinta eroa verrataan tarkkailijaolioon asetettuun suurimpaan sallittuun eroon ja tämän ylittyessä hälytys luodaan.

5.2.3 Takaisinheijastuksen seuraaminen

Toisin kuin laitteiston muun toiminnan seuranta, SWR:n lukemat kirjoitetaan lokiin, jossa on myös automaattisen tarkkailun kannalta turhaa dataa. Lokista luetaan kahdenkymmenen eri SWR-lohkon lukema, joista jokaiselle on asetettu eri rajat. Tähän lokiin kirjoitetaan SWR-lohkojen tietojen lisäksi jokaisella rivillä oleva aikaleima ja jokin tekstimuotoinen lokiviesti.

SWR-lohkot luovat kaksi lokikirjausta: aloitus- ja lopetusrivit. Tarkkailijan periytetyssä luokassa on määritelty SWR-arvojen tarkistusmetodi, joka käy koko lokin läpi ja vertaa lokirivin aikaleimaa lukuhetken aikaleimaan. Jos aikaleimojen erotus on tietyn rajan sisällä, etsitään lokirivstä aloitusriville ominainen merkkijono. Lokirivi, josta merkkijono löytyi, asetetaan sitten muuttujaan ja seuraavaksi tunnistetaan lokirivi, joka täyttää lopetusriville ominaisen merkkijonon, ja tämä löydetty rivi asetetaan toiseen muuttujaan. Näistä kahdesta rivistä sitten yhdistetään tärkeät tiedot.

Mikäli uusi SWR-lohkon aloittava lokirivi löytyy, juuri tämän SWR-lohkon tietojen hakeminen täytyy lopettaa, nollata kummatkin muuttujat sekä aloittaa seuraavan SWR-lohkon tutkiminen. Jos lopettava rivi löytyy, aloitusrivistä otetaan talteen lohkon numero ja taajuus, ja lopettavasta rivistä otetaan talteen SWR-lukema. Seuraavaksi listasta haetaan lohkon numeroa vastaava SWR-arvon yläraja ja sitä verrataan löydettyyn lukemaan. Sallitun rajan ylittyessä talletetut arvot tallennetaan listaan. Kun koko loki on käyty läpi, kaikista listatuista arvoista luodaan sitten mahdolliset hälytykset.

5.2.4 ”Collect”-palvelun seuraaminen

”Collect” on IoT-datan keräyspalvelu, joka käsittelee keräämänsä datan ja lähettää sen radion avulla KNL Networksin pilvipalveluun. Collect-järjestelmässä on monta rajapintaa, joiden toiminnan seuraamisen toteutus poikkeaa muista tarkkailtavista palveluista.

HTTP-pohjainen kyselymetodi selvittää Collect-palvelun isännän nimen, minkä avulla saadaan lista kaikista palvelussa tapahtuneista virheistä JSON-oliona. Seuraavaksi kaikista listan JSON-olioista muodostetaan merkkijono, jossa on virheen ID, tarkempi kuvaus, päivämäärä ja lähde. Hälytystä varten luodaan virhesanoma, jos isäntää ei löydy, isäntää ei voida selvittää tai JSON-oliota ei voida lukea. Kun virheet on kerätty listaan, listan pohjalta luodaan mahdolliset hälytykset yksitellen ja ne tallennetaan lähettämistä varten.

5.2.5 Rengaspuskuri

Paikallista tiedonhallintaa ja sen tallennusta varten hälytysjärjestelmään tuli kehittää ratkaisu. Tämän tallennusjärjestelmän tuli olla hyvin kevyt ja puhtaasti tiedostopohjainen. Sen tuli myös olla osa hälytysjärjestelmää eikä oma itsenäinen sovelluksensa.

Käytökseltään ratkaisun tuli toimia rengaspuskurin tavoin. Rengaspuskurit toimivat ’First In First Out’ -periaatteella, ja nimestään huolimatta ne ovat lineaarisia. (Wada, 2013). Esimerkkinä tästä on kymmenen olion lista, joka täyttyy normaalisti täyteen asti. Listan ensimmäinen olio poistetaan, jos täyteen listaan pyritään lisäämän uusi olio. Kaikkien jäljellä olevien olioiden osoitinindeksi muutetaan yhtä pienemmäksi, eli toisena ollut olio on ensimmäisenä, ja uusi olio lisätään listan loppuun kymmenenneksi olioksi.

Ratkaisun tulee myös täyttää ’ACID’-ominaisuudet, joita ovat ”Atomicity”, ”Consistency”, ”Isolation” ja ”Durability”. ”Atomicity”-termillä tarkoitetaan, että tapahtuva transaktio joko onnistuu täydellisesti tai jää täysin toteutumatta. (Møgelberg, 2011). ”Consistency”-termillä tarkoitetaan tallennusjärjestelmän yhtenäisyyttä: kun järjestelmään tapahtuu muutos, sen tulee muuttua yhdestä ristiriidattomasta ja yhtenäisestä tilasta toiseen. (Pearson Education, 2018). Tallennusjärjestelmän transaktioiden tulee olla eristettyjä, eli ne eivät saa olla näkyviä muille transaktioille ennen valmistumistaan, jotta järjestelmä täyttää ”Isolation”-termin vaatimuksen. ”Durability”-termi tarkoittaa, että jo tapahtuneiden transaktioiden vaikutuksen täytyy olla pysyvä eivätkä ne saa kadota myöhemmän häiriön takia. (Pearson Education, 2018)

Laitteistojen arvojen tallennuksessa tarvitaan vain kaksi saraketta, jotka ovat päivämäärä ja tarkkailtu arvo. Tämän vuoksi hyvänä ehdokkaana tämän ratkaisun pohjaksi ovat erilaiset aikasarjatietokannat, jotka ovat otollisia datan dokumentoinnin kannalta, jossa arvot muuttuvat suhteessa aikaan (Dunning & Friedman, 2015).

Olemassa olevista ratkaisuista sopivat vaihtoehdot toteutusta varten ovat Tiny DB ja ZODB. Yksi vaihtoehto oli myös kehittää oma ratkaisu. Tiny DB on nimensä mukaan minimalistinen tiedostopohjainen tietokantatoteutus, ja sitä on helppo laajentaa muihin tarkoituksiin (Siemens, 2016). Tiny DB on hyvin testattu, mutta sitä varten tulee kehittää ratkaisu, jotta se toimii rinnanajossa (Siemens 2016).

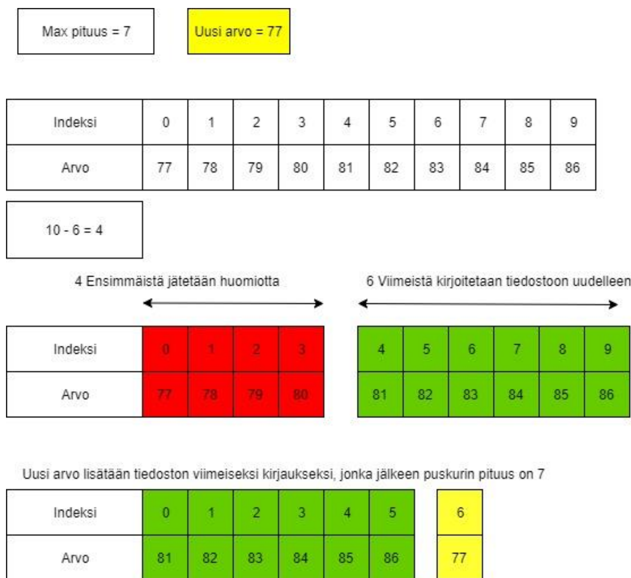
ZODB, eli Zope Object database, on Pythonille suunniteltu oliotietokanta. Se täyttää ”ACID”-ominaisuuden, ja se soveltuu dokumentaationsa mukaan hyvin paikallisten tiedostojen käsittelyyn ja muistissa olevan tietokannan käsittelyyn. ZODB kuitenkin vaatii omaa sovelluskoodia, jotta se toimii oikein rinnanajossa (Zope foundation, 2016).

Hyvänä puolena omakehitteisen ratkaisun puolesta on sen koodin riippumattomuus ulkoisista kirjastoista. Omasta ratkaisusta voi myös mahdollisesti tehdä keveämmän, sillä valmiissa ratkaisuissa tulee mukana ominaisuuksia, joita ei tarvita. Huonona puolena taas on suurempi työ testaamisen kanssa. Yhtäkään mainituista vaihtoehdoista ei kuitenkaan voi ottaa käyttöön ilman oman sovelluskoodin luomista. Mainitut seikat huomioon ottaen, järjestelmään kehitettiin oma ratkaisu paikallista tallennusta varten.

5.2.6 Rengaspuskurin toiminta

Rengaspuskurin toiminnan perusta ovat kaksi tapahtumaa, jotka ovat historian kirjoittaminen ja sen ylikirjoittaminen. Rengaspuskurin pituutta voidaan muuttaa tarkkailijan kokoonpanoasetuksia muuttamalla, joten ohjelman käynnistyksen välissä rengaspuskurin koko saattaa olla suurempi tai pienempi kuin aiemmin. Rengaspuskurin käynnistyksen hetkinen koko saattaa myös olla suurempi kuin sen kokoonpanoasetuksissa määritelty suurin sallittu koko. Prosessi alkaa tarkastamalla rengaspuskuritiedoston pituus laskemalla siinä esiintyvät rivit, ja jos tiedostoa ei ole, pituudeksi määritetään nolla. Uusi rivi kirjoitetaan tiedostoon, jos rengaspuskurin pituus on pienempi kuin suurin sallittu pituus. Jos rengaspuskurin pituus on sama tai suurempi kuin suurin sallittu pituus, kaikki rivit luetaan muistiin. Sitten luetut rivit kirjoitetaan lukuun ottamatta niin monta riviä alusta, että jäljelle jää enää yhtä vaille sallittu määrä, minkä jälkeen uusi kirjaus kirjataan.

Koska uusin kirjaus aina lisätään tiedoston loppuun, vanhin kirjaus on ensimmäisenä. Kun tiedosto luetaan Python-listaksi, voidaan kirjauksia myös esittää niiden indeksin numerolla päivämäärän sijaan. Kuva 10 esittää ylikirjoitusprosessia, kun tiedostossa on sallittua enemmän rivejä.



Kuva 10: Rengaspuskuri

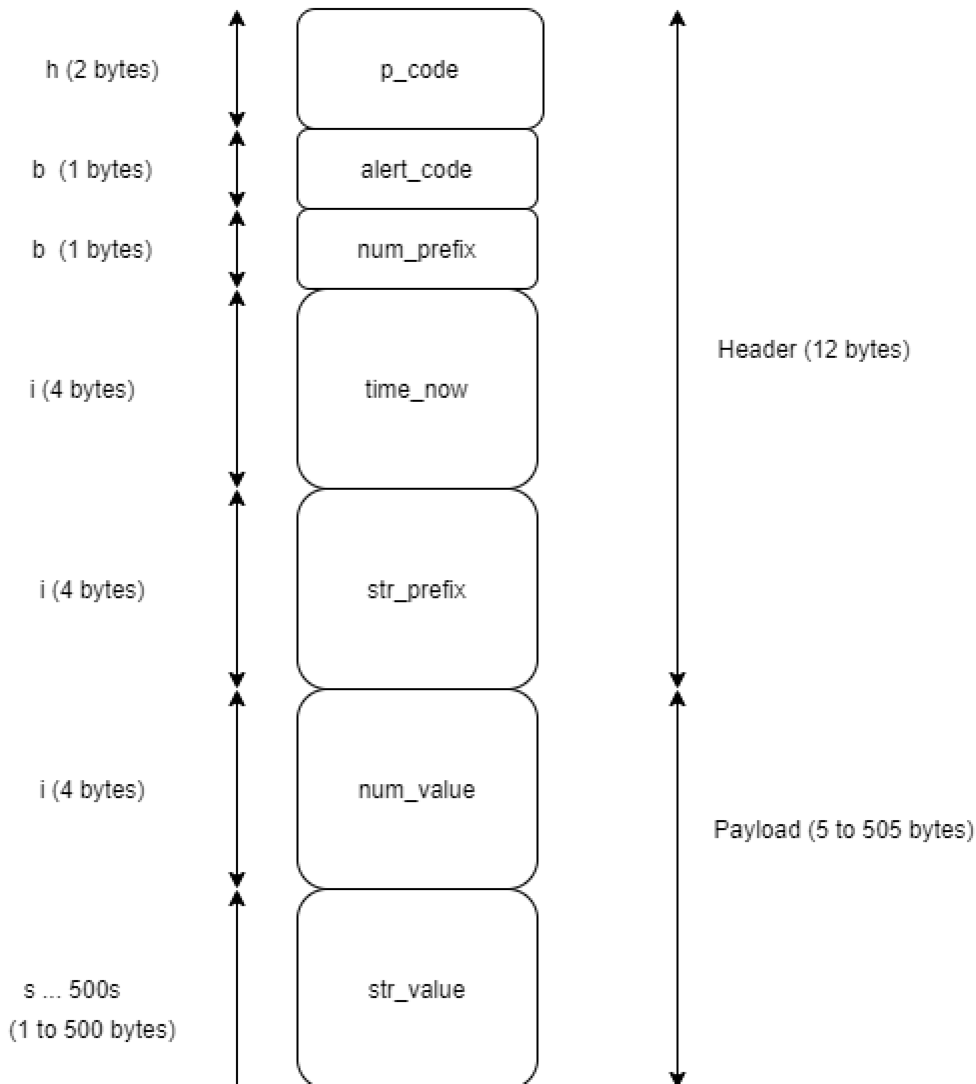
5.2.7 Sarjallistaminen

Sarjallistamisella tarkoitetaan datan muuntamista jostain tietystä tietorakenteesta tavujonoksi datan varastoimista tai lähetystä varten (Avro Big Data, 2019). Hälytysjärjestelmän käyttämä lähetysmoduuli lähettää kaiken datan tavujonona. Hälytyksen tietojen sarjallistamista varten täytyi kehittää ratkaisu, ja vastaavasti sarjallistaminen tulee purkaa vastaanottopäässä.

Sopivimmat Pythonille kehitetyt ratkaisuvaihtoehdot tätä ongelmaa varten ovat ”marshal”, ”Avro”, ”Struct” ja ”Pickle”. ”Avro” on tehokas ja kattava Apachen kehittämä sarjallistamistyökalu (Avro Big Data, 2019), mutta se ei kuulu Python:in peruskirjastoon. ”Pickle” ja ”marshal” ovat osa Pythonin peruskirjastoa, joista ”Pickle”-moduuli on hieman kehittyneempi (Python Software Foundation, 2019b). Picklen dokumentoinnin mukaan Pickle:ä tulisi aina käyttää marshalin sijaan, sillä marshal on suunniteltu Pythonin kääntäjän koodia varten (Python Software Foundation, 2019). Moduuli ”struct” on suoraviivainen tapa sarjallistaa ja käsitellä binääridataa ja se käyttää ”Format string” -muotoilufunktiota muuntaessaan Python-tietorakenteita (Python Software Foundation, 2019).

”Pickle”-moduuli tarjoaa kattavamman arsenaalin tiedon ja tietorakenteiden sarjallistamiseen ”struct”-moduuliin verrattuna ja on laajemmin käytetty. Kuitenkin struct-moduulin yksinkertaisuuden ja kompaktin pakkauksen vuoksi tässä työssä käytetään struct-moduulia.

Hälytysviesti muodostuu monesta sarjallistetusta arvosta. Viestiin on sarjallistettu pakkausetuliite, parametrin nimeä merkitsevä koodi ja hälytyksen selitekoodi. Muita arvoja ovat numeroarvon etuliite, unix-muotoinen päivämäärä, merkkijonoarvon etuliite, mahdollinen numeraalinen arvo ja mahdollinen merkkijonon arvo. Kuvassa 11 on havainnollistettu viestin rakenne.



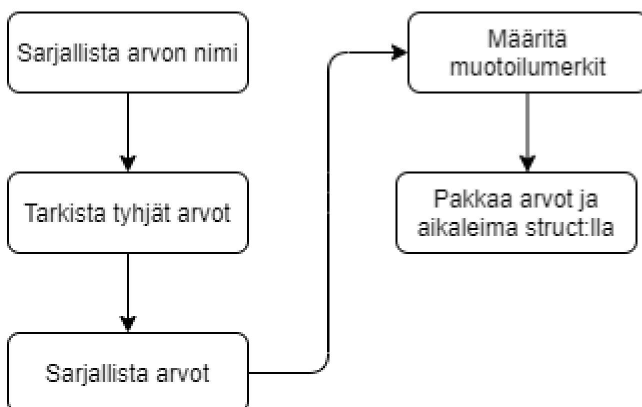
Kuva 11: Viestin rakenne

Kuten kuvasta nähdään, pakkauksen etuliite on ”hbbii”, jossa ”h” on short-tyyppinen kokonaisluku ja kooltaan kaksi tavua. ”b” on yhden tavun kokoinen signed char-kokonaisluku ja i on integer-kokonaisluku, jonka koko on neljä tavua. Yhdessä nämä viisi muotoilumerkkiä muodostavat 12 tavun kokoisen etuliitteen. Muotoilumerkkien järjestyksellä on väliä, sillä esimerkiksi yhdistelmä ”ibibh” on kooltaan 16 tavua johtuen tavasta, jolla rakenteen täyttö (engl. padding) toimii (Python Software Foundation, 2019).

Seuraavaksi arvon tyyppi tarkastetaan ja sen perusteella palautetaan numeroarvo, joka sijoitetaan ”alert_code” -kenttään, kuten kuvassa on esitetty. Tämän jälkeen muodostetaan numero- ja merkkijonoarvoille käytettävät etuliitteet, joita tarvitaan tässä komponentissa viestin sarjallistamiseen, sillä ohjelma tukee merkkijonon dynaamista kokoa huomioimalla sarjallistettavan viestin str_value-kentän pituuden.

Merkkijonon pituus lasketaan ja sen perään liitetään muotoilumerkki ”s” merkitsemään, että kyseessä on merkkejä. Esimerkiksi merkkijonolle ”koira” tulee etuliitteeksi 5s. Mikäli merkkijonon pituus on nolla tai sen arvo on null, merkkijonoksi asetetaan ”N/A”. Jos taas numeroarvoa ei ole, sen arvoksi asetetaan suurin mahdollinen muotoilumerkkiin ”i” mahtuva luku, eli 2147483647 (Python Software Foundation, 2019).

Sarjallistamisessa tarkkailtavan arvon nimi muutetaan sitä vastaavaksi numeroksi katsomalla numerokoodi Pythonin dictionary-oliosta, joka on luettu tarkkailijoiden tarkistustiedostosta. Myös hälytyksen syyn selitys katsotaan vastaavalla tavalla dictionary-oliosta, joka puolestaan on muodostettu hälytysten tarkistustiedostosta. Kuva 12 havainnollistaa hälytysviestin sarjallistamisen vaiheet.

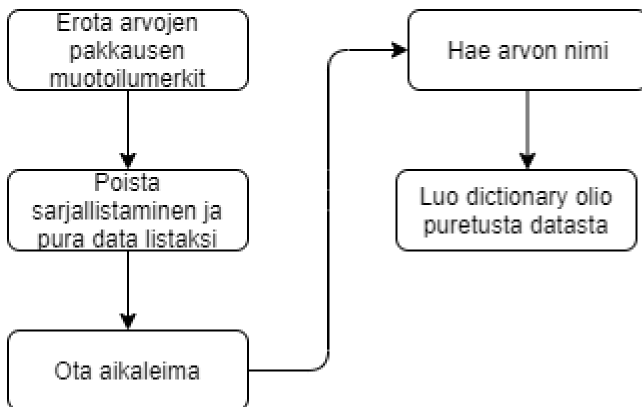


Kuva 12: Sarjallistamisen vaiheet

Sarjallistamisen poistamismetodi hakee viestistä sen pakkaustavan. Kuten kuvasta 12 voidaan nähdä, kaikki sarjallistettu data alkaa muotoilumerkkiyhdistelmällä ”hbbii”, joka on 12 tavun pituinen. Kolmanteen kenttään muotoilumerkkiä ”b” vastaamaan on tallennettu hälytysviestin numeerinen arvo. Jos merkkiin on tallennettu arvo on 0, käytetään integer-kokonaislukua ”i” .

Sitten luetaan hälytysmerkkijonon pituus, joka on tallennettu viidenneksi muotoilumerkkiä ”i” käyttämällä. Kun etuliite on nolla tai enemmän, se käsitetään ”s” -muotoilumerkin etuliitteeksi, joka tarkoittaa merkkijonon pituutta. Esimerkiksi merkkijonolle ”hevonen” tulisi muotoilumerkkiyhdistelmäksi purkamista varten ”7s”.

Kun purkamista varten on saatu pakkausmerkit, hälytysviesti puretaan struct-kirjaston ”unpack”-metodia käyttämällä ja viestin arvot kootaan listaan. Listasta otetaan unix-muodossa oleva päivämäärä ja muutetaan se muotoon YYYY/MM/DD HH/MM/SS. Parametrin nimen koodi luetaan tarkkailijoiden tarkistustiedostosta, joka on Pythonin dictionary-olion muodossa. Numeraalinen arvo tarkistetaan, ja jos arvo on 2147483647, se muutetaan vastaamaan tyhjää. Merkkijono tarkistetaan ”N/A” varalta, joka tarkoittaa tyhjää arvoa. Lopuksi kaikista puretuista ja käsitellyistä arvoista muodostetaan Python dictionary, jonka sarjallistamisenpoistometodi palauttaa. Kuvassa 13 vielä esitetty prosessin kulku.



Kuva 13: Sarjallistamisen poistaminen

5.3 Turvallisuuden huomiointi

Tuotantoon vietävän tuotteen turvariskit täytyi selvittää ja ratkaista niiltä osin kuin se oli järkevää, eli niiltä osilta, jotka on kehitetty ohjelmistoon, ja ohjelman alaisilta osilta. Kolmannen osapuolen koodin turvallisuus selvitetään ja ratkaistaan, jos se altistaa ohjelmiston osan haavoittuvaksi. Kyynel Oy noudattaa ISO/IEC 27001-sertifikaattia, joten turvallisuuden huomiointi oli tärkeä osa tätä työtä.

Tässä luvussa selvitetään, mitä turvallisuusriskejä työssä käytettyihin teknologioihin liittyy, ja tarkastellaan, miten ne voidaan ratkaista. Turvallinen koodi ei pelkästään tuota ennalta määritettyä asiaa ilman vahinkoa, vaan turvallisuutta varten myös sisäisen käyttäytymisen turvallisuus tulee huomioida (Bhowmick, 2014).

Kuten aiemmin esitetty, ohjelmointikielenä tässä työssä käytetään Python 2.7 ja tietokantaa käsitellään Postgresql SQL kielellä. Tämä tarkoittaa, että ohjelma voi olla altis SQL-injektioille etenkin sarjallistamisen poiston yhteydessä, kun viestin arvot liitetään SQL-lausekkeeseen. SQL-injektiolla tarkoitetaan vahingollisen SQL-komentojen ujuttamista koodin sekaan esimerkiksi lisäämällä vahingollinen koodin pätkä luotavaan SQL-komentoon. Nimenomaan SQL-komentojen muodostaminen ja sarjallistamisprosessien yhteydessä on yksi suurimmista tietoturvariskeistä (Checkmarx Ltd, 2018). Tässä työssä käytetään ”psycopg2”-adapteria Python-ohjelmointikielen ja Postgresql-tietokannan välillä, ja tämä adapteri on suojattu SQL-injektioita vastaan, kunhan sitä käytetään oikein (Di Gregorio, 2019).

Ohjelmassa luetaan komponenttien alustamisen yhteydessä kokoonpanotiedostoja, jotka voisivat sallia väylän vahingollisen koodin suorittamiseen. Tarkkailijoiden alustajamoduuli ”scheduler” on kehitetty käyttämään ”import”-metodia dynaamisesti niin, että se tuo oikean aliluokitetun moduulin, jonka pohjalta tarkkailijaolio luodaan. Kuitenkin tämä ja kaikki muu kokoonpanotiedostoista luetut arvot tarkistetaan ja todetaan niiden sisältävän odotetun arvon tai tyypin ennen kuin alustajamoduuli alustaa niiden pohjalta tarkkailijaoliot.

Suurta tuhoa aiheuttavia bugeja, jotka tarvitsevat korkeimmat käyttöoikeudet, ei luokitella korkealle riskitasolle (Python Software Foundation, 2019). Järjestelmää ei ole luotu estämään käyttäjää, jolla on korkeimmat käyttöoikeudet muuttamasta lähdekoodia tai kokoonpanoasetuksia vahingollisesti, sillä nämä riskit ovat vain pienen tärkeyden riskejä.

Ohjelma on rakennettu käyttämään Docker-säiliöitä Linux-pohjaiselle käyttöjärjestelmälle, joissa kummassakin on tietoturvariskejä riippuen käyttäjänoikeuksista (Willis, 2015). Tämän riskin torjumista ei kuitenkaan ole järkevää yrittää ratkaista tässä työssä. Vahingollinen käyttäjä tai ohjelma voi tehdä tässä vaiheessa suurempaa tuhoa kuin hälytysjärjestelmän lamauttamisen, eikä Docker-säiliöt altista hälytysjärjestelmää riskeille sen enempää kuin muutkaan komponentit.

6 TOIMINTA

Tässä luvussa kuvataan hälytysjärjestelmän toimintaa tarkemmin. Kuten nähdään luvussa viisi esitetystä kuvasta 7, ohjelma jakautuu lähetys- ja vastaanottopuoleen, joille yhteisinä moduuleina ovat kokoonpanoasetusten lukija ”Settings Handler” ja järjestelmän lokin kirjoittaja ”Alerts Logging”.

Radion lähetyskomponenttiin kuuluvat alustajamoduuli ”Scheduler”, tarkkailijamoduuli ”Observer” ja hälytyksen luomismoduuli ”Alert generator”. Tarkkailijalla tarkoitetaan mitä tahansa radion toimintaa tarkkailevaa oliota, joka on luotu ”Observer”-luokan pohjalta.

Palvelimen päässä vastaanotosta vastaa vastaanottajamoduuli ”Alert receiver”. Viestin käsittelystä vastaa hälytysviestien purkamiseen kehitetty ”Alert handler”, joka yhdessä ”db handler” -moduulin kanssa tallentaa viestien tiedot tietokantaan.

Lähetyksessä ja vastaanotossa ohjelma käyttää KNL Networksin Pythonille kehittämää lähetys-kirjastoa, jossa on määritetty lähetystä ja vastaanottoa varten metodit. Vastaanoton puolella käytetään ”Stream Receiver” ja ”Receiver” -moduuleita vastaanottamiseen, ja lähetyspäässä ” Stream Sender” -moduulia käytetään lähettämiseen.

6.1 Lähetyskomponentti

Tarkkailijat alustetaan lukemalla CSV-tiedosto, jossa niiden asetukset on määritelty. Tämän tiedoston arvot on pilkulla eroteltu ja yksi rivi vastaa aina yhden tarkkailijan asetuksia. Alla olevassa taulukossa on esitetty tarkkailijan parametrit ja lyhyt kuvaus niiden tarkoituksesta.

Taulukko 5: Tarkkailijoiden asetukset

Asetus	Kuvaus
parameter	Merkitsee tarkasteltavan arvon nimen.
type	Määrittää tarkkailijalle tyypin: disk, service, hw, swr tai collect
start_delay	Käynnistyksen jälkeinen viive ennen tarkkailurutiinien alkamista
sample_size	Rengaspuskurin koko riveinä
low_limit	Alarajan pienin sallittu lukema
high_limit	Ylärajan suurin sallittu lukema
warning_low	Alarajan varoituksen raja
warning_high	Ylärajan varoituksen raja
sample_interval	Tarkkailurutiinin aikaväli sekunteina
history_range	Merkitsee eron laskemiseen huomioitujen rivien määrän
difference_limit	Erotuksen rajan lukema
cooldown	Viive ennen samasta arvosta tapahtuvan hälytyksen luomista
readout_set	Hälytyksessä huomioitavan listan numero
active	Osoittaa aktiivisuuden. Arvolla 1 tarkkailija alustetaan.

Tarkkailijoita varten alustustiedoston lisäksi on myös eri ”readoutlist.txt” -tiedostoja, joiden nimet päätyvät numeroon ennen tiedostopäätettä. Taulukossa 5 esitetty ”readout_set” parametrin numero määrittää, mitä readout-listaa tarkkailija käsittelee hälytyksen yhteydessä. Nämä tiedostot sisältävät listan niiden rengaspuskureiden nimistä, jotka liittyvät hälytykseen ja hälytyksen tapahtuessa listatut rengaspuskurit otetaan talteen. Rengaspuskureiden arvojen kehitystä voidaan jälkeenpäin tutkia, jotta saadaan enemmän tietoa aiheutuneesta hälytyksestä.

Käynnistyessään ”Alert Sender” -moduuli lukee tarkkailijoiden kokoonpanoasetuslistan ja yleiset asetukset. Yleisiin asetuksiin kuuluu ”cooldown”-tiedoston polun määritys, rengaspuskureiden

tallentamiskansion määritys, tarkkailijoiden asetusten polun määritys, radion ajureiden kansion määritys, maksimi koko historian tallennusta varten ja se, millä aikavälillä seuranta tapahtuu.

Asetuksissa ”obs_conf.csv” on määritelty kaikki tietyn parametrin seuraamiseen tarvittavat arvot, joista ensimmäinen on ”parameter_name”, eli tarkkailtavan kohteen nimi. Otoksen koko ”sample_size” määrittelee, kuinka monta otosta rengaspuskuriin tulee. Esimerkiksi lämpötiloille ja jännitteille otetaan eri määrä otoksia tulosten analysoimiseen. Sarakkeet ”high_limit” ja ”low_limit”, eli parametrin ala- ja yläraja, määrittävät ne rajat, joiden sisällä tarkkailun kohteen tulisi olla, ja rajan rikkomisen yhteydessä hälytys laukeaa. Palveluiden tilaa tarkkaileville tarkkailijoille voidaan määrittää tämä arvo, mutta mikään palvelutarkkailijan metodi ei käytä sitä tarkkailleissaan palvelujen tilaa.

Otoksen aikaväli on ”sample_interval”, ja sillä määritellään huomioon otettava aikaväli tarkkailun kohteen arvojen eroa tutkittaessa. Esimerkiksi lämpötiloja tarkasteltaessa otetaan vain viimeinen tunti huomioon, eli riippuen asetetusta aikavälistä jokaista rengaspuskurin kirjausta ei oteta huomioon eroa tarkastellessa.

Sarake ”difference_limit” määrittää suurimman sallitun eron tarkkailun kohteen arvolle, ja sitä käytetään verrattaessa uusia kirjauksia vanhoihin. Asetus ”cooldown” merkitsee aikavälin, jolloin hälytys ei saa laueta. Tätä käytetään estämään uuden hälytyksen laukeamista samasta aiheesta. Kuten aiemmin on sivuttu, hälytyksen laukeamisen yhteydessä luetaan lista parametreistä, jotka ovat tärkeitä hälytyksen analysoinnin kannalta, ja ”readout_set” -asetuksen arvo ilmoittaa listan, joka huomioidaan. ”active” -asetus kertoo, onko kyseessä olevan parametrin tarkkailu aktiivisena. Kun arvo on 1, tarkkailu tapahtuu normaalisti, ja riviä, jossa esiintyy arvo 0, ei oteta huomioon tarkkailijoita alustettaessa.

6.1.1 Alert Sender -moduuli

Moduuli ”alert sender” toimii rajapintana käyttäjän ja ohjelman väillä. Tälle moduulille voidaan antaa komentorivikomentoja, joista tärkeimmät ovat ohjelman käynnistäminen ja yllä mainittujen kokoonpanoasetusten muokkaus. Näitä kokoonpanoasetuksia voi myös muokata käsin. Komennoilla muokkaaminen on tarkoitettu pienien muutosten tekemiseen, kuten yksittäisten parametrien muokkaamiseen.

”Alert sender” on rakennettu käyttäen ”argparse”- ja ”configparser” -moduuleita, jotka ovat kuuluvat Python 2.7:n peruskirjastoon (Python Software Foundation, 2019). Argparse-moduulia käytetään lukemaan ja käsittelemään komentorivin komennot ja configparser-moduulia käytetään kokoonpanoasetusten jäsentelyyn. Jos ohjelma pelkästään käynnistetään, se lukee kokoonpanoasetukset ”configs”-hakemistosta ja niiden perusteella luodaan ”Scheduler”-moduuli. ”Alert Sender” -moduulin alustuksen yhteydessä luodaan myös yksi semafori, jota käytetään lukkona jokaisessa muussa moduulissa loogisen eheyden ylläpidossa.

6.1.2 Scheduler-moduuli

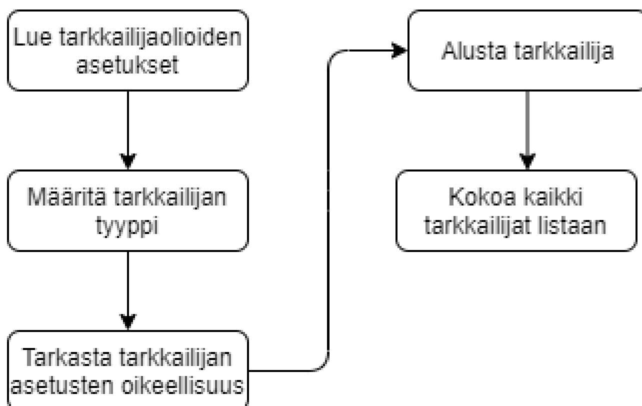
Alustajamoduulin ”scheduler” tarkoituksena on alustaa ”Observer”-oliot lukemalla kokoonpanoasetukset ja varmistaa, että tarkkailijaoliot on alustettu odotetuilla arvoilla. Scheduler-moduulin alustuksen yhteydessä yhtenä parametrinä odotetaan ”gevent”-kirjaston lukkoa, joka on luotu aiemmin mainitussa ”alert sender” -moduulissa. Käynnistyessään ”scheduler” lukee myös aiemmin esitellyt kokoonpanoasetukset ja tarkistaa, että ”cool down” -tiedostoa ei ole estämässä hälytyksen syntymistä.

Alustusmetodi lukee kokoonpanoasetukset rivi kerrallaan käyttäen Pythonin ”csv”-moduulia ja luo listan luetuista riveistä. Lista käydään läpi generaattorimetodilla, ja jokaisesta pätevästä rivistä luodaan tarkkailijaolio eli ”observer”-olio, ja nämä oliot kerätään listaan.

Tarkkailijaolioita luotaessa ensimmäisenä luetaan ”type”-parametri, joka määrää, mitä ”observer”-luokan aliluokitettua moduulia käytetään. Nämä moduulit ovat nähtävissä aiemmin esitetystä taulukosta 4. Luokkien tuonti tapahtuu lukemalla asetus, jonka jälkeen polku tähän moduulin selvitetään. Sitten Pythonin ”import”-metodia käyttämällä moduuli tuodaan ja tuotu luokka palautetaan oliona, jota käytetään kyseessä olevan tarkkailijaolion alustuksessa.

”Scheduler” lisää listaan kerätyt ”observer”-oliot gevent-säievarantoon ja käynnistää nämä säikeet. Ensin odotetaan asetuksissa määritetty aloitusviive, jonka jälkeen aloitetaan silmukka, jossa kaikki tarkkailijaolioiden toiminnallisuus tapahtuu. Jos hälytyksen laukeamisen yhteydessä asetettu hälytysviive on voimassa, silmukka tarkistaa viiveen eikä käy läpi tarkkailurutiinia. Muussa tapauksessa silmukka ajaa tarkkailijaolioiden tarkkailurutiinit vuorotellen ja käyttäen gevent-kirjaston ”sleep”-

metodia, jolla vuoro annetaan toiselle greenlet-säikeelle (Bilenko, 2019). Kuva 14 havainnollistaa ”Scheduler”-moduulin toimintaa.



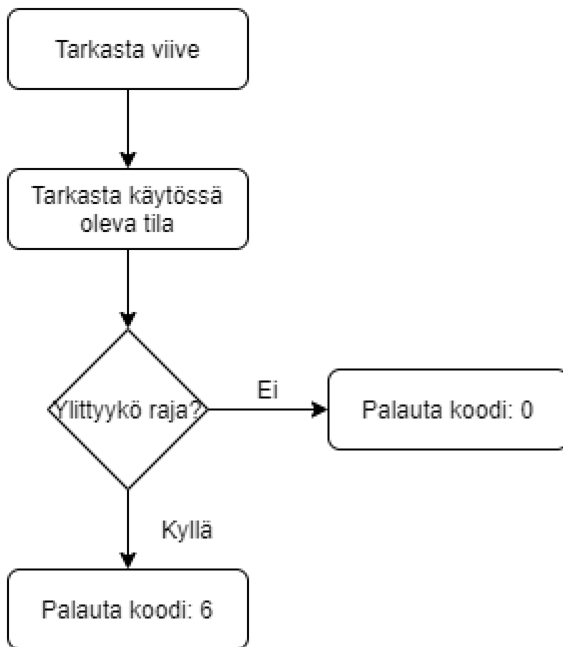
Kuva 14: Scheduler luokan toiminta

6.1.3 Observer-moduuli

Ohjelman ydin ovat tarkkailijaolioiden, eli ”observer”-luokkien, metodit. Aina yhtä tarkkailtavaa kohdetta varten ”scheduler” on luonut yhden tarkkailijaolion, kuten aiemmassa luvussa kerrottiin. Hälytyksen laukeaminen tapahtuu näiden olioiden sisällä, kun tietyt ehdot täyttyvät. Oliot myös hallitsevat rengaspuskurin historiaa kirjoittamalla uusimman havaitun arvon historiaan, joka on nimetty tarkkailun kohteen mukaan.

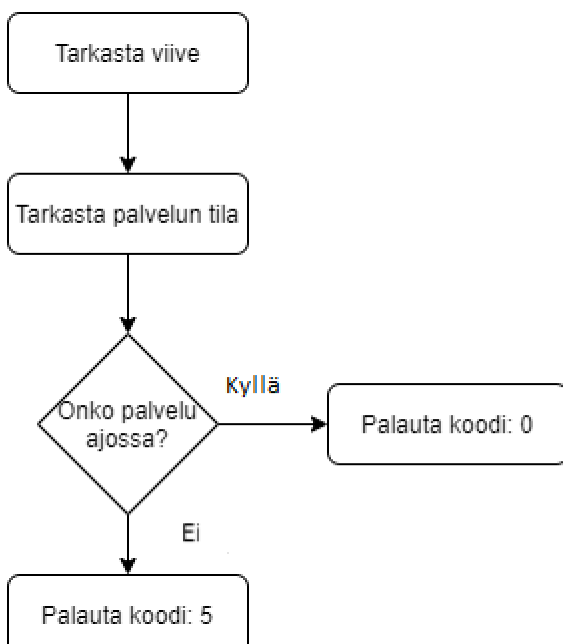
”Observer”-luokassa on määritetty kaikki käyttäytyminen ja toiminta lukuun ottamatta itse tarkkailua, joka on määritetty aliluokitetuissa moduuleissa. Eri tarkkailun kohteet voitiin luokittaa viiteen eri luokkaan, joiden mukaan aliluokitettiin observer-luokka ja niille luotiin omat rutiinimetodit. Nämä luokat ovat luvussa viisi esitetyt hw, service, swr, collect ja disk space. Tarkoituksena tälle periyttämiselle oli selkeästi erottaa käyttäytymisellään poikkeavat tarkkailijaoliot toisistaan ja selkeyttää tulevaisuudessa kehitettävien tarkkailijaolioiden luomista.

Jos kyseessä on ”Disk_Observer”-olio, olion ”inspect”-metodi laskee, kuinka monta prosenttia koko levytilasta on käytetty. Jos vapaa tila on alle tietyn prosenttimäärän, metodi palauttaa kokonaisluvun 6, joka kertoo hälytyksen käsittelijämetodille kyseessä olevan vähäinen levytila. Kuva 15 esittää tarkistuksen vaiheet.



Kuva 15: Levytilan tarkkailu

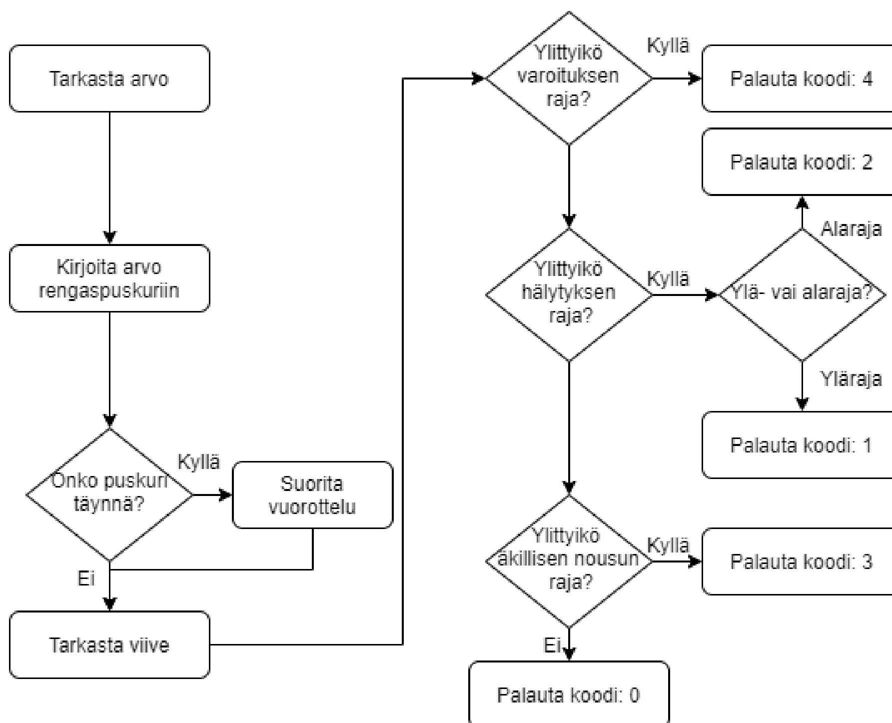
Jos tarkkailijaolio on alustettu ”service”-luokan pohjalta, kutsutaan metodi tarkistamaan, onko tarkkailijan nimen mukainen palvelu ajossa taustaohjelmien hallintaprosessissa. Metodi palauttaa kokonaisluvun, joka merkitsee palvelun toimettomuutta, jos hallintaprosessin mukaan parametrin nimellä olevaa prosessia ei löydy ajotilassa.



Kuva 16: Palvelun tarkkailu

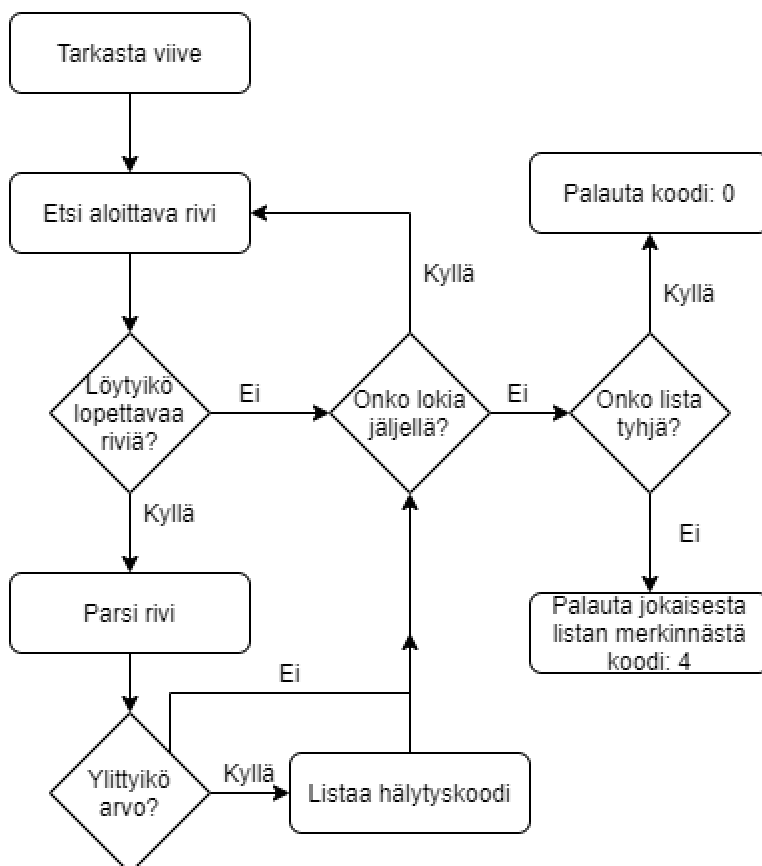
Radion laitteistojen toimintaa tarkkailevien olioiden eli ”hw”-luokan rutiini tarkastaa arvon ja vertaa sitä rajoihin ja rengaspuskurin kirjauksiin. Esimerkiksi radion lämpöä seuraava parametri luetaan lukumetodilla ja luettu arvo asetetaan olion sisäiseen muuttujaan. Seuraavaksi kutsutaan tarkistusmetodi, joka ensin tarkistaa, onko vaaran tai varoituksen ylä- tai alaraja ylitetty. Tämän jälkeen parametriä vastaava rengaspuskuritiedosto aukaistaan ja viimeisimpiä arvoja verrataan keskenään. Viimeisimpien otosten määrä on määritetty kokoonpanoasetuksiin. Mikäli kokoonpanoasetuksissa määritelty suurin sallittu ero ylittyy, hälytyksen koodiksi asetetaan hälytyksen aiheuttajaa merkitsevä koodi. Muut raja-arvot tarkistetaan samalla periaatteella.

HW-observer myös käsittelee luomiaan rengaspuskureitaan. Rengaspuskuritiedosto luodaan tarkkailtavan arvon nimen mukaan. Tiedostoon kirjoitetaan tapahtuma-ajankohta ja sen hetkinen luettu arvo. Rengaspuskuria varten on asetettu parametri, joka määrittää puskurin koon. Jos puskuri ei ole täynnä, eli puskurissa on vähemmän rivejä kuin asetuksissa on määritetty, uusi arvo kirjoitetaan tiedoston loppuun. Jos rivejä on saman verran tai enemmän kuin asetuksissa määrietty, rivit luetaan ja kirjoitetaan tiedostoon ilman ensimmäistä riviä, minkä jälkeen uusin merkintä lisätään. Kuvassa 17 on havainnollistettu prosessin kulku.



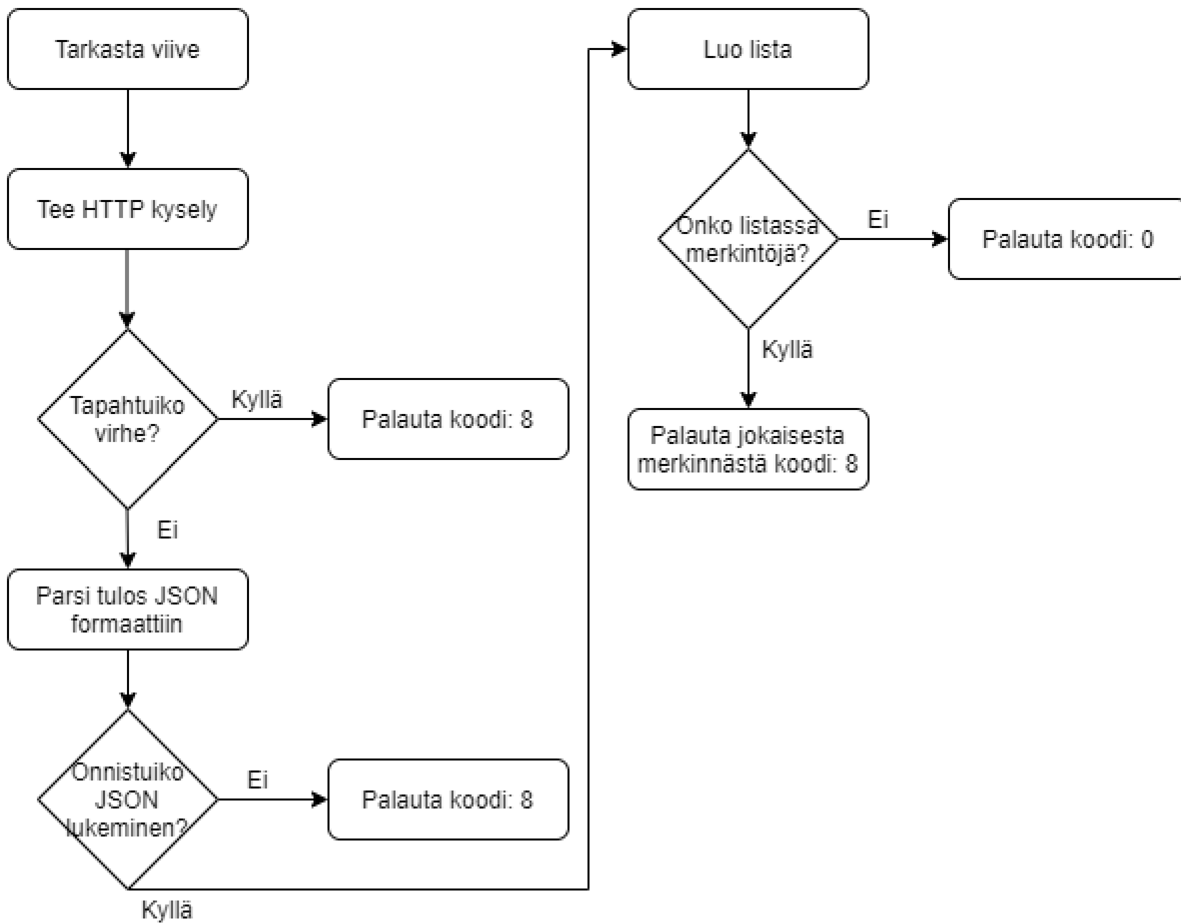
Kuva 17: Laitteiston tarkkailu

SWR-arvojen tarkkailu tapahtuu parsimalla radion lokia. Radio kirjoittaa lokiin lähetykseen liittyvää tietoa, jonka seassa on myös SWR-lohkojen tiedot. Lokia luettaessa otetaan huomioon päivämäärä ja vain edellisen tunnin tiedot luetaan. Jokainen viimeisen tunnin aikana kirjoitettu rivi luetaan ja etsitään tietty merkkijono, joka aloittaa lohkon tiedot. Toisella rivillä on loput lohkon tiedoista, mutta lopettava rivi ei aina ole seuraava rivi. Mikäli lopettavaa riviä ei löydy, ennen kuin seuraava aloittava rivi löytyy, sen hetkisessä lohkojen tietojen etsimisestä luovutetaan. Aloitus- ja lopetusrivin löytyessä SWR-arvo ja lohkon tunnus otetaan ylös, minkä jälkeen arvoa verrataan suurimpiin sallittuihin arvoihin. Arvon ylittyessä hälytyskoodi kirjoitetaan listaan lohkon tietojen lisäksi. Lopuksi hälytyskoodilista käydään läpi ja jokaisesta merkinnästä luodaan hälytys. Kuva 18 esittää prosessin kulun.



Kuva 18: SWR arvojen tarkkailu

COLLECT-palvelun toiminnan seuraaminen tapahtuu tekemällä HTTP-kysely palvelun rajapinnalle. Kyselyn vastaus muunnetaan JSON-muotoon, jonka jälkeen mahdolliset virhetiedot jäsennetään. Palvelussa voi tapahtua erilaisia virheitä ja jokaisesta eri virheestä luodaan oma kirjaus, jotka listataan. On myös mahdollista, että HTTP-kyselyä ei voida luoda, JSON-oliota ei voida lukea tai kyselyn tulosta ei voida käsitellä. Mainitut virheet myös listataan, jotta niistä voidaan luoda hälytys. Kuva 19 havainnollistaa vaiheiden kulun.



Kuva 19: Collect palvelun tarkkailu

Taulukosta 6 voidaan nähdä mitä mikäkin hälytyskoodi merkitsee. Taulun arvot haetaan hälytysten tarkistustiedostosta, johon vastaavan taulun tiedot on tallennettu Pythonin dictionary-muodossa. Koodi 7 ei ole käytössä tässä hälytysjärjestelmän versiossa, ja koodi -1 palautetaan, jos jokin olio kohtaa odottamattoman virheen.

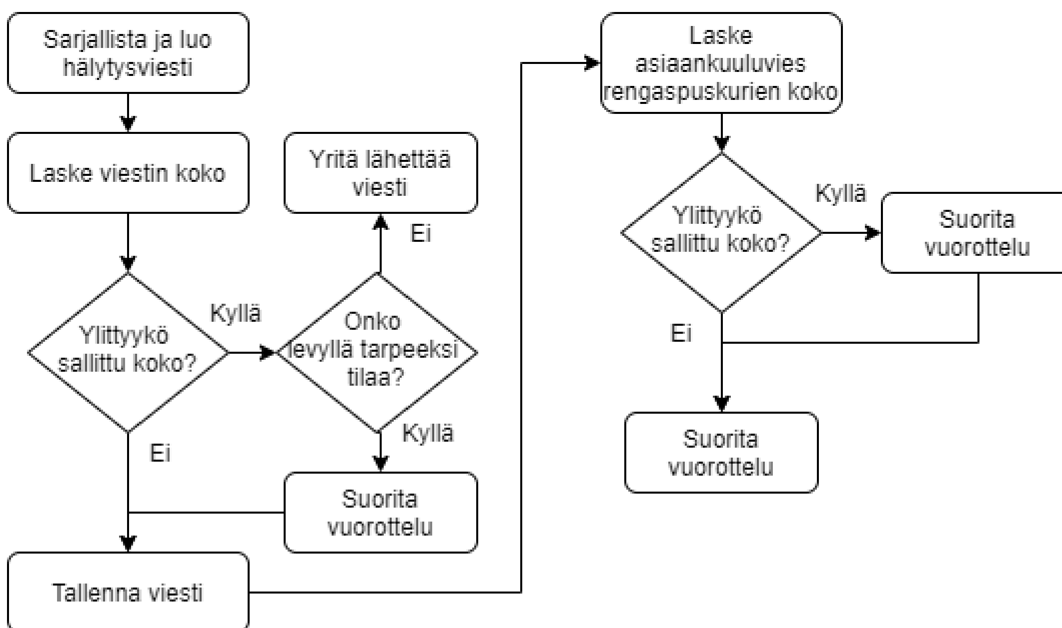
Taulukko 6: Hälytyskoodit

Koodi	Nimi	Kuvaus
-1	Observer Failed	Olio ei pystynyt lukemaan arvoa tai oliota ei alustettu oikein.
0	Heartbeat	Merkitsee, että mikään ei laukaissut hälytystä.
1	High Limit Reached	Ylärajan ylitys.
2	Low Limit Reached	Alarajan ylitys.
3	Sudden Change	Arvon äkillinen nousu.
4	Warning	Varoitustason hälytys.
5	Service Inactive	Palvelu ei ole ajotilassa.
6	Disk Storage Low	Levytila on vähissä.
7	Other Alert	Luokittelematon hälytys.
8	Collect Alert	”Collect”-palvelun luoma hälytys.

Tarkkailijaolio kutsuu ”alert_generator”-moduulin luomaan hälytyksen, jos ”alert”-muuttuja on erisuuri kuin nolla. Samainen metodi luo asetuksissa määritettyyn paikkaan tiedosto nimeltään ”cooldown”, johon kirjoitetaan parametrin numerokoodi ja jonka tarkoitus on estää samaa ”observer”-oliota luomasta hälytystä tietyn ajan. Ennen tätä tiedoston luomista ”check_cooldown”-metodissa tarkistetaan, onko kysesistä tiedostoa jo olemassa, mikä estää kyseisten laitteiston arvoista johtuvien hälytyksien luomisen.

6.1.4 Alert generator-moduuli

Alert generator muodostaa hälytysviestit, sarjallistaa ja tallentaa ne. Hälytyksen luonnin yhteydessä luetaan listasta ne muut parametrit, joita halutaan seurata hälytyksen laukeamisen yhteydessä.
Jokainen ”observer-”olio pitää ”readout_set”-muuttujan sisällä numeerisen arvon, jonka perusteella listausmetodi valitsee listan niistä parametreistä, joiden rengaspuskurihistoriat tallennetaan erilliseen hälytyslokihakemistoon. Tämän hälytyslokihakemiston alle sitten luodaan sisähakemisto, jonka nimeksi tulee hälytyksen laukeamisajankohta ja jonne itse historiatiedostot tallennetaan. Kuva 20 esittää viestin käsittelyn vaiheet.



Kuva 20: Hälytysten tallentaminen ja historian kopiointi

Viesti tallennetaan oletusarvoltaan hälytysjonohakemistoon sarjallistamisen jälkeen odottamaan sen lähetystä, mikäli levytila riittää. Jos levytilaa ei ole riittävästi, hälytysviesti yritetään kerran lähettää tallentamatta.

Viestin sarjallistamisen yhteydessä tarkistetaan myös, että arvot ovat oikeassa muodossa. Kun sarjallistamisfunktio on prosessoinut hälytysviesti, sarjallistettu viesti tallennetaan, minkä jälkeen hälytykselle oleellisten historialokit kopioidaan talteen toiseen hakemistoon.

Historian tallentaminen tapahtuu tarkistamalla muuttuja, jonka arvo merkitsee tiettyä ”readout”-listaa. Tämän perusteella luetaan oikea readoutlist-tekstitiedosto, josta tehdään lista ja joka sitten iteroidaan

läpi. Listassa esiintyvien parametrien nimien mukaiset rengaspuskurilogit tallennetaan ”alert_log”-hakemistoon hälytyksen kansion alle, jonka nimi on hälytyksen laukaisuajankohta.

Lisäksi tarkkaillaan hälytyksen yhteydessä luodun hakemiston kokoa, jonne hälytykselle tärkeät rengaspuskurit kopioidaan. Hakemiston tiedostojenhallinta tapahtuu rengaspuskurin tapaan pudottamalla vanhin sisähakemisto pois uuden tieltä, kun päähakemiston koko on liian suuri. Lokin kiertoa varten on tehty metodi, joka tarkistaa päähakemiston koon ja päivämäärään perustuen valitsee vanhimman tiedoston poistettavaksi.

6.2 Vastaanotto palvelimella

Hälytysviestien vastaanottopuoli vastaanottaa viestit, poistaa viestien sarjallistuksen sekä muodostaa yhteyden tietokantaan viestin tallennusta varten. Vastaanotto koostuu hälytysviestien käsittelystä ja tietokantaan tallennuksesta, joita ”alertreceiver.py”-moduuli käyttää muodostaen vastaanottokokonaisuuden. Kuten lähetyspuolellakin, vastaanottomoduuli lukee kokoonpanoasetuksista arvot ja mahdollistaa niiden helpon muokkaamisen käyttäen aiemmin esiteltyä ”Settings handler”-moduulia. Kokoonpanoasetuksiin kuuluvat ”Stream Receiver” -komponentin alustuksen arvot, joita ovat osoitteet ja portit. Tietokannan yhteyttä varten tiedostossa on myös määritelty tietokannan nimi, taulun nimi ja kirjautumistunnukset.

6.2.1 Alert receiver-moduuli

Kuten lähetysmoduuli, myös ”Alert Receiver” toimii rajapintana, ja se on rakennettu käyttämällä Pythonin moduuleita ”configparser” ja ”arg-parse”. Moduulin pohjana on KNL Networksin kehittämä ”Stream Receiver” -moduuli, jonka metodeilla hälytysviestejä pystytään ottamaan vastaan.

Moduulin alustuksen yhteydessä luodaan gevent-lukko eheyden ylläpitämiseksi, ”Settings Handler” -olio kokoonpanoasetusten lukua varten, aiemmin mainittu ”Stream Receiver” -olio ja yhteys tietokantaan. Alustuksen jälkeen ”Alert Receiver” -prosessi käynnistyy, ja aina vastaanottaessaan hälytyksen callback-metodi käyttää tietokantayhteyttä upottaakseen viestin tiedot tietokannan tauluun.

6.2.2 Alert handler-moduuli

Moduuli ”alerthandler.py” on vastakappale lähetyspuolen ”Alert generator” -sarjallistamismoduuli. Moduulin pääasiallisena tarkoituksena on poistaa hälytysviestin sarjallistaminen samaan tapaan kuin se on tehty, mutta päinvastaisessa järjestyksessä käyttäen ”struct”-moduulia.

Moduulin metodit tunnistavat tyhjäksi määritellyt arvot (None), settavat aikaleiman ja tarkistavat muut arvot, joita ovat parametrin nimi ja hälytyskoodi. Sarjallistamisen yhteydessä tyhjää numeraalista arvoa merkitsee muotoilumerkin ”i” suurin luku ja merkkijonon tyhjää arvoa merkitsee merkkijono ”N/A”. Sarjallistamisen poistamisen jälkeen metodi palauttaa viestin tiedot Python ”dictionary”-olion muodossa.

6.2.3 Database handler-moduuli

”Databasehandler.py”-moduuli pitää sisällään metodit ja skeemat tietokannan transaktioita varten. Se käyttää aiemmin esiteltyä ”alerthandler.py”-moduulia hälytysviestien prosessoimiseen. Moduuli käyttää Pythonin ”psycopg2”-moduulia postgres-tietokannan yhteyden luomiseen ja SQL-komentojen suorittamiseen.

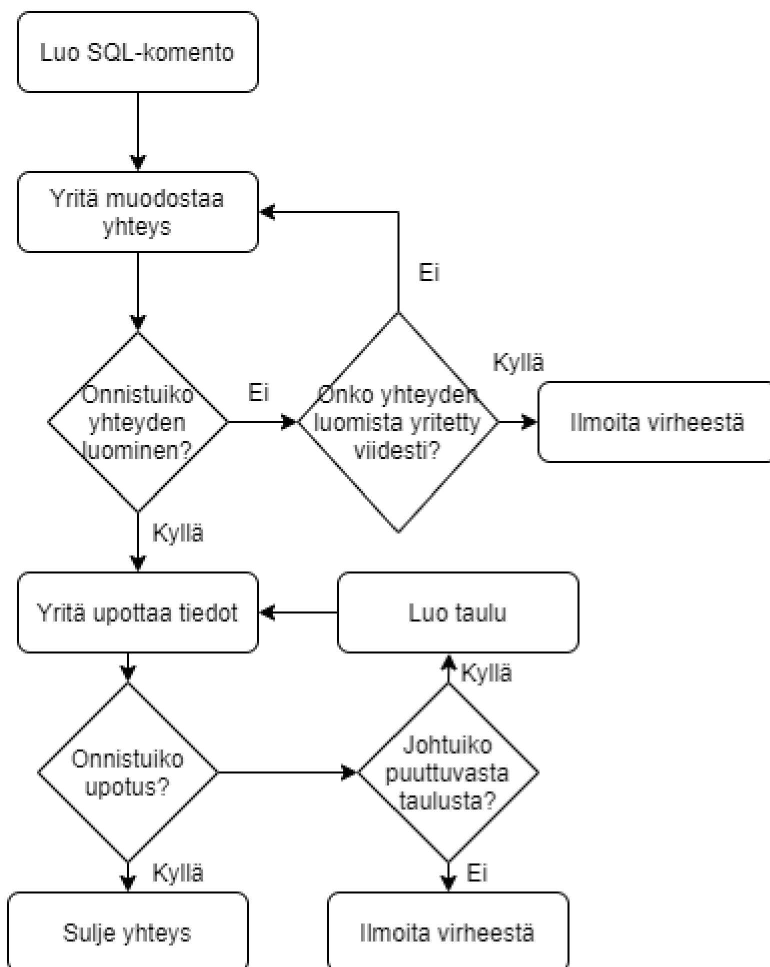
”PSQL Handler”-moduuli alustetaan lukemalla Docker-ympäristön ympäristömuuttujista kirjautumistiedot ja tietokannan nimi. Alustamisen yhteydessä myös yhteys tietokantaan muodostetaan ja luodaan psycopg2:n ”cursor”-olio, jolla voidaan kommunikoida tietokannan kanssa. ”PSQL Handler” -luokan metodi luo hälytysviestistä SQL-komennon.

Metodille annetaan ”alerthandler”-moduulissa Python ”dictionary”-olioksi muutettu hälytysviesti argumenttina ja toisena argumenttina lähettäjän radion tunnistenumeron. Ensimäisenä tarkastetaan, onko numeerinen arvo tai viestin merkkijono tyhjä. Seuraavaksi haetaan hälytyksen numerokoodin perusteella hälytyksen selitys hälytyskoodien tarkistustiedostosta. Kun kaikki koodit ovat purettu, viestin tiedoista muodostetaan SQL-komento.

Tietokantaan luodaan yhteys ”psycopg2”-adapterin ”connect”-metodilla käyttäen argumentteina sisäisiksi muuttujiksi määritettyjä pääsytietoja ja tietokannan tietoja. Jos yhteyden muodostaminen ei onnistu, yhteyden muodostamista kokeillaan viisi kertaa ennen kuin virhesanoma nostetaan ja yhteyden

muodostaminen luovutetaan. Muussa tapauksessa metodi palauttaa tietokantayhteysolion, jolle voidaan sitten antaa SQL-komentoja suoritettaviksi.

SQL-muotoisen upotekomennon luo hälytyksen muunto-metodi, ja sitten luotu SQL-komento yritetään suorittaa. Komento upottaa hälytysviestin arvot tietokannan tauluun, joka on esitelty kuvassa. Mikäli taulua ei ole olemassa, taulu yritetään luoda SQL-komennolla. Lopuksi yhteys tietokantaan suljetaan. Kuvassa 21 havainnollistettu tapahtumat.



Kuva 21: Tietokantaan upottaminen

6.3 Kokoonpanoasetukset

Moduuli ”Settings handler” hallitsee kokoonpanoasetuksia ja on yhteinen sekä lähetys- että vastaanottopuolella. Kokoonpanoasetukset on kirjoitettu csv-, JSON- ja conf-tiedostoihin, joista olioiden ja luokkien alustamiseen vaaditut tiedot ja arvot haetaan.

Nopeaan asetusten muuttamiseen on luotu komentoja vaihtelemaan, seurataanko tiettyä asiaa. Kokoonpanoasetusten säätämiseen ja tietyn parametrin arvon muuttamiseen on myös luotu komennot. Tarkoituksena oli tehdä pienien muutoksien tekemisestä mahdollisimman helppoa. Kuitenkin jos halutaan tehdä paljon muutoksia, on helpompi avata kokoonpanoasetukset hakemistosta ja muuttaa arvot käsin.

7 TESTAUS

Tässä luvussa käydään läpi, miten hälytysjärjestelmää testattiin. Järjestelmän toiminnallisuuden virheiden löytämiseksi tehtiin unit- ja integraatiotestit Pythonin omaa ”unittest”-kirjastoa käyttäen. Unittesteissä keskityttiin datan oikeellisuuden ja eheyden virheiden havaitsemiseen Python-kielen dynaamisen luonteen vuoksi ja havaitsemaan mahdolliset huolimattomuusvirheet. Python muuttujat eivät ole sidoksissa yhteen tietotyyppiin, ja niitä voidaan kesken ohjelman muuttaa, minkä vuoksi muuttujia saatetaan käyttää väärin. (Python software foundation, 2019). Testeissä käytetyt arvot ovat keksittyjä, eivätkä vastaa radiolla käytettäviä oikeita arvoja.

Hyvän testin tulee olla automaattinen, perusteellinen, toistettava, riippumaton ja helposti ymmärrettävä. Automaattisella tarkoitetaan sitä, että testin kutsuminen ja sen tulosten tarkistaminen hoituu ilman ulkopuolista syötettä. Testin on hyvä olla myös nopeasti ajettava, jotta testaaminen on tehokasta. Testin perusteellisuudella tarkoitetaan sen kattavuutta, eli hyvässä testissä mahdollisimman moni seikka on otettu huomioon. Hyvässä testissä ei pelkästään keskitytä tiettyihin bugeihin vaan pyritään ottamaan huomioon monia erilaisia skenaarioita ja testattavan asian poikkeava kulku. (Hunt & Thomas 2011.) Testien täytyy olla myös toistettavissa, eli kirjoitetun testin tulee aina tuottaa tulokset samalla tavalla. Se ei myöskään saa olla riippuvainen muuttujista, joita ei voi hallita. Esimerkiksi aika on muuttuja, joka voi olla ongelmallinen testeissä.

Testien tulee myös olla siistejä ja eristettyjä muusta koodista, mikä tarkoittaa, että kaikki se, mitä testin aikana muutettiin, palautetaan takaisin lähtötilaan. Testien ei myöskään tule testata mitään muuta kuin kehitettyä koodia, eli käytettyjen kolmansien osapuolien koodeja ei oteta huomioon testeissä (Hunt & Thomas 2011).

Kaikissa testeissä pyritään noudattamaan yllä mainittuja hyvän testin ohjeita. Testit on myös rakennettu ”AAA”-periaatteen mukaisesti, joka tarkoittaa Arrange, Act & Assert -mallia. Tämä tarkoittaa, että testeissä ensin määritellään kaikki siinä käytettävät muuttuja, minkä jälkeen tapahtuu kaikki funktioiden ja metodien kutsuminen ja lopuksi katsotaan testin tulokset läpi varmistusmetodeilla. (Wake 2011). Testin vaiheet jaotellaan siis sievästi lukemisen parantamiseksi ja vältetään varmistusmetodeja tai uusien muuttujien esittelemistä kesken testin.

Integraatiotesteissä puolestaan keskityttiin kokonaisuuden testaamiseen. Päästä päähän toiminnallisuuden testaamisessa painopiste oli vian sattuessa oikean käytöksen varmistaminen. Esimerkiksi lähetyksen epäonnistuessa viesti ei saa poistua radion hakemistosta ja vikatilassa prosessien tulee kaatua oikealla tavalla. Lopuksi toiminnallisuutta vielä testattiin fyysisellä radiolla ja simuloidulla pääpalvelimella, mitä tässä työssä ei kuitenkaan esitellä ja avata.

7.1 Yksikkötestit

Hyvä yksikkötesti testaa vain yhtä asiaa kerrallaan. Varmistusvaiheessa voi kuitenkin olla monta varmistusmetodia, kunhan ne kaikki varmistavat vain yhden tietyn asian eri puolia. Tärkeimmät testattavat asiat liittyvät hälytysviestien käsittelyyn, ja ohjelman käyttäytyminen varmistetaan, kun syöte on virheellinen.

7.1.1 Kokoonpanoasetukset ja niiden hallinta

Moduulin ”settingshandler” testaus jakaantuu kolmeen pääryhmään, jotka ovat kokoonpanoasetusten lukeminen, muuttaminen ja uuden kokoonpanoasetusrivin lisääminen. Kokoonpanoasetuksissa on monta eri lohkoa, ja kaikki asetukset haetaan tiedostosta oletukseltaan merkkijonoina, joten on tärkeä testata, että määritetyt arvot muunnetaan oikeaan muotoon. Lähetys- ja vastaanottopuolen kokoonpanoasetusten ”conf”-tiedostojen muokkaaminen ja uusien ”service”- ja ”hw”-tarkkailijoiden kokoonpanoasetusten lisääminen testataan myös.

Testin alussa järjestelymetodissa määritetään uudet ”hw”-luokkaan kuuluvan olion kokoonpanoasetukset ja ne kirjoitetaan tiedostoon. Samalla alustetaan ”settingshandler”-olio. Testin purkamismetodissa määritetään poistettaviksi kaikki testin tekemät tiedostot ja hakemistot.

Ensimmäisessä testissä tiedosto, johon kokoonpanoasetukset on tallennettu, avataan ja varmistetaan arvojen paikkansa pitävyys ”unittest”-kirjaston varmistusmetodilla. Toisessa testissä muutetaan sekä lähetys- että vastaanottopuolen asetuksia ohjelman komentorivikomentojen metodeilla. Tämän jälkeen kokoonpanoasetukset luetaan muuttujaan, minkä jälkeen muutettujen arvojen täsmällisyys varmistetaan. Kolmannessa tarkkailijaolion asetuksia muutetaan metodilla, joka on tarkoitettu kirjoitettavan komentoriville, ja onnistuminen varmistetaan.

Neljännessä testissä tarkistetaan aktiivisuustilan muuttamista komentorivin metodilla. Tiedostoon määritetyn rivin ”active”-sarakkeen on määrä muuttua arvosta yksi arvoon nolla, mikä ilmoittaa alustuksen yhteydessä, käytetäänkö juuri tämän rivin asetuksia olion alustamiseen ollenkaan.

7.1.2 Hälytysviestin käsittely

Hälytyksen käsittelyn testaamiseen on suunniteltu testimoduuli ”test_alert_processing”. Tämän moduulin sisältämät testit jakaantuvat hälytysviestien tallentamiseen ja niiden sarjallistamiseen, eli moduulit ”AlertHandler” ja ”AlertGenerator” ovat tarkastelun alla. Sarjallistamisen yhteydessä rakennetaan muotoilumerkkiyhdistelmä viestin etuliitteeksi, ja sen toiminta varmistetaan. Koko sarjallistamisprosessin toimivuus ja sarjallistamisen poistaminen testataan. Sarjallistamismoduulin tyhjän arvon käsittely ja sarjallistamisen toiminta erityistilanteissa testataan myös. Lisäksi testataan hälytysviestin tallentaminen levylle, sen lukeminen ja historian hallinnan oikein käyttäytyminen.

Testin aloitusjärjestelyssä poistetaan käytettävät hakemistot ja tiedostot, jos nämä ovat olemassa, esimerkiksi aiemman epäonnistuneen testin vuoksi. Sitten luodaan tyhjät kansiot testiä varten, minkä jälkeen luodaan ”settingshandler”-olio ja olion metodilla muutetaan testeissä käytettävät polut löytämään polut, joiden pohjalta testissä käytettävät asetukset luetaan. Testin purkamismetodissa poistetaan luodut kansiot ja tiedostot.

Ensimmäisessä testissä testataan hälytysviestien etuliitteiden luomista. Koska sarjallistamisesta on tehty dynaaminen, tämän testin tarkoitus on havaita virheet tämän dynaamisen pakkausmerkkijonon muodostamisessa. Testi aloitetaan luomalla kolme eri hälytysviestiä ”alertgenerator”-moduulin sarjallistamismetodilla. Viestit luodaan alla taulukon 7 mukaisesti.

Taulukko 7: Testiviestit 1

Viesti	Parametrin nimi	Numero arvo	Merkkijono arvo	Hälytyskoodi	Lukemaryhmä
1	radio_temperature_1	10	”Viesti”	7	0
2	radio_temperature_1	-90	”5”*100”	7	0
3	radio_temperature_1	None	”kaksi sanaa”	7	0

Kun viestit on sarjallistettu muuttujaan, niistä erotetaan pakkauksessa käytettyjen etuliitteiden muotoilumerkkijonot, joiden numero- ja merkkijonoarvojen muotoilumerkki sitten varmistetaan ”assert”-metodilla oikeaksi. Ensimmäinen viesti on tehty mahdollisimman tavalliseksi ja sen

muotoilumerkit kuuluu olla ”i” ja ”6s”. Viestin numero kaksi numeroarvo on negatiivinen, ja sen merkkijonoksi on asetettu seitsemänsataa ”5”-merkkiä. Koska merkkijonoarvon enimmäispituudeksi on määritetty 500, tästä viestistä odotetaan muotoilumerkkejä ”i” ja ”500s”.

Kolmannessa viestissä tarkoituksena on oikean etuliitteen rakentaminen, kun numeroarvoksi määritetään tyhjä arvo ja merkkijonoksi muuta kuin kirjaimia tai numeroita. Tyhjän arvon kohdalla pakkauksen luominen on määritetty asettamaan tyhjän arvon sijasta suurin ”i”-muotoilumerkin kokonaisluku, eli 2147483647. Merkkijonon arvojen tulisi toimia normaalisti. Muotoilumerkeiksi siis odotetaan ”i” ja ”7s”.

Seuraavassa testissä aluksi luodaan viestit samaan tapaan kuten edellisessäkin testissä. Taulukossa 8 on listattu arvot, joilla viestit on määritelty.

Taulukko 8: Testiviestit 2

Viesti	Parametrin nimi	Numeroarvo	Merkkijonoarvo	Hälytyskoodi	Lukemaryhmä
1	radio_temperature_1	100	None	7	0
2	radio_temperature_1	-90	””	6	0
3	radio_temperature_1	None	”X”	1	0
4	radio_temperature_1	None	None	2	0

Tässä setissä tarkastellaan tarkemmin käyttäytymistä, kun viestissä on tyhjiä arvoja. Ensimmäisessä viestissä merkkijonoarvo on asetettu tyhjäksi. Toisessa viestissä merkkijonoarvoksi on asetettu merkkijono, jolla ei ole pituutta. Kolmannessa viestissä numeroarvo on asetettu tyhjäksi ja neljännessä viestissä molemmat arvot ovat asetettu tyhjäksi. Kun merkkijonoarvolle rakennetaan muotoilumerkkijono pakkausta varten, tyhjän arvon ja nollan mittaisen arvon sijasta arvoksi asetetaan ”N/A”. Kun numeroarvo on tyhjä tai jokin muu kuin kokonaisluku, sen arvoksi asetetaan ”i”-muotoilumerkin suurin kokonaisluku. Enimmäisestä viestistä siis odotetaan muotoilumerkkejä ”i” ja ”3s”, toisesta ”i” ja ”3s”, kolmannesta ”i” ja ”1s”, ja viimeisestä ”i” ja ”3s”.

Seuraavassa testissä varmistetaan sarjallistamisen poiston toimivuus. Testissä käytetään juuri samoja viestejä kuin edellisessäkin tehtävässä, ja näiden viestien määritykset voidaan nähdä taulukosta 8. Viestien sarjallistamisen jälkeen ”alerthandler”-olion sarjallistaminen poistetaan olion ”deserialize_alert”-metodilla. Kun sarjallistaminen on poistettu, viestien arvot varmistetaan ”assert”-funktiolla. Ensimmäisen viestin numeroarvojen tulee olla 100 ja sen merkkijonoarvon tulee olla ”None”. Toisen viestin arvojen tulee olla -90 ja ”None”, kolmannen ”None” ja ”X”, ja neljännen viestin molempien arvojen tulee olla ”None”.

Kolmannessa testissä tarkastetaan käyttäytyminen erityistilanteissa. Kuten edellisissäkin testeissä, viestit ensin sarjallistetaan ”alertgenerator”-olion metodilla. Taulukossa 9 on esitetty, miten viestit on määritetty.

Taulukko 9: Testiviestit 3

Viesti	Parametrin nimi	Numeroarvo	Merkkijonoarvo	Hälytyskoodi	Lukemaryhmä
1	””	100	None	7	0
2	radio_temperature_1	”X”	””	6	0
3	radio_temperature_1	None	”ääö”	2	0
4	radio_temperature_1	None	8	1	100

Ensimmäiseen viestiin on määritetty parametrin nimi puuttuvaksi, toisessa viestissä numeroarvoksi on asetettu merkki ja kolmannen viestin merkkijonon arvoksi on asetettu ääkköset ”ääö”. Neljännessä viestissä merkkijonon arvoksi on asetettu kokonaisluku ja viestin lukemaryhmäksi on asetettu luku, jonka perusteella lukemaryhmähakemistosta ei löydy tiedostoa.

Sarjallistetuista viesteistä poistetaan sarjallistaminen, jonka jälkeen arvoja vertaillaan. Ensimmäisen viestin kohdalla parametrin nimeksi odotetaan ”unknown”, joka liitetään viestiin sarjallistamisen yhteydessä, jos viestin tarkkailun kohteen nimeä ei löydy sen tarkistustiedostosta. Toisessa viestissä numeroarvoksi todetaan jokin muu kuin kokonaisluku ja sen arvoksi asetetaan sarjallistamisen yhteydessä muotoilumerkin ”i” suurin arvo, joka luetaan sarjallistamisen poiston yhteydessä tyhjäksi arvoksi. Kolmannen viestin merkkijonoarvo puretaan tavallisesti ja arvoksi saadaan ”ääö”. Neljännessä viestissä lukemaryhmää ei löydetä hakemistosta, joten sille on asetettu sarjallistamisen yhteydessä arvo 0. Merkkijonoon asetettu kokonaisluku puretaan tavallisesti takaisin merkiksi ”8”.

Seuraavassa testissä ”alertgenerator”-olion viestinluomismetodille ”generate_alert” syötetään arvot ”radio_temperature_1”, 599, ”Koira”, 2, 1. Viestin luomisen jälkeen tallennushakemisto avataan, viesti luetaan muuttujaan ja siitä poistetaan sarjallistaminen, minkä jälkeen todetaan molempien arvojen paikkaansapitävyys ja se, että viesti todella on tallennettu hakemistoon.
Viimeisessä testissä luetaan valmiina olevien rengaspuskuritiedostojen koko, joka tallennetaan muuttujaan. Sitten lasketaan, kuinka monta tällaista muuttujaa mahtuisi lokihistoriahakemistoon.

7.1.3 Tarkkailija

Tarkkailijaolioiden yksikkötesteissä varmistetaan viiveiden toiminta, dynaamisen luokan tuomisen toimivuus ja näiden luokkien ”inspect”-metodien toimivuus. Tässä testissä tarvitaan ”Scheduler”-moduulin muokattua alustusmetodia, joka alustaa vain yhden tarkkailijaolion. Tarvitaan myös ”SettingsHandler”-moduulia kokoonpanoasetusten muokkaamiseen ja ”AlertHandler”-moduulia viestien purkamiseen. Mainittujen moduulien pohjalta alustetaan oliot testin alussa, minkä jälkeen määritetään kokoonpanoasetusten polut löytämään testien kokoonpanoasetustiedostot.

Tämän jälkeen määritetään kolme eri Pythonin dictionary-oliota määrittämään ”observer”-oliot, jotka myöhemmin alustetaan. Taulukossa 10 on esitetty nämä testiä varten keksityt määritysarvot.

Taulukko 10: Testitarkkailijoiden määritykset 1

	ob_setting_1	ob_setting_2	ob_setting_3
type	hw	disk	service
start_delay	10	5	5
parameter	radio_temperature_1	service1	disk1
sample_size	10	None	None
low_limit	0	None	None
high_limit	100	None	None
warning_low	1	None	None
warning_high	99	None	None
sample_interval	2	2	2
difference_limit	10	None	None
cooldown	2	10	10
readout_set	1	0	0

Kuten aiemmissakin testeissä, testin aloitusjärjestelyssä siivotaan hakemistot, jotka ovat mahdollisesti jääneet lojumaan, ja luodaan sitten testissä käytettävät hakemistot. Testiä varten on myös kirjoitettu metodi ”write_history”, jonka avulla voidaan kirjoittaa ja simuloida rengaspuskuria testejä varten.

Ensimmäisessä testissä varmistetaan moduulien dynaamisen tuomisen toimivuus. Testi aloitetaan alustamalla jokaisen dictionary-olion pohjasta tarkkailijaolio, jonka jälkeen ”assert”-funktiolla varmistetaan, että ensimmäinen olio on instanssi ”HW_Observer”, toinen ”Service_Observer” ja kolmas on ”Disk_Observer”.

Seuraavassa testissä testataan olion alustamisen käyttäytymistä, kun se yritetään alustaa parametrin nimellä, jota ei löydy tarkkailijoiden tarkistustiedostosta. ”Observer”-olio alustetaan ”ob_setting_2”-dictionaryn mukaan, jonka jälkeen olio kutsuu metodia luomaan hälytyksen. Tämän jälkeen tallennettu

viesti luetaan hakemistosta ja sarjallistaminen poistetaan. Lopuksi varmistetaan, että parametrin nimi on ”unknown”, ja se asetetaan niille parametreille, joita ei ole kirjoitettu tarkistustiedostoon.

Testissä ”test_cooldown” testataan hälytyksen aiheuttaman viiveen toimivuus. Testi käynnistyy alustamalla ”ob_setting_1” pohjalta ”observer”-olio, joka luo hälytyksiä ”issue_alert”-metodillaan. Metodi toistetaan neljä kertaa ja jokaisella kerralla annetaan parametrille ”alert” eri arvo tunnistamisen helpottamiseksi testin lopussa. Ensimäiselle metodille syötetään arvo 7, toiselle arvo 6, kolmannelle arvo 5 ja neljännelle arvo 4. Hälytyksenluomismetodien toistojen välissä käytetään ”sleep”-metodia arvolla 1, eli toiston jälkeen aina odotetaan yksi sekunti ennen seuraavan alkua. Koska määrityksessä annettiin ”cooldown”-parametrin arvoksi 2 ja ”start_delay”-parametrin arvoksi 0, toistojen, joille annettiin arvot 5 ja 7, odotetaan tallentuneen. Testin lopuksi kaikki hakemistoon sarjallistetuista ja tallennetuista viesteistä luetaan ja niistä poistetaan sarjallistaminen. Sitten varmistetaan, että viestit, joiden ”alert”-arvot ovat 5 ja 7, löytyvät tallennettuina.

Seuraavaksi testataan aloitusviiveen toimivuus ”test_start_delay”-testillä. Kuten muutkin testit, tämäkin aloitetaan alustamalla ”observer”-olio, ja tämän olion alustamiseen käytetään ”ob_setting_1” dictionary:n parametrejä. Seuraavaksi olion aloitusviive asetetaan kymmeneen sekuntiin ja hälytysviive ”cooldown” asetetaan nollaksi. Tämän jälkeen kutsutaan testin ”write_history”-metodia kirjoittamaan rengaspuskuri, minkä jälkeen olion hälytyksen luomismetodia toistetaan yhdessä viisi kertaa yhden sekunnin välein. Koska rengaspuskureiden kirjoittamisessa menee kahdeksan sekuntia ja olion metodit toistetaan viiden sekunnin välein, testi lopetetaan varmistamalla, että kolme hälytysviestiä on tallennettu hakemistoon.

7.2 Integraatiotestit

Integraatiotestit keskittyvät testaamaan rajapintojen välisiä toimintoja, ja näillä testeillä pyritään mallintamaan tosielämän ympäristössä tapahtuvaa toimintaa mahdollisimman hyvin. Integraatiotesteissä luodaan erilaisia tilanteita, joista ohjelman tulee selvitä kadottamatta tärkeää tietoa ja olla aiheuttamatta ei-toivottuja tiloja.

Vikatilojen tai virheiden sattuessa kaikkien tiedostojen tulee jäädä järkevään tilaan. Palauduttuaan vikatilasta ohjelman tulee myös jatkaa toimintojaan ja kesken jääneitä asioita sujuvasti. Kaikkia mahdollisia käytössä ilmeneviä virheitä on hankala mallintaa, mutta näissä integraatiotesteissä testataan tärkeimpien vikatilojen käsittely.

Kaikki integraatiotestit suoritetaan Docker-ympäristössä, jonka KNL Networks on rakentanut mallintamaan yhden tai useamman radion toimintaa. Näiden virtuaalisten säiliöiden avulla testaus voidaan pitää erillään muusta koodista. Integraatiotesteissä käytetään KNL Netwrorks Oy:n jo valmiiksi kehittämiä ”composetool”-, ”templates”- ja ”testhelpers” -tiedostoja, jotka auttavat integraatiotestien kirjoittamista ja Docker-työkalun hallintaa. Testejä varten on myös luotu omat kokoonpanoasetukset, joissa on määritetty polut niin, että suoritettavat testit ovat täysin erillään muusta koodista.

7.2.1 Aloitusasettelu ja purkaminen

Kaikkien integraatiotestejen alussa luodaan virtuaalinen radio ja palvelin, johon radio voi ottaa yhteyden. Testin alussa määritetään kolmelle eri ”observer”-oliolle testiin mukautetut asetukset Python dictionary-olioina. Asetukset ovat nähtävillä alla taulukossa 11.

Taulukko 11: Testitarkkailjoiden määritykset 2

	ob_settings_1	ob_settings_2	ob_settings_3
parameter	”radio_temperature_1”	”bsselector”	5
type	”hw”	”service”	”NOTHING”
start_delay	0	0	
sample_size	10	0	0
low_limit	0	0	None
high_limit	90	0	”None”
warning_low	0	0	0
warning_high	80	0	”X”
sample_interval	2	2	2
history_range	550	0	0
difference_limit	10	0	0
cooldown	5	5	-9
readout_set	1	0	0
active	1	1	1

Ensimmäisessä oliossa on määritelty hyvin tavalliset laitteiston arvojen seuraamiseen tarkoitetut asetukset. Toisessa oliossa on asetukset ”bsselector”-palvelun tilan tarkkailemiseen. Viimeisessä oliossa on satunnaisesti keksitty virheellisiä arvoja, joilla tarkkailija alustetaan. Testien aloitusasettelussa luodaan väliaikainen kokoonpanoasetustiedosto asetuksien muutosten palauttamisen helpottamiseksi. Alussa luodaan myös hakemistot hälytysviestien tallentamista, lokin tallentamista ja rengaspuskurien kirjoittamista varten.

Näiden toimenpiteiden jälkeen ”settingshandler”-moduulin avulla kokoonpanoasetuksiin määritetään virtuaalisen radion ja palvelimen asetukset yhteyden luomista varten. Seuraavaksi määritetään ja luodaan yhteys tietokantaan käyttämällä ”dbhandler”-moduulin metodeja. Viimeiseksi määritetään purkamistoimenpiteet, joissa virtuaaliset radiot puretaan ja luodut hakemistot poistetaan tiedostoineen.

7.2.2 End-to-end

Ensimmäisenä integraatiotestinä on kirjoitettu kaikkea perustoiminnallisuutta demonstroiva päästä päähän-testi. Testin tarkoituksena on varmistaa, että kaikki ohjelman osat toimivat yhdessä. Testi alkaa luomalla mukautettu callback-funktio, jota kutsutaan hälytysviestin saapuessa vastaanottokomponentille. Funktiossa haetaan vastaanottopuolen tietokantaan tallettama viesti SQL-komennolla. Viestin ominaisuudet sitten varmistetaan vastaamaan odotettuja arvoja, minkä jälkeen viesti poistetaan SQL-komennolla tietokannasta. Viimeiseksi tapahtuma kuitataan.

Sitten alustetaan instanssit lähetys- ja vastaanottokomponenteista. Vastaanottopuolen ajometodi asetetaan ”greenlet”-olioon, minkä jälkeen lähetyspuolen metodia kutsutaan ajamaan tarkkailurutiini läpi. Tämän jälkeen lähetyspuoli lähettää mahdolliset tallennetut viestit ja tapahtumaolio odottaa testissä määritetyn funktion kuittausta tapahtumasta. Hälytysviestin onnistuneen lähetyksen yhteydessä tapahtuu vastakutsu, joka poistaa levylle talletetun viestin. Tämän jälkeen vastaanotto lopetetaan ja testi on päättynyt.

7.2.3 Ei hälytyksiä

Tässä perustoimivuutta varmistavassa testissä testataan käyttäytyminen, kun mitään hälytyksiä ei ole luotu lähetettäväksi. Testin alussa lähettäjän ja vastaanottajan alustuksen yhteydessä alustetaan tarkkailijaolio taulukon ”ob_settings_1” mukaisilla asetuksilla. Ennen tarkkailijaolion alustamista sen ’warning_high’ arvo asetetaan saavuttamattoman korkeaksi, ettei se laukaise hälytystä.

Hälytyksen vastakutsuun asetetaan käsitellyn viestin haku tietokannasta ja varmistetaan haetun viestin syyn olevan ”Heartbeat”. Kun testi on ajettu ja ’heartbeat’-viesti käsitelty, muutetut arvot muutetaan takaisin normaaliksi.

7.2.4 Epämääräinen kokoonpanoasetus

Seuraavassa testissä testataan järjestelmän käyttäytymistä epätäydellisiä kokoonpanoasetuksia luettaessa. Kuten aiemmassakin testissä, lähetys- ja vastaanottokomponentit alustetaan ja hälytyksen saapumisen yhteyteen asetetaan vastakutsu. Tällä kertaa vastakutsussa vain kuitataan viestin saapuminen tapahtumaoliolle. Tässä testissä lähetyspuolen tarkkailuolio alustetaan taulukon ”ob_settings_3”

mukaisilla asetuksilla ja olion tarkkailurutiini ajetaan, minkä jälkeen lähetyspuoli lukee ja lähettää mahdolliset viestit.

Epätäydellisen tarkkailuolion alustaminen onnistuu muuten käsittelemällä epämääräiset ja puuttuvat tiedot, mutta itse tarkkailtavan arvon polun puuttuessa 'observer'-olio luo tästä tavallisen hälytyksen sijaan hälytyksen koodilla ”-1”, joka tarkoittaa virheellistä hälytystä. Testi loppuu, kun vastakutsu kuittaa saadun viestin selityksen olevan ”Observer failed” eli virheellinen hälytys.

7.2.5 Vähäinen levytila

Vähäinen levytila on järjestelmän erikoistila, jonka sattuessa hälytysjärjestelmä ei saa tallentaa hälytysviestiä radiolle. Käyttäytyminen tässä tilanteessa on määritelty niin, että hälytysjärjestelmä pyrkii kerran lähettämään luodun viesti radiolta KNL networksin tietokantaan tallentamatta viestiä.

Testin alussa lähetyspuolen kokoonpanoasetuksiin järjestelmän pienin sallittu koko muutetaan arvoksi 1, joka tarkoittaa sataa prosenttia. Tarkoituksena on luoda tarkkailijaoliolle mahdoton vertailutilanne, eli olion verratessa vapaan levytilan prosentuaalista arvoa sataan hälytys laukeaa aina.

Viestin saapuvaan vastakutsuun liitetään tietokantakysely, joka varmistaa, että viestissä on odotetut arvot. Olio käynnistetään taulukon ”ob_settings_1” mukaisilla arvoilla, minkä jälkeen varmistetaan, ettei hakemistoon ole tallennettu hälytystä. Tämän jälkeen vastakutsu varmistetaan kuitatuksi ja testin päätteeksi muutetut arvot muutetaan takaisin lähtötilaan.

7.2.6 Yhteys katkeaa

Viallista yhteyttä testataan kahdella tavalla kahdessa eri testissä. Ensin testataan käyttäytyminen, kun yhteys palvelimeen katkaistaan juuri ennen lähetystä ajamalla virtuaalinen palvelin alas. Sitten testataan käyttäytyminen radion yhteyksienhallitsijan kaatuessa. Näissä testeissä on oleellista, että vastakutsua onnistuneesta lähetyksestä ei ikinä tule radion lähettäjäkomponentille ja näin ollen viestiä ei poisteta radiolta.

Radion yhteyksien hallitsija ajetaan alas testin aluksi ja palvelun kaaduttua radiolta yritetään lähettää hälytysviesti. Epäonnistuneen lähetyksen jälkeen varmistetaan, että luotua hälytysviestiä ei ole poistettu

ja että hälytysviestiä ei ole vastaanotettu ja kuitattu. Testin lopuksi yhteyksiä hallitseva palvelu käynnistetään uudelleen.

Toisessa viallisen yhteyden testissä vastaanottopuolen Docker-säiliö pysäytetään, kun hälytysjärjestelmän vastaanottokomponentti on käynnistetty pysäyttäen koko vastaanoton. Vastaanotetun viestin vastakutsuun kirjoitetaan vain kuittaus siltä varalta, että viesti kuitenkin menee läpi. Hälytysviesti luodaan ja yritetään lähettää, minkä jälkeen varmistetaan, ettei hälytysviestiä ole poistettu eikä vastakutsua ole kutsuttu kuittaamaan viestiä.

8 YLLÄPITO

Uusi hälytysjärjestelmä tarvitsee pääpirtteittäin vain vähän ylläpitoa, joka liittyy tietokannan hallintaan ja kokoonpanoasetusten muuttamiseen. Ensimmäisessä hälytysjärjestelmässä oli erillinen järjestelmä, kuten luvussa kolme esiteltiin, ja sitä vastaavaa järjestelmää ei uuteen hälytysjärjestelmään kehitetty. Koska tämä automaattinen valvonta puuttuu palvelimen päässä, tietokantaa tulee seurata manuaalisesti ja tarkistaa, mitä uudet kirjaukset pitävät sisällään.

Kun radiolla otetaan käyttöön jokin olemassa oleva aktivoimaton palvelu, jota voi seurata, se tulee aktivoida manuaalisesti. Tämän voi tehdä lopettamalla hälytysjärjestelmä, minkä jälkeen ohjelma voidaan ajaa komennolla ”activate”, joka ottaa argumenttina tarkkailijan nimen. Kun kaikki muutokset on tehty, hälytysjärjestelmä käynnistetään uudelleen ja muutokset tulevat voimaan. Hälytyksen raja-arvojen muuttaminen onnistuu yksinkertaisesti muuttamalla kokoonpanoasetuksia ”edit”-komennolla, joka ottaa argumentteina tarkkailijan nimen, muutettavan arvon ja uuden arvon.

Kaikki muutokset voidaan myös suoraa tehdä tarkkailijan kokoonpanoasetustiedostoon. Lähetys- ja vastaanottokomponenttien kokoonpanoasetusten muuttaminen onnistuu myös suoraa muuttamalla komponenttien kokoonpanoasetuksia.

9 JATKOKEHITYS

Hälytysjärjestelmä on kehitetty jatkokehitysmahdollisuuksien helppoutta ajatellen. Vaatimusmäärittelyssä määriteltiin, että kehitettävän järjestelmän tulee olla skaalautuva uusia palveluita ajatellen ja muita ominaisuuksia ajatellen. Tämä vaatimus täytettiin hyvin, sillä esimerkiksi uutta palvelun tilaa voidaan seurata pelkän kokoonpanoasetustiedoston muuttamisella.

9.1 Uudet hälytykset

Hälytysjärjestelmää voidaan laajentaa nopeasti seuraamaan uusia laitteistoin sensoreita tai palveluita. Kun halutaan lisätä uusi laitteiston toimintaa seuraava tarkkailija, täytyy vain lisätä uusi rivi tarkkailijoiden kokoonpanoasetustiedostoon.

Uusia tarkkailijoita määrittäessä tulee kiinnittää erityistä tarkkuutta siihen, että kaikki tarkistustiedostossa esiintyvät tarkkailijoiden nimien tunnistuskoodit ovat uniikkeja. Kahta tarkkailijaa ei saa merkitä saman tunnistuskoodin alle.

Koodit väliltä 1 – 99 on varattu palveluiden tunnisteiksi, koodit väliltä 101 – 199 on varattu laitteiston arvojen tunnisteiksi, ja koodit kahdestasadasta eteenpäin ovat sekalaisia tässä hälytysjärjestelmän versiossa. Tätä luokittelua ei ole koodissa pakotettu, vaan se on ohjenuora manuaalisen ylläpidon helpottamiseksi.

Rivillä tulee olla tarkkailtavan arvon nimi niin kuin se on esitetty järjestelmässä ja tyyppi, joka tässä tapauksessa on ”hw”. Aktiivisuutta osoittavan sarakkeen tulee olla joko 0 tai 1, ja tarkastustiheyttä esittävä ”sample_interval” tulee määrittää. Muut sarakkeet voidaan tarpeen mukaan joko määrittää halutuksi arvoksi tai jättää arvoksi 0, tai None. Tämän jälkeen tarkkailtavan arvon nimi tulee kirjoittaa tarkistustiedostoon niin palvelimen kuin radionkin päässä.

Uuden palvelun tilaa tarkkailevaa tarkkailijaa varten tulee samaan tapaan kirjoittaa uusi rivi tarkkailijoiden kokoonpanotiedostoon ja tarkistustiedostoon. Palvelun nimen tulee ilmetä tarkkailijan nimeä merkitsevässä kentässä, tarkkailijan tyypin tulee olla ”service”, tarkkailuaikaväli tulee määrittää ja aktiivisuus merkitä joko arvoksi 0 tai 1.

9.2 Uuden tyyppiset hälytykset

Täysin uusia hälytyksiä luodessa täytyy paneutua kokoonpanoasetuksia syvemmälle ja tehdä uusi moduuli, jos tarkkailijan käyttäytyminen poikkeaa jo olemassa olevien tarkkailijaluokka käyttäytymisestä. Yksinkertaisuudessaan, ”observer” tarlkkailijaluokka aliluokitetaan ja tarkkailijaluokan ”inspect”-metodi kirjoitetaan uudestaan. Alla on yksinkertaistettu esimerkki siitä, miten uusi metodi voidaan kirjoittaa.

Esimerkissä on totuuskysely levytilan koosta. Jos koko on yli määritetyn rajan, metodi palauttaa kokonaisluvun 3, joka tarkoittaa rajan ylitystä. Samalla se kirjoittaa sisäisiin muuttujiin ”str_value” ja ”num_value” lisää tietoa tilanteesta. Mikäli levytilakysely menee läpi eli pysyy sallituissa rajoissa, metodi palauttaa arvon 0, joka kertoo hälytyksenluontimetodille, ettei mitään tarvitse tehdä. Metodin määrittelyssä on siis paljon joustavuutta, mutta palautettavan arvon tulee aina olla kokonaisluku, joka on määritetty tarkistustiedostossa ”alertLookup.py”.

Moduulin, jossa on aliluokitettu tarkkailijapääluokka ja määritelty uuden tarkkailijan käyttäytyminen, nimi on tämän uuden tarkkailijan tyyppi. Tarkkailijoiden kokoonpanoasetuksissa tulee tyyppisarakkeeseen kirjoittaa moduulin nimi, ilman Python:ille ominaista tiedostopäätettä. Muuten samaan tapaan, kuin muidenkin uusien tarkkailijoiden määrittelyssä tulee tarkkailijan nimi, aktiivisuustila ja tarkkailuväli merkitä. Tarkkailijoiden tarkistustiedostoon vielä merkitään luodulle tarkkailijalle uniikki tunniste, minkä jälkeen luotu tarkkailija on valmis käyttöön.

9.3 Uudet ominaisuudet

Ominaisuus, joka jäi rajauksen ulkopuolelle, oli hälytysten julkaiseminen KNL Networksin ERP-järjestelmään. Ominaisuus on tärkeä valvonnan automatisoinnin kannalta, joten tämä ominaisuus olisi luonnollinen seuraava askel. Ominaisuuden kehittäminen ei ole kovin iso työ, sillä siihen on jo pohja olemassa alkuperäisestä hälytysjärjestelmästä.

Hälytysten jälkikäsittelyn vuoksi, järjestelmään tai sen rinnalle voisi kehittää ohjelman, joka automaattisesti tekee yhteenvetoraportteja hälytysdatasta. Tiettyjen hälytysten esiintymistiheydestä esimerkiksi voi päätellä, että esimerkiksi jokin radion laitteiston osa ei toimi oikein. Yhteyksiä hälytysten esiintymiseen voisi verrata myös maantieteelliseen sijaintiin. Kattavan raportoinnin

kehittäminen olisi oma ohjelmansa, mutta esimerkiksi hälytystiheyden huomioinnin voisi liittää kehitettyyn hälytysjärjestelmään.

Kehitetty hälytysjärjestelmä luo lyhyitä historialokeja ja kirjoittaa ne talteen radiolle. Näitä historialokeja ei kuitenkaan lähetetä ollenkaan radiolta KNL-networks:in palvelimelle, sillä se kuormittaa verkkoa liikaa tärkeämmän tiedonsiirron sijasta. Tulevaisuudessa hälytysjärjestelmän luomia lokitiedostoja voisi lähettää, kun radio tunnistaa yhteyden olevan tarpeeksi nopea.

10 YHTEENVETO

Työn tavoitteena oli suunnitella ja kehittää moniajoa tukeva hälytysjärjestelmä, jota voi mielivaltaisesti laajentaa yksinkertaisella tavalla. Valmis tuote tuli testata hyvin sen toimivuuden takaamiseksi. Järjestelmää varten täytyi myös tehdä tietokantaan uusi taulu, jonne hälytysviestit voisi tallettaa jatkotoimenpiteitä varten.

Yksityiskohtaisempina tavoitteina voidaan luetella, että uuden järjestelmän täytyi pystyä seuraamaan palvelun tilaa ja radion laitteiston toiminnallisia arvoja. Seurattaville asioille tuli olla määritettävissä olevat kokoonpanoasetukset, joita voi tarpeen tullen muuttaa. Näistä seurattavista asioista järjestelmän tuli myös luoda, lähettää ja käsitellä hälytysviestit. Järjestelmän tuli kirjoittaa hälytysviestit levylle odottamaan lähetystä. Laitteiston arvoista järjestelmän tuli kirjoittaa rengaspuskuria ja hälytyksen yhteydessä määrätyt rengaspuskurit tuli ottaa talteen. Kaikkea kirjoittamista varten tuli myös olla kiertologiikka levyn täyttymisen ehkäisemiseksi. Työ täytyi kehittää Python 2.7 -kieltä käyttäen Linux-pohjaiselle käyttöjärjestelmälle, ja työ tuli tehdä niin, että sen pystyi ajamaan Docker-säiliön sisällä. Kuitenkin työssä oli muuten vapaat kädet. käytettävien työkalujen suhteen.

Työn alussa tehtiin päätös, että tulisi luoda kokonaan uusi järjestelmä, sillä arkkitehtuuriin tulisi paljon muutoksia, vaikka alkuperäistä järjestelmää lähtisi muuttamaan. Työ lähti käyntiin vertailemalla eri toteutustapoja, kuten rengaspuskurin toteuttamisen suhteen. Käytettävät Python-kirjastot valittiin ja päästiin toiminnan suunnittelemiseen. Kun ensimmäinen suunnitelmaluonnos oli valmis ja esitetty palaverissa, työtä lähdettiin viemään eteenpäin koodaamalla. Suunnitteluvaiheen palaverit olivat hyvin tärkeitä, ja niissä kuultiin operaatiotiimin ja tutkimus- ja kehitystiimin näkemyksiä, jotka olivat esinarvoisen tärkeitä työn laadun kannalta.

Työn edetessä käytiin palavereita tietyistä ominaisuuksista. Eräässä palaverissa todettiin, että olisi järkevä hajauttaa ”observer”-oliot alaluokkiin koodin selkeyttämisen vuoksi ja jatkokehitystä ajatellen. Loppuvaiheella päätettiin, että kehitettävään hälytysjärjestelmään tulisi myös kehittää uusi tarkkailutapa collect-palvelulle. Toinen ominaisuus tuli myös kehittää radioaaltojen takaisinheijastuksen arvoja seuraamaan.

Työ kuitenkin noudatti kaikkiaan hyvin suunnitelmaa ja tavoitteita. Uuteen järjestelmään voidaan lisätä uusia tarkkailtavia asioita muuttamalla kokoonpanoasetuksia. Mikäli tarvitaan täysin uudenlainen

tarkkailija, sen logiikka voidaan suoraan liittää valmiiseen runkoon. Tallennettavat asiat tallentuvat oikein, ja järjestelmän tarkkailijaoliot toimivat rinnan.

Työhön jäi lieviä puutteita. Esimerkiksi ”alertsender.py” ja ”alertreceiver.py” kumpikin hoitavat itse argumenttien jäsentämisen. Olisi järkevämpää, jos mainitun asian hoitaisi erillinen moduuli. Toisena parannuskehotuksena järjestelmä olisi voinut käyttää globaaleja muuttujia hälytysviiveen ja aloitusviiveen kanssa tiedostojen kirjoittamisen sijaan.

Loogisen eheyden ylläpitämiseksi järjestelmässä luodaan alustusvaiheessa yksi lukko, jonka saadessaan olio voi kirjoittaa tiettyyn rengaspuskuriin tai kopioida tietyt rengaspuskurit. Tässä ongelmana skaalautuvuutta ajatellen on lukon hankkimisen ja vapauttamisen hitaus, kun ajatellaan todella suurta määrää laitteistoa seuraavia tarkkailijoita. Yhtenä ratkaisuna voisi olla uusi moduuli, joka hoitaa kirjoittamisen ja kopioimisen olion puolesta. Tämä uusi moduuli voisi luoda tiedostokohtaiset lukot, jolloin tarkkailija, joka ei tarvitse tiettyä tiedostoa, ei estä tiedoston kopioimista tai siihen kirjoittamista.

Työssä käsiteltiin hälytysjärjestelmän koko elinkaari suunnittelusta kehitykseen ja käyttöönottoon asti. Työssä oli paljon laajoja kokonaisuuksia, ja työ venyi hahmotellusta aikataulusta melkein kuukaudella. Hälytysjärjestelmää kehitettäessä täytyi opetella paljon uusia asioita ja asiakokonaisuuksia, minkä takia työ oli hyvin tärkeä seuraavia projekteja varten. Kaiken kaikkiaan työ onnistui hyvin ja se otetaan käyttöön joulukuussa.

Tulevaisuutta ajatellen hälytysjärjestelmään tai sen rinnalle voi luoda hälytysten aiheuttaman rengaspuskurihistorian lähetyksen. Tämä helpottaisi hälytysten analysoimista. Toisena ominaisuutena voisi olla hälytysraporttien automaattinen generointi. Radion järjestelmän etävalvonnan kehittämistä ajatellen tämän työn tuote on hyvä runko. Hälytysjärjestelmään voidaan liittää helposti uusia asioita, mikä helpottaa näiden uusien asioiden toiminnan seuraamista.

Tämän työn aikana oppi todella paljon uusia asioita ja se kehitti opittuja asioita. Työssä oli mukana tuotteen koko elinkaari, joten projektin ja prosessien hallinta kehittyi. Teknologioita ja työkaluja tuli tutuksi, kuten moni työssä käytetty Python-kirjasto ja Docker-virtuaaliympäristön hallinta. Koodin tuottaminen ja sen laatu paranivat myös huomattavasti. Tuotteen testaamisesta kehittyi parempi ymmärrys, ja hyviin käytäntöihin alkoi kiinnittämään enemmän huomiota.

LÄHTEET

Admin's Choice. 2019. Crontab – Quick Reference. (Saatavissa: https://www.adminschoice.com/crontab-quick-reference Viitattu: 21.8.2019)

Avro Big Data. 2019. Data Serialization and Evolution. Confluent Inc. (Saatavissa: https://docs.confluent.io/1.0/avro.html Viitattu 28.10.2019)

Berg N., Kanto E., Rantanen A. 2019. Digitalization as a tool to reduce GHG emissions in maritime transport. Traficom Research Reports 28/2019 (Viitattu: 2.12.2019)

Bhowmick, S. 2014 "Command Injection/Shell Injection" Independent Consulting Security Evangelist (Saatavilla: https://www.exploit-db.com/docs/english/42593-command-injection---shell-injection.pdf Viitattu: 20.11.2019)

Bilenko, D. 2019. Gevent Docs. (Saatavissa: http://www.gevent.org/contents.html Viitattu: 17.8.2019)

Brouwer, A. 2003. Linux Programmer's Manual. (Saatavissa: http://man7.org/linux/man-pages/man3/statvfs.3.html Viitattu: 28.9.2019)

Checkmarx Ltd. 2018. Python Security Vulnerabilities and Language Overview. (Saatavissa: https://www.checkmarx.com/sast-supported-languages/python-security-vulnerabilities-and-language-overview/ Viitattu: 8.10.2019)

Di Gregorio F. 2019. psycopg2 - Python-PostgreSQL Database Adapter. (Saatavilla: https://pypi.org/project/psycopg2/ Viitattu: 10.12.2019)

Docker Inc. 2019. Docker Documentation. (Saatavissa: https://docs.docker.com/ Viitattu: 15.11.2019)

Dunning, T & Friedman, E. 2015. Time Series Databases. 1. painos. O'Reilly Media. Yhdysvallat. Viitattu: 25.8.2019)

Fabrizio, G. 2013. High Frequency Over-the-Horizon Radar: Fundamental Principles, Signal Processing, and Practical Applications. McGraw-Hill Education; 1 edition

Giesbrect, J. 2008. Aspects of HF Communications: HF Noise and Signal Features. Tohtorin tutkinnon tutkielma. Saskatchewan Yliopisto. (Saatavissa: https://www.researchgate.net/publication/233531539_Aspects_of_HF_Communications_HF_Noise_and_Signal_Features Viitattu: 21.9.2019)

Git. 2019. About. (Saatavilla: https://git-scm.com/about Viitattu 10.12.2019)

Hunt A, Thomas D. 2010. Pragmatic Unit Testing in Java with JUnit. The Pragmatic Programmers. (Saatavissa: https://doc.lagout.org/programmation/Pragmatic%20Programmers/Pragmatic%20Unit%20Testing%20in%20Java%20with%20JUnit.pdf Viitattu: 20.10.2019)

KNL Networks. 2019a. KNL Corporate Master. (Viitattu: 18.9.2019)

KNL Networks. 2019b. KNL File Product Sheet. (Viitattu: 18.9.2019)

KNL Networks. 2019c. KNL Mail Product Sheet. (Viitattu: 5.8.2019)

KNL Networks . 2019d. KNL Radio Product Sheet. (Viitattu: 11.9.2019)

KNL Networks. 2019e. KNL Track Product Sheet. (Viitattu: 14.9.2019)

KNL Networks. 2019f. KNL WaveAccess COLLECT Product Sheet. (Viitattu: 26.8.2019)

Microsoft. 2019. Why did we build Visual Studio Code? (Saatavilla: https://code.visualstudio.com/docs/editor/whyvscode Viitattu: 10.12.2019)

Mällinen, J. Kyberturvalliset radioaallot haastavat satelliittiyhteydet. Oulun Yliopisto. Oulu. (Saatavissa: https://www.oulu.fi/yliopisto/node/49762 Viimeksi päivitetty: 6.9.2018. Viitattu: 9.9.2019)

Møgelberg, R. 2011. Transactions and ACID properties: Introduction to Database Design. (Saatavissa: https://itu.dk/~mogel/SIDD2011/lectures/SIDD.2011.13.pdf Viitattu: 21.8.2019)

Pearson Education. 2018. Transaction Management. University of Highlands and Islands. (Viitattu: 17.10.2019)

pgAdmin. 2019. FAQ. (Saatavilla: https://www.pgadmin.org/faq/ Viitattu: 10.12.2019)

Python Software Foundation. 2019a. asyncio — Asynchronous I/O. (Saatavissa: https://docs.python.org/3/library/asyncio.html Viitattu: 16.8.2019)

Python Software Foundation. 2019b. pickle — Python object serialization. (Saatavissa: https://docs.python.org/3/library/pickle.html Viitattu: 20.9.2019)

Python Software Foundation. 2019c. Struct — Interpret strings as packed binary data. (Saatavissa: https://docs.python.org/2/library/struct.html Viimeksi päivitetty: 19.8.2019 Viitattu: 1.9.2019)
Raschka, S. 2014. An introduction to parallel programming using Python's multiprocessing module.

Siemens, M. 2016. Project description TinyDB. PYPI. (Saatavissa: https://pypi.org/project/tinydb/ Viitattu: 20.9.2019)

Space Weather Services. 2016. Introduction to HF Radio Propagation. Australian Government Bureau of Meterology (Saatavissa: http://www.sws.bom.gov.au/Category/Educational/Other%20Topics/Radio%20Communication/Intro%20to%20HF%20Radio.pdf Viitattu: 9.9.2019)

Wada, K. 2013. Ring buffer basics. Embedded.com. (Saatavissa: https://www.embedded.com/ring-buffer-basics/ Viimeksi päivitetty: 7.8.2013Viitattu: 20.8.2019)

Wake B. 2011. 3A – Arrange, act, assert. XP123 (Saatavissa: https://xp123.com/articles/3a-arrange-act-assert/ Viitattu: 20.12.2019)

Watson R. N. M. 2005 Introduction to Multithreading and Multiprocessing in the FreeBSD SMPng Network Stack. University of Cambridge. (Saatavissa: http://www.watson.org/~robert/freebsd/netperf/20051027-eurobsdcon2005-netperf.pdf Viitattu: 30.12.2019)

Willis J. 2015. Docker and the Three Ways of DevOps. Docker. (Saatavissa: https://goto.docker.com/rs/929-FJL-178/images/20150731-wp_docker-3-ways-devops.pdf Viitattu: 29.11.2019)

Zope Foundation. 2016. ZODB Tutorial. (Saatavissa: http://www.zodb.org/en/latest/tutorial.html Viitattu: 20.9.2019)