VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Masi Rinne

ANVIAN REAGOINTIJÄRJESTELMÄ

Tekniikka ja liikenne
2012

VAASAN AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

TIIVISTELMÄ

Tekijä	Masi Rinne
Opinnäytetyön nimi	Anvian Reagointijärjestelmä
Vuosi	2012
Kieli	suomi
Sivumäärä	43
Ohjaaja	Ghodrat Moghadampour

Tämän opinnäytetyön lähtökohtana oli tarve Anvia Oyj:n valvomossa seurata kaikkia hälytyksiä samassa näkymässä sekä hälytysten niputtaminen ja hälytysraporttien tulostus. Tällä hetkellä käytössä olevat sovellukset mahdollistavat hälytysten vastaanoton vain tietyistä laitteista ja valvomon työntekijät joutuivat seuraamaan hälytyksiä useasta eri näkymästä. Tällä hetkellä käytössä olevissa sovelluksissa ei myöskään ole mahdollista niputtaa hälytyksiä eikä raporttien tulostusmahdollisuutta. Anvian reagointijärjestelmää lähdettiin kehittämään uutena sovelluksena jossa olisi vanhojen käytössä olleiden sovellusten perustoiminnot, mutta lisäksi siinä olisi halutut uudet ominaisuudet.

Tässä raportissa käsitellään Anvian reagointijärjestelmän toteutusta, toteutuksessa käytettyjä tekniikoita ja Anvian reagointijärjestelmän toiminnallisuuksia. Myös Anvian reagointijärjestelmän toimintaympäristöä käydään läpi sekä Anvian reagointijärjestelmän käyttämää tietokantaa ja rajapintoja. Anvian reagointijärjestelmän toteutuksessa käytetty kehitysympäristö on Aptana Studio 3, joka on suunniteltu erityisesti PHP ja JavaScript–ohjelmointikielien kehitykseen. Anvian reagointijärjestelmän toteutuksessa käytettiin pääosin seuraavia tekniikoita: PHP, JavaScript ja MySQL. Anvian reagointijärjestelmän käyttöympäristö tulee olemaan Anvia Oyj:n sisäverkossa, ja sen käyttäjät yrityksen henkilökuntaa. Monet Anvian reagointijärjestelmän toiminnallisuuksista ovat yhteydessä Anvian toisiin järjestelmiin eri rajapintojen kautta ja niiden avulla välitetään tietoa hälytyksistä toisiin järjestelmiin.

Anvian reagointijärjestelmän käyttämään tietokantaan tallennetaan tietoa eri tilanteissa ja tiedonhaku on pyritty tekemään mahdollisimman nopeaksi. Osaa tietokantaan tallennetuista tiedoista käytetään vain senhetkisten hälytysten esittämiseen näkymässä ,mutta raportin luontiin tarvitaan tietoa pidemmältä ajalta. Raportin luontiin tarvittava tieto päätettiin tallentaa omiin tietokantatauluihin, jotta sovelluksen tietokantahakujen ajat pysyisivät lyhyinä.

Avainsanat PHP, JavaScript, AJAX

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Masi Rinne
Title	Anvias reagointijärjestelmä
Year	2012
Language	Finnish
Pages	43
Name of Supervisor	Ghodrat Moghadampour

This thesis was based on Anvia Ltd control room with the need arising to be able to monitor all the alerts on the same view and also make alert bundles and create alert reports. At present there is much view in the control room to be monitored at once. At present there is not the possibility of bundle the alerts or create alert reports. Anvias Reagointijärjestelmä was motivated and developed in a new application that has basic functions for the old systems, but also new defined functions.

This report deals with realization of the Anvias Reagointijärjestelmä, using engineering and Anvias Reagointijärjestelmä functionalities. The Anvias Reagointijärjestelmä environment is also discussed along with the database and interface that Anvias Reagointijärjestelmä uses. Aptana Studio 3 was the IDE of this project, it is designed specifically for development PHP and JavaScript– programming languages. The following languages were applied in implementing Anvias Reagointijärjestelmä: PHP, JavaScript and MySQL. Anvias Reagointijärjestelmä intranet is the environment were Anvias Reagointijärjestelmä will be used by the Anvia staff. The multiple features of Anvias Reagointijärjestelmä include the ability to communicate with other Anvia systems via different interfaces and pass information on about the alerts.

Data that Anvias Reagointijärjestelmä use is stored in a database in different kind of situations. In addition, the database communication strives to be as fast as possible. Part of the database data is using just Anvias Reagointijärjestelmä alert view, but report tools need data for long period. Report data was to be saved according to decision made in -their own database tables, so applications database search would be as fast as possible.

Keywords:PHP,JavaScript,AJAX

SISÄLLYS

TERMIT JA LYHENTEET

Ajax	Ajax (Asynchronous JavaScript And XML) on joukko web–sivujen toteutuksiin käytettäviä tekniikoita. Esim. web–sivujen päivitys voidaan toteuttaa Ajax - tekniikoiden avulla.
Funktio	Ohjelmoinnissa funktiolla tarkoitetaan rakennetta, joka sisältää toiminnallisuuden ja sitä voidaan kutsua aina tarvittaessa. Funktiolle voidaan välittää parametrejä ja se voidaan määritellä palauttamaan vastaus kutsujalle.
JavaScript	JavaScript on ohjelmointikieli jota käytetään web-sivujen selainympäristön toteutuksissa, katso luku 3 JavaScript.
jQuery	jQuery on JavaScript -ohjelmointikielen suosittu kirjasto, joka sisältää erilaisia toiminnallisuuksia mm. AJAX-tekniikan käyttöön.
Muuttuja	Muuttujia käytetään lähes kaikissa ohjelmissa. Yleensä muuttujiin tallennetaan merkkijonoja tai lukuja. Kun ohjelmassa on luotu muuttujia, jotka on nimetty sisältöä hyvin kuvaaviksi, helpottaa se koodin lukemista ja muokkaamista.
MySQL	MySQL–tietokantaohjelmisto on yleisesti käytössä web-palveluiden tietokantana. MySQL -tietokantaohjelmistosta on saatavana sekä ilmainen että maksullinen versio.
PHP	PHP (Hypertext Preprocessor) on ohjelmointikieli jota käytetään pääosin web–sivujen palvelinympäristön toteutuksissa, katso luku 2 PHP.

1 JOHDANTO

Tietoliikenneverkkojen toiminnan kannalta keskeisiä ovat aktiivilaitteet. Isoissa tietoliikenneverkoissa vikatilanteita ilmenee päivittäin ja yksittäinen vika voi vaikuttaa suureenkin asiakasmäärään. Verkon suunnittelulla pystytään rajaamaan yksittäisen vikatilanteen vaikutusta verkkoon ja aktiivilaitteiden uusimisella voidaan ennaltaehkäistä vikatilanteiden syntymistä. Myös tietoliikenneverkon ulkopuoliset tekijät vaikuttuvat verkon vikatilanteiden syntymiseen, näitä ovat mm. sähkökatkot, sääilmiöt ja maanrakennustyöt.

Tämän työn toimeksiantaja Anvia Oyj on tietoliikenneyhtiö, jolla on pitkät perinteet puhe- ja tietoliikenneverkkojen rakentamisessa ja ylläpidossa entisen Vaasan Läänin alueella. Tässä opinnäytetyössä toteutettua Anvian reagointijärjestelmää tullaan käyttämään hälytysnäkymänä Anvia Oyj:n valvomossa ja siihen toteutettavilla toiminnallisuuksilla reagoidaan erilaisiin hälytyksiin. Anvian reagointijärjestelmällä pystytään käsittelemään verkon aktiivilaitteista tulevia hälytyksiä ja tekemään tarvittavat toimenpiteet. Hälytysten nopea käsittely on tarpeellista, koska osa hälytyksistä vaatii välittömiä toimenpiteitä. Anvian reagointijärjestelmä myös helpottaa hälytysten käsittelyä ja hallinnointia. Hälytykset esitetään Anvian reagointijärjestelmän hälytysnäkymällä siten, että niitä on mahdollisimman helppo seurata ja ryhmitellä. Hälytyksissä käytetään erilaisia taustavärejä jotka määräytyvät hälytyksen kiireellisyyden sekä tilan mukaan. Niputtamisella hoidetaan hälytysten ryhmittely ja sillä voidaan käsitellä laajempaa vikatilannetta yhtenä kokonaisuutena.

Anvian reagointijärjestelmään haluttiin myös toiminnallisuus jolla voidaan tarkastella eri hälytyksiä ja niiden tilaa jälkikäteen. Raportoinnilla voidaan luoda dokumentti josta näkee Anvian reagointijärjestelmässä luodut niput ja niissä olleet hälytykset eri tilanteissa. Raportointi oli yksi Anvian reagointijärjestelmälle asetetuista keskeisistä vaatimuksista hälytysten niputtamisen ohella.

2 KÄYTETYISTÄ TEKNIIKOISTA

2.1 PHP

PHP (Hypertext Preprocessor) -ohjelmointikieli on kehitetty vuonna 1995 Kanadalaisen Rasmus Lerdorfin toimesta. Aluksi hän kehitti yksinkertaisen kävijälaskurin omille kotisivuilleen, mikä laski kävijöiden määrän ja näytti sivuilla tuloksen. Koska vuonna 1995 ei ollut juurikaan kävijälaskurin kaltaisia sovelluksia olemassa, herätti se runsaasti mielenkiintoa. Lerdorf kehitti kävijälaskurin Perl-ohjelmointikielen ja CGI-rajapinnan avulla, mutta nimesi kehittämänsä ”työkalusarjan” *Person Home Page* (PHP) ja alkoi jakamaan sitä halukkaille. PHP:n kehittäminen ei jäänyt siihen Lerdorfin osalta vaan hän lähti parantelemaan ja jatkokehittämään uusia ominaisuuksia PHP:lle. Jatkokehitysvaiheessa Lerdorf käyttä C-kieltä Perlin sijaan. Kun vuonna 1997 julkaistiin PHP-versio 2.0, oli siinä mm. ominaisuus tiedon siirtämiseen HTML-lomakkeelta muihin järjestelmiin. PHP 2.0 julkistamisen jälkeen useat ohjelmoijat ympäri maailmaa alkoivat kehittää parannuksia ja laajennuksia PHP–kieleen. Vuonna 1997 julkaistu versio sai myös nykyisen nimensä kun Personal Home Page muutettiin Hypertext Preprocessoriksi. Nimen muuttaminen oli helppoa, koska yleisesti käytetty lyhenne PHP säilyi ennallaan. Uusi ydin, Zend Engine julkaistiin PHP:lle vuonna 1999 ja sen kehittivät Andi Gutmans ja Zeev Suraski. Vuosina 1997-1999 PHP:n suosio kasvoi räjähdysmäisesti ja 90-luvun lopussa PHP:n käyttäjiä oli arviolta yli miljoona. PHP:n suosio ja kielen laaja käyttö oli kehittäjille yllätys, mikä lisäsi myös paineita uusien parannuksien käyttöönottoon. Vuosi 2000 oli PHP:n kehityksen kannalta merkittävä, koska silloin julkaistiin versio 4. Version 4 keskeisimmät parannukset olivat kehittynyt resurssien hallinta, tuki oliopohjaiselle toiminnalle, HTTP-istuntohallinta, MCrypt-kirjasto, yhteensopivuustuki IIS Web-palvelimen kanssa, tuki COM-objekteille sekä Java-tuki. Edellä mainitut parannukset mahdollistivat PHP:n käytön laajamittaisissa yrityssovelluksissa paremman skaalautuvuutensa ansiosta. Uusien käyttäjien oli helpompi omaksua PHP-versio 4 oliopohjaisen toiminnallisuuksien vuoksi. PHP- versio 4 pyrittiin parantamaan erityisesti Win-

dows yhteensopivuutta, tästä osoituksena IIS Web -palvelimen ja COM/DCOM-objektien tuet. Uudet ominaisuudet saivat myös osakseen kritiikkiä pian version 4 julkistamisen jälkeen, mutta pääosin uuteen versioon oltiin tyytyväisiä. PHP-versio 4 tuki päättyi virallisesti vuonna 2008. Vuonna 2005 oli julkaisuvuorossa PHP 5, jonka ytimenä oli Zend Engine II. Osin versio 5 täydensi version 4 ominaisuuksia, mutta uusiakin ominaisuuksia versiosta 5 löytyi. PHP 5 keskeisimpiä parannuksia oli oliopohjaisten ominaisuuksien kehittyminen, poikkeuksien hallinta, merkkijonojen hallinta, SQLite-tuki sekä pitkälle kehittynyt XML- ja Web Services-tuki. Version 5 ei tuonut yhtä radikaaleja parannuksia kuin aiempi versio 4, mutta se oli joka tapauksessa merkittävä julkaisu. Uusista ominaisuuksista Try/catch-poikkeuksien hallinta oli ollut jo aiemmin monissa ohjelmointikielessä kuten C++, C#, Python ja Java, joten se oli varmasti monelle PHP:n käyttäjälle mieluisa ominaisuus. Aiempaa laajempi tuki oliopohjaiseen ohjelmointiin paransi olioiden hallintaa, mahdollisti olioiden kloonauksen sekä luokan abstraktion.

Uutta ohjelmaa suunniteltaessa saattaa jo olemassa oleva ympäristö vaatia tietyn ohjelmointikielen käyttöä. Jos ohjelmointikieli voidaan vapaasti valita, tärkeimpinä kielen valintaperusteina on kieleltä vaadittavat ominaisuudet. PHP:n käytännöllisyyden vuoksi nousee se yleensä potentiaaliseksi ehdokkaaksi web-sovellusta suunniteltaessa. PHP sisältää runsaasti valmiita funktioita, esimerkiksi päivämäärien ja merkkijonojen käsittelyssä. Kun C-kielessä joudutaan kirjoittamaan itse kymmeniä rivejä pitkä funktio, saattaa PHP:ssä riittää pelkkä yhden rivin funktiokutsu samaan toimenpiteeseen. Funktioita voidaan myös sisentää PHP:ssä, jolloin koodista tulee tiiviimpää ja näin myös helpommin luettavaa. Muuttujien tietotyyppejä ei PHP:ssä tarvitse määritellä, koska kieli on heikosti tyypitetty. Kun esimerkiksi Javassa pitää määritellä muuttujien tietotyypit, osaa PHP itse määritellä tietotyypin muuttujan arvon perusteella. Tietotyypin määrittely onnistuu myös PHP:ssä, mutta sitä käytetään harvoin. Ainoat rajoitteet muuttujan nimeämisessä on että sen pitää alkaa $ -merkillä eikä seuraava merkki saa olla numero. Skandinaaviset merkit on sallittuja muuttujien nimissä. Myöskään järjestelmän resurssien vapauttamista ei tarvitse erikseen määritellä muuttujien osalta vaan

PHP osaa vapauttaa ne itse koodin suorittamisen jälkeen. PHP:n monipuolisuudesta kertoo sen laaja tuki eri tietokantoihin. Tuki löytyy ainakin seuraaville tietokannoille: Adabas D, dBase, Empress, FilePro, FrontBase, Hyperwave, IBM DB2, Informix, Ingres, Interbase, mSQL, direct MS-SQL, MySQL, Oracle, Ovrimos, PostgreSQL, Solid, Sybase, Unix dbm ja Velocis. Laaja tietokantatuki ja mahdollisuus käyttää oliopohjaista tai funktionaalista mallia takaa osaltaan PHP:n suuren suosion. Se sopii myös mainiosti ympäristöihin joissa on käytössä useita eri tietokantoja. Kynnys PHP:n käyttöön on matala myös koska se on open source–ohjelmisto. /1/

2.2 JavaScript

JavaScript on ohjelmointikieli jota käytetään verkkosivuilla, yleensä muiden ohjelmointikielien lisänä. JavaScriptillä ei ole juurikaan tekemistä Java–ohjelmointikielen kanssa, vaikka nimi niin antaa ymmärtääkin. HTML–sivuilla JavaScriptiä käytetään yleisesti, mutta myös PHP–sivujen interaktiivisuutta parannellaan sillä. Alun perin JavaScript–koodi sisällytettiin pääohjelmointikielen sisään, mutta nykyään suositellaan erillistä js–tiedostoa josta toiminnallisuuksia pystytään kutsumaan. Kun JavaScript–koodia lisätään toisen ohjelmointikielen sisään, aloitetaan se <script>-tagilla ja lopetetaan </script>-tagilla. Näin selain tietää tagien sisällä olevan koodin JavaScriptiksi. JavaScriptillä voidaan tehdä verkkosivuille toiminnallisuuksia, joita ei pelkällä HTML–kielellä pysty toteuttamaan. Toimintoja pystytään toteuttamaan myös ilman palvelinta, jolloin esimerkiksi laskutoimitukset tapahtuvat asiakas- (selain) päässä. Myös sivujen ulkoasun muuttaminen, käyttäjän valintojen mukaan on toteutettavissa JavaScriptillä. JavaScript mahdollistaa siis erilaisia toimintoja asiakaspäässä, mutta palvelimella sen toiminta turvallisuussyistä on estetty. Tiedon tallentamiseen palvelimelle tarvitaan oma ohjelma, joka käsittelee ja tallentaa halutut tiedot. Asiakaspään toimintoihin on asetettu myös tiettyjä rajoituksia JavaScriptissä. Tiedostojen kirjoittaminen ja lukeminen on estetty, mutta evästeiden luonti on mahdollista. Tietyt toi-

minnot kuten selain–ikkunoiden sulkeminen ja vieraiden verkkosivujen lukeminen on myös osittain estetty. /2/

3 PROSESSIN KULKU

3.1 Vaatimusten määrittely ja analysointi

Vaatimukset Anvian reagointijärjestelmään tulivat Anvia Oyj:n valvomosta, joka toimi myös sovelluksen tilaajana. Vaatimusten pohjana oli vanhat, käytössä olevat valvontanäkymät. Anvian reagointijärjestelmää suunniteltaessa oli ajatus, että käytössä olisi vain yksi näkymä, johon kaikki hälytykset koottaisiin. Hälytykset pitää myös järjestää niiden kriittisyyden mukaan sekä hälytyksistä pitää pystyä lähettämään kuittaus kun se on huomioitu. Hälytysten taustaväri näkymällä pitää muuttua, hälytyksen senhetkisen tilan mukaan. Eri hälytyksiin reagoidaan eri tavalla ja hälytyksissä täytyy olla myös mahdollisuus lähettää tietoa hälytyksestä muihin järjestelmiin.

Hälytysten esittämisen lisäksi hälytyksiä pitää pystyä niputtamaan. Kaikki näkymään ilmestyvät hälytykset pitää olla mahdollista siirtää nippuun. Jos nippua ei vielä ole olemassa, täytyy sellainen pystyä luomaan ja siirtämään hälytys siihen. Samaan nippuun pitää pysyä siirtämään useita hälytyksiä, riippumatta hälytyksen kriittisyydestä. Nipussa olevia hälytyksiä pitää pystyä poistamaan tai siirtämään toiseen nippuun. Myös tieto nipussa olevista hälytyksistä pitää pystyä tallentamaan aina tarvittaessa. Kun nippua ja sen hälytyksiä ei enää tarvitse seurata, pitää nippu pystyä sulkemaan. Nipun hälytykset pitää tarvittaessa pystyä piilottamaan, jolloin nipun viemä tila näytöltä pienenee.

Päivitys Anvian reagointijärjestelmän hälytysnäkymässä pitää tapahtua asynkronisesti, jolloin vain tietty osa näkymästä päivitetään. Myös eri toiminnallisuudet pitää toteuttaa asynkronisesti, jolloin hälytyksiä pystyy siirtämään nippuun tai hälytysten tilaa vaihtamaan ilman koko näkymän uudelleen lataamista.

Yksi keskeinen vaatimus oli raporttien tulostus tallennetuista hälytysnipuista. Raportti pitää pystyä luomaan kaikista nipuista joita on luotu ja niissä tallennushetkellä olleista hälytyksistä. Raportoinnissa pitää pystyä luomaan Word-asiakirja,

jolloin raporttiin pystyy lisäämään tai muuttamaan tietoa. Myös nippujen hakuun raporttia luodessa pitää luoda työkalu, jotta pystytään hakemaan haluttu nippu.

Vaatimusten analysoinnissa tarkasteltiin annettuja vaatimuksia ja sitä mitkä vaatimuksista olisi mahdollista toteuttaa. Koska työn tilaaja tunsi hyvin toteutusympäristön, lähes kaikki annetut vaatimukset oli mahdollista toteuttaa. Vaatimuksista valittiin keskeisimmät ja tärkeimmät toiminnallisuudet joita lähdettäisiin ensimmäisenä toteuttamaan. Osa vähemmän tärkeistä lisäominaisuuksista päätettiin jättää opinnäytetyön ulkopuolella, koska muuten työstä olisi tullut liian laaja. Vaatimusten rajaus tehtiin työn tilaajan toiveiden mukaan ja opinnäytetyön ulkopuolelle jätetyt ominaisuudet tullaan toteuttamaan myöhemmässä vaiheessa.

Opinnäytetyön keskeiset vaatimukset lueteltuna QFD:n mukaisesti:

Pakolliset ominaisuudet: (must have)

- asynkronisesti päivittyvä hälytysten koontinäkymä
- hälytysten tilan muuttaminen
- hälytysten niputtaminen ja nippujen tallennus
- tallennettujen nippujen haku ja raporttien luonti
- hälytysten järjestäminen kriittisyyden mukaan
- hälytysdatan hakeminen kahdesta keskeisestä järjestelmästä
- toiminnallisuudet nippuihin: sulje, tallenna ja suurenna/pienennä
- rajapintojen toteutus hälytysdatan hakemiseen
- tietokannan suunnittelu ja toteutus.

Toivotut ominaisuudet: (should have)

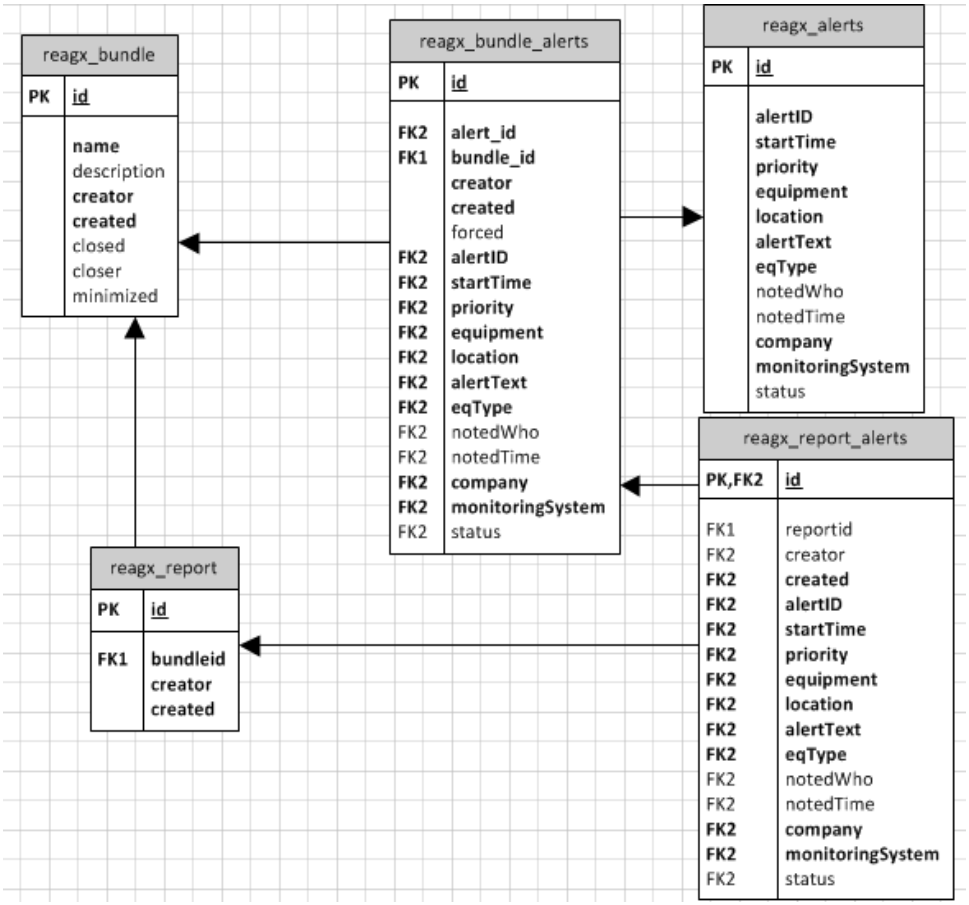
- hälytystietojen välitys rajapintojen kautta muihin järjestelmiin
- hälytysten taustaväri hälytyksen kriittisyyden ja tilan mukaan
- hälytysnäkymän asynkroninen päivitys muutoksen jälkeen.

Ei pakolliset ominaisuudet: (nice to have)

- hälytystietojen välitys sähköpostiin
- hälytystiedotteen luonti intranet-sivulle
- nipun tilan automaattinen tallennus
- asiakasmäärien haku hälytyksille.

3.2 Tietokannan suunnittelu ja toteutus

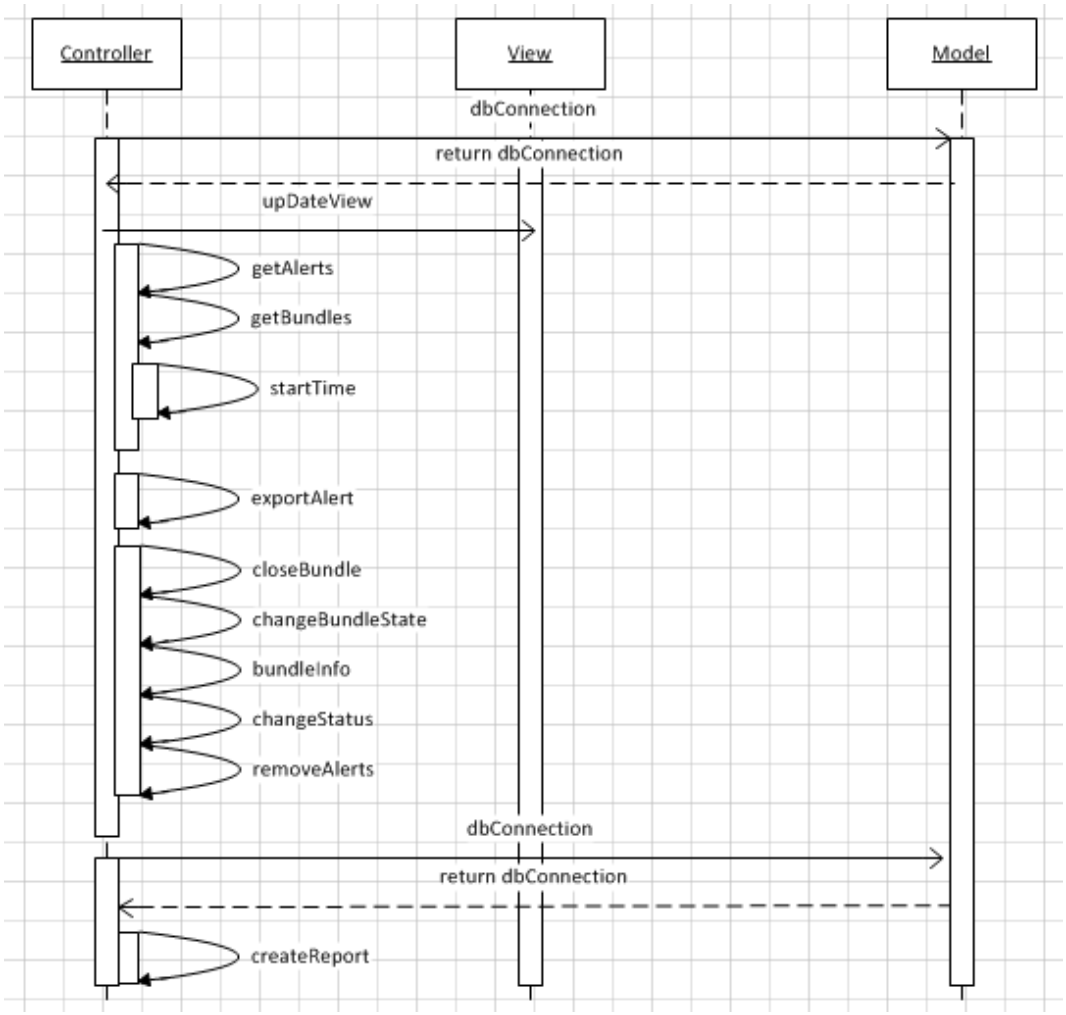
Jotta itse sovellusta voidaan lähteä suunnittelemaan ja toteuttamaan, pitää olla tietokanta josta hakea ja johon tallentaa tietoa. Tietokannan suunnittelussa pitää ottaa kolme eri näkökulmaa huomioon: normaali hälytysdatan tallennus, nippujen tallennus sekä raporttidatan tallennus. Kaikissa kolmessa tilanteessa pitää tallentaa samoja tietoja, mutta käytettävyyden kannalta oli parasta toteuttaa tallennus eri tauluihin eri tilanteissa. Raporttidataa joudutaan säilyttämään pidempiä aikoja ja sitä kertyy paljon, joten sen tallentaminen omiin tauluihin nopeuttaa sovelluksen tietokantahakuja hälytysnäkymän päivityksessä (**Kuva 1**). Anvian reagointijärjestelmän käyttämät tietokantataulut. Reagx_alerts–taulusta haetaan näytettävien hälytysten data ja reagx_bundle_alerts–tauluun tallennetaan nipuissa olevat hälytykset. Reagx_bundle–tauluun tallennetaan nippujen metatieto. Reagx_report_alerts- sekä reagx_report–tauluihin tallentuu raporttien tiedot.



Kuva 1. Anvian reagointijärjestelmän tietokantataulut ja niiden muuttujat

3.3 Sovelluksen suunnittelu ja toteutus

Sovelluksen suunnittelu käynnistyi annettujen vaatimusmäärittelyjen sekä käyttöympäristön perusteella. Toiminnallisuuksien suunnittelu perustui annettuihin vaatimusmäärittelyihin, mutta myös vanhoista järjestelmistä oli apua suunnittelussa. Vanhojen järjestelmien hyväksi todetut ominaisuudet pyrittiin hyödyntämään suunnittelussa ja yhdistämään niitä uusiin ominaisuuksiin. Kuvassa 2 Anvian reagointijärjestelmään toteutetut keskeisemmät funktiot. StartTime–funktiota kutsutaan sivua ladattaessa sekä määräajoin ajastetusti, funktion sisällä kutsutaan edelleen getAlerts- ja getBundles–funktioita. ExportAlert–funktio puolestaan hoitaa uuden nipun luonnin ja hälytyksen siirtämisen jo luotuun nippuun. CreateReport–funktio tallentaa nipun tilan tietokantaan myöhempää tarkastelua varten.



Kuva 2. MVC–mallin mukainen sekvenssikaavio funktioista.

Sovelluksen toteutus alkoi hälytysdatan esittämisen ja niputuksen toiminnallisuuden rakentamisella. Hälytysdatan esittäminen tehtiin taulukoilla, joihin hälytykset lisätään silmukoita käyttäen. Jokaiselle hälytysriville täytyy saada myös toiminnallisuuksia kyseistä hälytystä varten. Nippudatan esitys vaati huomattavasti enemmän toiminnallisuuksia kuin hälytysdatan. Niput pitää pystyä avaamaan ja sulkemaan sekä hälytyksiä poistamaan nipusta tai siirtämään toiseen nippuun. Nippuihin pitää luoda myös painike, jolla nipun tila tallennetaan tietokantaan. Toteutusvaiheen suurin osuus oli itse hälytysnäkymän rakentaminen ja siihen liittyvät toiminnallisuudet, mutta myös rajapintojen ja tietokannan toteutus vei aikaa. Rajapinnoissa täytyy tehdä ensin sovelluksenpuoleinen toteutus ja sen jälkeen

vastaava toteutus palvelimen päähän. Monet toiminnallisuudet vaativat tietokantakyselyitä useasta eri tietokantataulusta. Esimerkkinä toiminnallisuus kun luodaan hälytykselle uusi nippu: Käyttäjä valitsee hälytysnäkymän hälytyksestä uusi nippu. Tämän jälkeen välitetään parametrinä hälytyksen id funktiolle joka puolestaan avaa ruudulle tallennusikkunan. Tallennusikkuna näyttää saadun id:n perusteella hälytyksen tiedot ja käyttäjä voi syöttää nipun nimen ja kuvauksen. Kun käyttäjä painaa tallenna-painiketta, välitetään annetut tiedot post- metodilla serverin Rest-rajapintaan. Rajapinta kutsuu tietokannan omaa rajapintaa ja tietojen tallennus MySQL–tauluihin alkaa. Ensimmäisenä tallennetaan tiedot nipusta omaan tauluun, sitten haetaan juuri luodun rivin id nippu-taulusta ja haetaan hälytyksen tiedot hälytys-taulusta. Viimeisenä tallennetaan hälytys omaan, nipussa oleville hälytyksille tarkoitettuun tauluun. Nipussa oleville hälytyksille tarkoitettuun tauluun tallennetaan hälytyksen tiedot sekä tieto mihin nippuun hälytys kuuluu.

4 ANVIAN REAGOINTIJÄRJESTELMÄN KUVAUS

4.1 Toteutusympäristö

Anvian reagointijärjestelmää lähdettiin toteuttamaan jo olemassa olevaan Triton-ympäristöön. Triton on Anvia Oyj:n oma järjestelmä, johon on koottu Anvian sähköiset työkalut. Anvian reagointijärjestelmän kehitysvaiheessa käytettiin testiympäristöä mm. koodin ja tietokantojen osalta mikä helpotti kehitystyötä. Kehitysympäristössä oli käytössä joitain valmiita tyylimäärittelyjä ja ponnahdusikkunoita joita pystyi hyödyntämään sovelluksen toteutuksessa. Tosin valmiiden toiminnallisuuksien hyödyntäminen tuntui välillä vaikeammalta kuin omien tekeminen. Toteutusympäristössä joitain rajapintoja oli valmiina, mutta osa toteutettiin tässä työssä.

4.2 Hälytysnäkymä

Hälytysnäkymälle eli sovelluksen päänäkymälle tulostuu kaikki Anvian reagointijärjestelmään saapuvat hälytykset. Hälytysten taustaväri määräytyy hälytyksen kriittisyyden ja tilan mukaan. Hälytyksissä on myös seuraavat toiminnallisuudet:

- Hälytyksen voi merkitä huomioiduksi
    - Kun hälytys merkitään huomioiduksi, siitä lähtee tieto muihin järjestelmiin sekä hälytyksen taustaväri muuttuu Anvian reagointijärjestelmässä.
- Hälytyksen voi siirtää nippuun
    - Hälytystä siirrettäessä nippuun, luodaan uusi nippu tai siirretään hälytys jo olemassa olevaan nippuun. Kun hälytys on siirretty nippuun, poistuu se Anvian reagointijärjestelmän hälytysnäkymältä ja tulee näkymään haluttuun nippuun.
- Hälytyksestä voi luoda ilmoituksen vikalokiin
    - Vikalokiin luotu hälytys tulee näkyviin vikalokissa. Vikalokista asiakaspalvelu yms. Henkilökunta voi seurata vikailmoituksia.

- Hälytyksestä voi luoda netAdmin -tiketin
    - NetAdmin–tiketillä välitetään tietoa vian korjaajalle, tikettiin tallentuu keskeiset tiedot hälytyksestä sekä hälytyksen aiheuttaneesta laitteesta.

Hälytysnäkymän hälytykset järjestyy näkymään niiden kriittisyyden ja tilan mukaan. Kun hälytyksen aiheuttaja on korjattu/poistettu, muuttuu hälytys vihreäksi ja sen jälkeen poistuu. Osa hälytyksistä vaatii välittömiä toimenpiteitä, joten on hyvä että nämä hälytykset on koottu näkymän samaan kohtaan ja keskeiselle paikalle.

4.3 Niput

Hälytykselle voidaan luoda nippu, jonka jälkeen samaan nippuun voidaan siirtää muitakin hälytyksiä. Koska usein yhdestä viasta tulee useita hälytyksiä, helpottaa niputus näiden vikojen seuraamista. Hälytysnäkymän yläosassa näkyy ne niput jotka on aktiivisia ja joissa on hälytyksiä. Nipussa olevissa hälytyksissä on samat toiminnallisuudet kuin niissäkin hälytyksissä jotka eivät ole nipussa: muuta tilaa, siirrä nippuun, luo ilmoitus vikalokiin ja luo netAdmin–tiketti. Lisäksi nipun hälytyksissä on toiminnallisuudet siirtää hälytys toiseen nippuun ja poistaa hälytys nipusta. Nipussa olevan hälytyksen taustaväri määräytyy samalla tavalla kuin niidenkin jotka eivät ole nipussa, kriittisyyden ja tilan mukaan. Kun hälytyksen aiheuttanut vika on korjattu, muuttuu nipussa oleva hälytys vihreäksi, mutta ei poistu nipusta. Tämän lisäksi jokaisessa nipussa on seuraavat toiminnallisuudet:

- Nipun sulkeminen
    - Kun nippu suljetaan, poistuu se hälytysnäkymältä eikä siihen voida enää siirtää uusia hälytyksiä.

- Nipun pienentäminen/suurentaminen
    - Pienennä/suurenna–painikkeesta nipun tila muuttuu. Kun nippu on pienennetty, siitä näkyy vain nimi sekä sulje- suurenna- ja tallenna-painikkeet.
- Nipun tilan tallennus
    - Kun nipun tilan tallennus–painiketta painaa, tallentuu nipun tiedot ja tiedot sen hetkisistä hälytyksistä. Myös tieto siitä kuka tallennuksen teki tallennetaan.

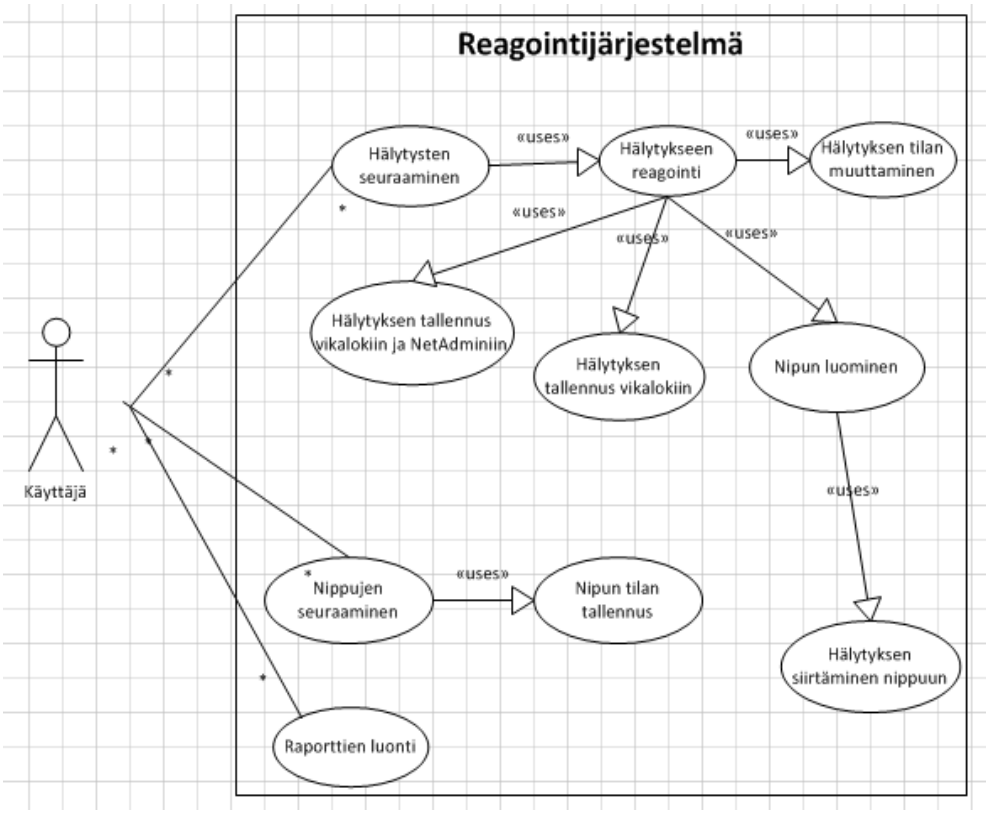
Hälytysnäkymällä on lisäksi kello, josta näkee kellonajan lisäksi että näkymä on toiminnassa.

4.4 Raportointi

Yksi keskeinen vaatimus Anvian reagointijärjestelmää lähdettäessä toteuttamaan oli raportointi. Se että hälytyksiä pystyy seuraamaan reaaliaikaisesti ja niihin reagoimaan ei riitä, vaan hälytyksistä ja vikatilanteista pitää pystyä raportoimaan jälkikäteen. Anvian reagointijärjestelmässä pystyy luomaan raportteja luoduista nipuista ja niiden tallennetuista tiloista. Aina kun uusi nippu luodaan, tallentuu siitä tieto tietokantaan raporttia varten. Nipun luontitilanteessa (**Kuva 3)** tallentuu nipun tiedot sekä tieto hälytyksestä joka sinne viedään. Tieto nipussa olevista hälytyksistä tallentuu myös aina kun painetaan nipun tallenna–painiketta. Näin saadaan tarkkoja raportteja hälytystiedoista ja raportteja pystytään tulostamaan järjestelmästä myöhemmin.

4.5 Näkymän päivitys

Anvian reagointijärjestelmän näkymän päivitys voidaan määritellä tarpeen mukaan. Myös aina muutoksen jälkeen se näkymän osa jota muutos koskee päivitetään. Näkymän päivityksessä käytetään AJAX–tekniikkaa, joka tekee näkymän päivityksestä joustavan ja huomaamattoman.



Kuva 3. Käyttötapakaavio Anvian reagointijärjestelmästä.

5 TOTEUTUS

5.1 Ajastettu asynkroninen päivitys

Anvian reagointijärjestelmän toiminnallisuudet toteutettiin PHP- ja JavaScript–ohjelmointikielillä. JavaScriptiä käytettiin lähinnä selaimen ohjaamiseen ja funktioiden toteuttamiseen. PHP:llä puolestaan toteutettiin osa tiedon esittämisestä sekä tiedon välittäminen rajapinnalle ja tiedon vastaanottaminen rajapinnalta. Kuvassa 4 esimerkki koodista jossa tieto välitetään JavaScript–funktiosta PHP–tiedostoon.

```
1  <script type="text/javascript">
2
3  function insertText() {
4  $.post('/PHP/data/getdata.php',
5      function(data) {
6      var textToInsert = data;
7      $("#div_data").html(textToInsert);
8      });
9      });
10 </script>
11
12 <div id="div_data"></div>
```

Kuva 4. Esimerkki JavaScript -funktiosta.

Kuvan 4 JavaScript–koodissa on insertText–funktio jota voidaan kutsua eri tilanteissa. Rivillä neljä post–metodilla kutsutaan getdata.php–tiedostoa. Getdata.php paluuarvo tallennetaan data–muuttujaan ja data puolestaan tallennetaan textToInsert–muuttujaan. Rivillä seitsemän textToInsert–muuttuja syötetään tagiin, jonka id on div_data. Aina kun insertText–funktiota kutsutaan, päivittää se div–tagin sisällön. Anvian reagointijärjestelmässä sivun päivitys tapahtuu vastaavalla funktiolla. Funktiota kutsutaan ajastetusti, jolloin näkymä päivittyy säännöllisesti. Funktiota kutsutaan myös sen jälkeen kun käyttäjä tekee muutoksia, jos käyttäjä esimerkiksi luo uuden nipun, funktiota kutsutaan ja näkymä päivittyy. Anvian reagointijärjestelmässä toteutettiin useita funktiota joilla näkymää voidaan päivit-

tää. Jos käytössä olisi vain yksi funktio, koko näkymä jouduttaisiin päivittämään, vaikka muutos koskisi vain osaa näkymästä.

Sivun ajastettu päivitys tapahtuu startTime–funktion avulla. StartTime–funktiota kutsutaan sekunnin välein, jolloin samalla funktiolla voidaan päivittää päänäkymän kellonaika.

```
1  function startTime()
2  {
3  var today = new Date();
4  var h = today.getHours();
5  var m = today.getMinutes();
6  var s = today.getSeconds();
7
8  if (s == 60) {
9      getAlerts();
10     getBundles();
11     setTimeout(function(){loadjQuery()},500);
12 }
13 m = checkTime(m);
14 s = checkTime(s);
15
16 document.getElementById('txt').innerHTML = h+":"+m+":"+s;
17 t = setTimeout(function() {startTime()},1000);
18 }
19
20 function checkTime(i)
21 {
22 if (i<10)
23     {
24     i = "0" + i;
25     }
26     return i;
27 }
```

Kuva 5. Anvian reagointijärjestelmän startTime–funktio.

Kuvassa 5 startTime–funktio riveillä 1-18.Riveillä 3-6 luodaan Date–objekti ja kutsutaan Date–objektin metodeita getHours, getMinutes ja getSeconds. Samalla kun metodeita kutsutaan, luodaan muuttujat ja sijoitetaan saadut tunnit, minuutit ja sekunnit muuttujiin. Riveillä 8-12 olevalla if–lauseella hoidetaan sivun ajastettu

päivitys. If -lause suoritetaan aina kun s–muuttuja on saanut arvon 60, eli minuutin välein. If–lauseen sisällä olevilla funktiokutsuilla päivitetään Anvian reagointijärjestelmän näkymä sekä JavaScript toiminnot jotka käyttävät jQuery–kirjastoa. GetAlerts–funktio päivittää hälytysnäkymän hälytykset, getBundles–funktio päivittää niput ja niissä olevat hälytykset sekä loadjQuery–funktio jQuery–toiminnallisuudet. Rivin 11 loadjQuery–funktio suoritaan puolen sekunnin viiveellä. Viive vaaditaan jotta kaikki hälytysdata on ehditty päivittää ennen jQuery-päivitystä. Rivillä 17 kutsutaan startTime–funktiota aina ajastetusti sekunnin välein. Edellä luetelluilla toiminnallisuuksilla pystytään siis ajastetusti päivittämään JavaScriptin avulla toteutettu web-sivu.

Kuvan 5 toinen funktio, checkTime, lisää nollan tarvittaessa sekunti- tai minuuttiluvun eteen. Tämä toiminnallisuus tarvitaan ajan esittämistä varten. Funtion sisällä rivillä 22 olevalla if–lauseella verrataan onko sekunti tai minuutti luku alle 10. Riveillä 13 ja 14 on checkTime–funktion kutsut. Rivillä 16 päivitetään html–elementti, jonka id on txt. Elementti saa arvon h+":"+m+":"+s, eli tunnit, minuutit ja sekunnit päivittyvät näkymään.

StartTime-funktiota kutsutaan kuvan 5 koodissa ainoastaan funktion sisällä, joten startTime pitää kutsua erikseen sivua ladattaessa ensimmäisen kerran. Myös ajan esittämää txt–elementtiä ei kuvassa 5 ole esitelty.

```
<div width='100%' align='right' style='color:red' id='txt'><body onload='startTime()'></div>
```

Kuva 6.

Kuvassa 6 määritelty div–elementti jota checkTime–funktio päivittää. Elementin tunnistus tapahtuu id:n avulla. Onload–määrityksellä kutsutaan startTime–funktiota aina kun sivu web-sivu ladataan uudestaan. Kun kuvan 6 div–elementissä ollaan kerran kutsuttu startTime–funktiota ja saatu elementissä näytettävä kellonaika, hoitaa startTime–funktion sisäinen kutsu ajan päivityksen siitä eteenpäin.

Kuvan 4 getdata.php–tiedostossa kutsutaan tietokannan rajapintaa ja kutsun paluuarvona saadaan haluttu data. Getdata.php–tiedostossa voidaan myös muokata esitettävä data haluttuun muotoon. Kuvassa 7 esimerkki getdata.php–tiedoston rajapintakutsu, sekä vastaanotetun datan sijoittaminen uuteen luokkaan.

```
1  $InnerText = $database->getInnerText();
2  foreach ($InnerText->answer as $dataRow) {
3
4      $text = new stdClass;
5      $text->id           = html_entity_decode($dataRow->id);
6      $text->startTime    = html_entity_decode($dataRow->startTime);
7      $text->endTime      = html_entity_decode($dataRow->endTime);
8      $text->priority     = html_entity_decode($dataRow->priority);
9      $text->equipment    = html_entity_decode($dataRow->equipment);
10     }
```

Kuva 7. Getdata.php–tiedoston datan haku tietokannasta.

Kuvassa 7 rivillä yksi rajapintakutsu, jolla data haetaan ja haettu data tallennetaan muuttujaan $InnerText. Riviltä kaksi alkaen foreach–silmukassa vastaanotettu data tallennetaan uuteen, rivillä neljä luotuun $text–luokkaan. Vastaanotetulle datalle tehdään myös merkistökoodaus, jotta mahdolliset erikoismerkit näkyvät oikein.

```
1  $criticalAlerts;
2  $majorAlerts;
3  $warningAlerts;
4
5  if(!empty($text->endTime)) {
6      if($text->priority) == 'critical') {
7
8      $select = '<select name="status">
9                      <option value=""></option>
10                     <option value="Acked(Ei ilm)">Acked(Ei ilm)</option>
11                     <option value="Acked">Acked</option>
12                     <option value="New">New</option>
13                     <option value="Cancelled">Cancelled</option>
14                 </select>';
15
16     $criticalAlerts = '<tr align = "center">
17                         <td>'.$select.'</td>
18                         <td>'.$text->id.'</td>
19                         <td>'.$text->startTime.'</td>
20                         <td>'.$text->endTime.'</td>
21                         <td>'.$text->priority.'</td>
22                         <td>'.$text->equipment.'</td>
23                         <td><a href = "window/popup.php"></a></td>
24                         ';
25     }
26 }
```

Kuva 8. Datan tallennus esitettävään muotoon getdata.php–tiedostossa.

Kun kuvassa 7 haettiin data tietokannasta ja tallennettiin se $text–luokkaan, kuvassa 8 $text–luokan data muokataan esitettävään muotoon. Riveillä yksi-kolme luodaan muuttujat critical-, major- ja warning-hälytyksille. Rivin viisi if–ehdolla tarkastetaan että endTime on tyhjä. Kun endTime on tyhjä, hälytys on silloin vielä aktiivinen, eikä sille ole viety loppupäivämäärää. Rivin kuusi if -ehdolla tarkastetaan onko hälytyksen prioriteetti critical, näin saadaan valittua pelkästään kriittiset hälytykset. Riveillä 8-14 tallennetaan muuttujaan $select select–elementti. Itse hälytyksen tiedot tallennetaan riveillä 16-24 $criticalAlerts-muuttujaan. Hälytystietojen lisäksi $criticalAlerts–muuttujaan tallentuu taulukon tr ja td elementit, koska tieto halutaan esittää taulukossa. Rivillä 17 olevassa taulukon solussa esitetään $select–muuttujan sisältä, eli select–elementti. Select–elementistä voi valita eri tiloja hälytykselle ja muutos koskee vain sitä hälytystä josta select- elementin

arvo muutetaan. Tietokannasta haettua sisältöä tallennetaan rivien 18-22 soluihin ja rivin 23 soluun tallennetaan linkki josta aukeaa popup–ikkuna.

Kuvissa 7. ja 8. näkyy siis rajapintakutsu sisällön hakemiseen sekä sisällön esittämistä taulukossa. Kuvassa 8 lisätään $criticalAlerts–muuttujaan sisältö, ja samalla tavalla pystytään lisäämään myös muuttujien $majorAlerts ja $warningAlerts sisältö. Toistetaan vain rivin kuusi if–lause ja muutetaan toteutumisehdoksi major tai warning. Kun myös major- ja warning-hälytykset on tallennettu muuttujiin, päästään tiedon lopulliseen esittämiseen. Koska hälytykset halutaan esittää taulukossa, vaatii lopullinen sisällön esittäminen hälytystietojen lisäksi myös taulukon rakenteen. Osa taulukon takenteesta on valmiiksi tallennettuna $criticalAlerts, $majorAlerts ja $warningAlerts–muuttujiin. Taulukon osittainen tallentaminen hälytysdatan mukana muuttujiin oli pakollista, koska muuten hälytysdatan esittäminen taulukossa ei onnistuisi. Tietokannasta haettuna hälytysdata joudutaan purkamaan hälytys kerrallaan foreach–silmukassa ja tallentamaan yksi hälytys per silmukan kierros. Paras tapa järjestää hälytykset taulukkoon on luoda jokaisella foreach–silmukan kierroksella taulukon rivi tr–tagilla ja tallentaa hälytysdata riville. Tietokannasta haettu data saadaan näin samaan muotoon kuin se on tietokannassakin, eli riveittäin järjestettynä.

Lopuksi kun kuvan 7 foreach–silmukka on käynyt kaikki rivit läpi ja kaikki hälytysdata on tallennettu muuttujiin, voidaan aloittaa tiedon lopullinen esittäminen.

```
1  echo '<table border = "1" width = "100%" style = "border-collapse;border-color:black";>
2          <thead style = "background-color:LightSlateGray;color:White;">
3              <tr>
4                  <th>ID</th>
5                  <th>Start Time</th>
6                  <th>End Time</th>
7                  <th>Priority</th>
8                  <th>Equipment</th>
9              </tr>
10         </thead>
11         <tbody>
12             '.$criticalAlerts.'
13             '.$majorAlerts.'
14             '.$warningAlerts.'
15         </tbody>
16       </table>';
```

Kuva 9. Tiedon lopullinen esittäminen getdata.php–tiedostossa.

Sisällön lopullinen esittäminen tapahtuu PHP–kielen echo–komennolla. Echo–komennon syntaksi on*: echo ’’;* . Heittomerkkien väliin lisätty sisältö palautuu kuvassa 4 rivillä viisi olevaan data–muuttujaan paluuarvona. Kun esitettävää sisältöä on suuri määrä, on hyvä käyttää apumuuttujia tiedon lopullisessa esityksessä. Apumuuttujien käyttö selkeyttää koodin lukemista ja helpottaa näin ollen jälkeenpäin tehtäviä muutoksia ja mahdollista vian etsintää. Tässä tapauksessa käytettävät apumuuttujat $criticalAlerts, $majorAlerts ja $warningAlerts helpottavat koodin lukemista, mutta niiden käyttöön oli toinenkin syy. Koska näytettävät hälytykset halutaan esittää niiden prioriteetin mukaan, täytyy hälytysdata tallentaa erillisiin muuttujiin. Kuvassa 9 esitettävä sisältö lopullisessa muodossa. Riveillä 2-10 taulukon otsikkokentät, joita ei ole vielä määritelty apumuuttujissa. Riveillä 12-14 apumuuttujat, jotka sisältää taulukon hälytysrivit. Apumuuttujien ympärille on lisätty heittomerkit ja pisteet, tämä on PHP–kielen syntaksia ja niillä erotetaan merkkijono muuttujasta. Koska taulukon apumuuttujiin lisättiin sisältö foreach–silmukassa, on taulukon koko dynaaminen. Kun hälytysten määrä vaihtelee koko ajan, on myös taulukon koon vaihdeltava. Apumuuttujat $criticalAlerts, $majorAlerts ja $warningAlerts sisältävät jokainen hälytysdatan lisäksi taulukon tr ja td–tagit, jotka myös kuvassa 7 näkyvät.

5.2 Hälytysnippujen esitys

Hälytysnippujen tiedot ja nipuissa olevien hälytysten esittäminen hälytysnäkymällä tapahtuu asynkronisesti ja ajastetusti, kuten aiemmassa kappaleessa kerrottiin. Nippujen ja niissä olevien hälytysten esittämisen lisäksi pitää olla nippujen lisätoiminnot: nipun suurennus/pienennys, nipun sulkeminen, nipun tilan tallennus sekä hälytyksen poistaminen nipusta. Tieto siitä onko nippu suurennettu vai pienennetty tai tallennetaan tietokantaan ja näin ollen nipun tila päivittyy aina kun näkymäkin päivittyy. Nippujen hälytykset esitetään taulukossa ja nipun suurennus/pienennys–toiminnallisuus toteutettiin taulukon tyyliasetuksella.

```
$bundleHeader.=
'<table style='display:".$showAlert."'id='".$bundledata->id."'border='1'width='100%';>
```

Kuva 10. Nipuissa olevien hälytysten piilotus.

Kuvassa 10 $bundleHeader–muuttujaan lisätään taulukon aloitustagi jossa $showAlert–muuttuja määrittelee taulukon näkyvyyden. Jokaisen nipun kohdalla on painike, jolla voidaan pienentää tai suurentaa nippu. Painikkeessa on toiminnallisuus joka tarkistaa tietokannasta kyseisen nipun nykyisen tilan ja vaihtaa tilaa. Taulukon tyylimäärittely display -arvolla none nippu on pienennetty ja arvolla table suurennettu.

```
1  $bundleAlerts = $database->getBundleAlerts();
2  $bundleMetaData = $database->getBundleMetaData();
3
4  foreach ($bundleMetaData->answer as $bundleRow) {
5
6      if($bundleRow->minimized != true) {
7          $stateButton = 'Suurenna';
8      }else {
9          $stateButton = utf8_encode('Pienennä');
10     }
11
12     if($statusButton == 'Suurenna') {
13         $minimized = true;
14     } else{
15         $minimized = false;
16     }
```

Kuva 11. getBundleData.php–tiedoston rajapintakutsut ja nipun tilan muuttaminen.

Nippujen ja niissä olevien hälytysten esittämiseen vaaditaan hälytysdatan lisäksi myös tiedot nipuista. Kuvassa 11 rivillä kaksi tehdään rajapintakutsu, jolla haetaan tiedot nipuista. Rivin neljä foreach–silmukalla käydään läpi jokainen haettu nippu erikseen. Riveillä 6-10 verrataan nipun minimized–muuttujaa, jolla on boolean–tyypinen arvo, joko true tai false. Koska Anvian reagointijärjestelmässä nipun suurentaminen ja pienentäminen on toteutettu samaan painikkeeseen, täytyy painikkeen teksti muuttua nipun tilan mukaan. Kun nippu on pienennetty painikkeessa lukee suurenna ja kun nippu on suurennettu painikkeessa lukee pienennä. Kuvassa 11 rivillä kuusi oleva if–else–lause vaihtaa $stateButton–muuttujan tilaa sen mukaan onko nippu pienennetty vai suurennettu. Rivin 12 if–else–lauseella puolestaan määritellään $minimized–muuttujan arvo ja se välitetään rajapinnan kautta tietokantaan. Rivin neljä foreach–silmukka jatkuu vielä kuvassa näkyvän if–else–lausekkeen jälkeenkin. Silmukassa mm. tallennetaan näytettävät tiedot nipusta ja nipun hälytyksistä. Kuvassa 12 on rakenne jolla esitetään nippujen tiedot ja oikeat hälytykset kunkin nipun alla.

Anvian reagointijärjestelmän näkymässä on kymmeniä hälytyksiä ja useita nippuja samanaikaisesti. Jokaisessa hälytyksessä ja nipussa on omat toiminnallisuudet, joilla hallitaan kyseistä hälytystä tai nippua. Ongelmaksi nousee, miten tunnistaa hälytys tai nippu jota halutaan toiminnallisuuden koskevan. Helpoin tapa painikkeen ja sitä koskevan hälytyksen tunnistamiseen on id–numero.

```
1  foreach ($bundleMetaData->answer as $bundleRow) {
2
3      $closeButton = "
4      <button type='button' id='".$bundleRow->id."'
5      name='".$stateButton."' onclick='closeBundle(this)'>Sulje</button>
6      ";
7      $stateChangeButton = "
8      <button type='button' id='".$bundleRow->id."'
9      name='".$stateButton."' onclick='changeBundleState(this)'>".$stateButton."</button>
10     ";
11     $saveBundleButton = "
12     <button type='button' id='".$bundleRow->id."'
13     name='".$stateButton."' onclick='saveBundleState(this)'>Tallenna</button>
14     ";
15     $aElement = "
16     <a style='font-family:Times new roman;color:White;font-size:12px;' id='".$bundleRow->id."'></a>
17     ";
```

Kuva 12. getBundleData.php–tiedoston painikkeet

Kun haetaan tietokannasta useita taulun rivejä joilla on yksilöllinen id-numero, voidaan samoja id-numeroita käyttää myös tagien id-numeroina. Kuvassa 12 rivillä yksi foreach–silmukka jolla käydään läpi $bundleMetaData–objektin sisältö. Riveillä kolme-kuusi $closeButton–muuttujan tallennus, erillinen muuttuja on helppo lisätä muualle koodiin esittämistä varten. Riveillä 7-10 tallennetaan muuttujaan $stateChangeButton -painike jolla nipun tilan voi muuttaa ja riveillä 11-14 $saveBundleButton–painike jolla nipun tila tallennetaan. Rivien 15-17 $aElement–muuttuja on tiedon esittämistä varten. Näihin neljään muuttujaan tallennetaan siis jokaisella foreach–silmukan kierroksella kolme painiketta ja a–tagi. Jokaisella foreach–silmukan kierroksella tallennetaan eri nipun tiedot ja tallennettavat tagit saa jokaisella kerralla yksilöllisen id–numeron. Yksilöllisten id–numerojen perusteella pystytään erottamaan jokainen nippu ja nipun painike toisistaan. Jos esimerkiksi painetaan $closeButton–muuttujaan tallennettua painiketta, kutsuu se closeBundle–funktiota. Funktiolle välitetään parametreinä kaikki button–tagin tiedot. CloseBundle–funktiossa voidaan helposti toteuttaa toiminto

jolla nippu suljetaan yksilöllisen id–numeron perusteella. Vastaavalla tavalla voidaan a -tagin sisältö päivittää yksilöllisen id:n perusteella.

Kun hälytyksiä siirretään nippuihin, tallentuu tieto nippuun siirretystä hälytyksestä uuteen tietokantatauluun. Jos hälytys on siirretty nippuun, ei sitä haluta näyttää enää hälytysnäkymän päänäkymällä, vaan pelkästään siinä nipussa johon se on siirretty. Hälytys ei kuitenkaan poistu alkuperäisestä tietokantataulusta nippuun siirrettäessä vaan se pelkästään kopioidaan uuteen tauluun. Ohjelmaan täytyi siis rakentaa toiminto joka tarkistaa onko hälytys nipussa ja jos on, sitä ei näytetä varsinaisessa hälytysnäkymässä. Anvian reagointijärjestelmässä saattaa olla useita eri nippuja luotuna ja nipuissa useita eri hälytyksiä. Hälytysten yhdistäminen oikeisiin nippuihin vaatii myös oman toiminnallisuuden.

```
1  foreach ($bundleMetaData->answer as $bundleRow) {
2      foreach ($bundleAlertData->answer as $bundleRow) {
3          if($alert->bundle_id == $bundleRow->id) {
4
5              //Tässä tallennetaan jokaisen nipun hälytykset
6          }
7      }
8  }
```

Kuva 13. Nippujen ja hälytysten yhdistäminen.

Kuvassa 13 kaksi sisäkkäistä foreach–silmukkaa, joilla käydään läpi kahden eri tietokantataulun sisältö. Tietokantataulujen yhdistävää tekijää, eli nipun id–numeroa verrataan rivillä seitsemän. BundleMetaData–objektissa on reagx_alerts–taulun sisältö ja taulun id–numero on nipun yksilöllinen id–numero. Bundlealertdata–objektissa on reagx_bundle_alerts–taulun sisältö ja bundle_id–kentässä on id–numero josta tunnistetaan mihin nippuun hälytys kuuluu. Kun suoritetaan kaksi sisäkkäistä silmukkaa ja verrataan id–numeroita keskenään, saadaan reagx_alerts–taulusta ja reagx_bundle_alerts–taulusta oikeat rivit.

5.3 Nipun tilan tallennus

Aina kun hälytysnipussa tapahtuu muutos, sen tila tallennetaan. Tallennus tapahtuu siis kun nippu luodaan, siihen lisätään tai siitä poistetaan hälytys, nipussa olevien hälytysten tila muuttuu tai kun nippu suljetaan. Anvian reagointijärjestelmään haluttiin myös erillinen painike, josta tila voidaan tallentaa. Jokaisen nipun kohdalle toteutettiin myös kenttä, jossa näytetään ilmoitus milloin nipun tila on tallennettu. Kuvassa 12 nipun tilan tallennuspainike tallennetaan muuttujaan riveillä 11-14 ja a–tagi jossa voidaan esittää nipun tallennusaika, tallennetaan riveillä 15-17.

```
1  function saveBundleState(formdata) {
2      var bundleid = formdata.id;
3      var loggeduser = '<?php= $login?>';
4      $.post('/PHP/data/saveBundleState.php', {bundleid: bundeid, loggeduser: loggeduser},
5          function(data) {
6              if(data == 1) {
7                  var time = document.getElementById("txt").innerHTML;
8                  var i = 0;
9                      function state() {
10                         $("#savebundle").html("Tallennettu: "+time);
11                         i++;
12                             if(i<50) {
13                                 setTimeout(state,100);
14                             };
15                     };
16                     state();
17                     setTimeout(function() {$("#savebundle").html("")},6000);
18             }else {
19                 alert('Tallennus ei onnistunut');
20             }
21         });
22 };
```

Kuva 14. JavaScript -funktio jolla nipun tila tallentuu.

Kuvassa 14 saveBundleState–funktio jolla tallennetaan nipun tila, sekä tulostetaan näytölle tieto onnistuiko tallennus. Rivillä kaksi tallennetaan bundleid–muuttujaan parametrinä saatu nipun id ja rivillä kolme loggeduser–muuttujaan kirjautunut käyttäjä. Kirjautunut käyttäjä on tallennettu PHP–muuttujaan $login, joten muuttuja pitää olla <? ?> -tagien sisällä. Rivillä neljä post–metodilla välitetään parametrit bundleid ja loggeduser saveBundleState.php–tiedostoon. Paluuarvon post–metodi palauttaa rivin viisi data–muuttujaan. Jos tallennus onnistui, paluuarvo on yksi ja rivin kuusi if–ehto saa arvon true. Rivillä seitsemän time–muuttujaan tal-

lennetaan kellonaika. Kellonaika saadaan Anvian reagointijärjestelmän päänäkymän kellosta. Haetaan siis elementin sisältö jonka id on txt ja senhetkinen kellonaika tallentuu. Kun funktiolla on tieto siitä onko tallennus onnistunut ja tarvittavat parametrit, voidaan aloittaa tiedon esittäminen. Ilmoitus nipun tilan tallennuksen onnistumisesta haluttiin näkyvän muutaman sekunnin ajan ja sen jälkeen ilmoituksen täytyisi hävitä näkyvistä. JavaScriptissä on tällaisen toiminnallisuuden toteuttamiseen setTimeOut–funktio, jolla voidaan asettaa aikaviiveitä. Anvian reagointijärjestelmän asynkroninen hälytysnäkymän päivitys asetti omat haasteensa setTimeOut–funktion käytölle. Nippujen asynkroninen päivitys tyhjää nipuissa olevien tagien sisällön, eikä näkynä toimi kuten haluttiin. Ratkaisu ongelmaan on kuvassa 14 rivillä yhdeksän oleva state–funktio. State–funktion sisällä päivitetään 50 kertaa 100 millisekunnin viiveellä nipussa olevaa tagia, jolloin tagin sisältö pysyy myös mahdollisen näkymäpäivityksen jälkeenkin. State–funktion jälkeen rivillä 17 kuuden sekunnin viiveellä asetetaan tagi tyhjäksi, eli ilmoitus poistuu kuuden sekunnin jälkeen näkyvistä. Rivillä 19 popup–ikkuna näytetään, jos tallennus ei onnistunut.

```
1  $class = new Reagointi();
2
3  $bundle = new Datamodel_Bundle();
4      $bundle->id = $_POST['bundleid'];
5      $bundle->creator = $_POST['loggeduser'];
6
7      try {
8          $return = $class->saveBundleState($bundle);
9      } catch (Exception $e) {
10         die();
11     }
12     echo $return->response;
```

Kuva 15. SaveBundleState.php–tiedosto.

Kuvassa 15 PHP–toiminnallisuus, jota kutsutaan kun nipun tila tallennetaan. Tätä toiminnallisuutta kutsutaan kuvassa 14 ja välitetään bundleid ja loggeduser parametrit. Rivillä yksi luodaan ilmentymä Reagointi–luokasta ja rivillä kolme luokas-

ta Datamodel_Bundle. Riveillä neljä ja viisi otetaan vastaan parametrit, jotka post–metodin mukana välitettiin. Parametrit tallennetaan $bundle–olioon ja olio välitetään rajapintaan parametrinä. Riveillä 7-11 try-catch–lauseella tarkistetaan tapahtuiko tallennuksessa jokin virhe. Mikäli kaikki halutut tiedot onnistutaan tallentamaan tietokantaan, palauttaa rajapinta arvon true eli yksi. Saatu paluuarvo tallennetaan $return–objektiin ja rivillä 12 echo–komennolla objektiin tallennettu response–arvo palautuu kutsuvaan post–metodiin.

5.4 Hälytysten ryhmittely ja taustavärit

Anvian reagointijärjestelmässä toteutettiin hälytysten ryhmittely kahden eri muuttujan mukaan, hälytyksen kriittisyyden ja sen mukaan onko hälytykseen reagoitu. Ensin hälytykset ryhmitellään tärkeimmän eli kriittisyyden mukaan. Hälytykset on jaettu neljään eri ryhmään niiden kriittisyyden perusteella, ja jokaisessa ryhmässä hälytyksillä on eri taustaväri. Kun hälytykset on ryhmitelty hälytysnäkymälle kriittisyyden perusteella ja lisäksi jokaisen ryhmän hälytyksillä on erilainen taustaväri, helpottaa se hälytysnäkymän seuraamista. Hälytysten taustaväri muuttuu myös sen mukaan onko hälytykseen reagoitu. Hälytyksillä on siis useita eri taustavärejä, jotka pitää muuttua aina hälytyksen kriittisyyden ja tilan mukaan.

```
1  if(empty($alert->notedWho)){
2      switch($alert->priority){
3          case 'Critical':
4              $class = 'uusiCritical';
5              break;
6          case 'Major':
7              $class = 'uusiMajor';
8              break;
9      }
10 }else {
11     switch($alert->priority) {
12         case 'Critical':
13             $class = 'huomioituCritical';
14             break;
15         case 'Major':
16             $class = 'huomioituMajor';
17             break;
18     }
19 }
20 if($alert->status == 'Cancelled') {
21     $class = 'Cancelled';
22 }
```

Kuva 16. Hälytysten taustavärin määrittäminen.

Kuvassa 16 määritellään hälytysten taustavärit. Ensimmäisenä rivillä yksi tarkastetaan onko $alert–olion notedWho–muuttujaa määritelty. Jos notedWho–muuttujaa ei ole määritelty tarkoittaa se että hälytykseen ei ole vielä kukaan reagoinut. Riveillä kaksi-yhdeksän määritellään hälytysten taustavärit sellaisille hälytyksille joihin ei ole kukaan vielä reagoinut. Switch–case–lauseella määritellään $class–muuttujan arvo. $class–muuttujaan taas käytetään esitettävien hälytysrivien tyylimäärittelyssä. UusiCritical ja uusiMajor–tyylit on puolestaan määritelty erillisessä tiedostossa ja niissä myös värit on määritelty. Mikäli rivin yksi if–ehto ei toteudu, suoritetaan rivien 10-19 else–lause. Else–lauseessa määritellään taustavärit niille hälytyksille joihin on reagoitu. Kun hälytys on poistumassa, saa status arvon Cancelled. Poistumassa olevat hälytykset näkyvät Anvian reagointijärjestelmän näkymässä vihreinä, jotta nähdään hälytyksen aiheuttaneen vian poistu-

neen. Rivien 20-22 if–lauseella tarkistetaan onko hälytyksen status asetettu Cancelled–tilaan.

5.5 Raportin tulostus

Anvian reagointijärjestelmässä on toiminnallisuus nippujen tallentamiseen. Toiminnallisuus on automaattinen ja tallennus tapahtuu aina kun nipun tilassa tapahtuu muutos. Myös manuaalinen tallennus on mahdollista, jolla käyttäjä voi tallentaa nipun tilan koska itse haluaa. Nippujen tiloja tallennetaan jotta myöhemmin voidaan tarkastella millaisia vikatilanteita on ollut. Nipun tila tallentuu erikseen tietokantaan myöhempää käyttöä varten. Anvian reagointijärjestelmään toteutettiin työkalu, jolla nippujen tallennettuja tiloja voidaan tarkastella ja tulostaa niistä raportteja. Ensimmäinen toiminnallisuusraportointi työkalussa on nippujen hakeminen päivämäärän perusteella.

```
1  <script src="http://code.jquery.com/jquery-1.8.2.js"></script>
2  <script src="http://code.jquery.com/ui/1.9.0/jquery-ui.js"></script>
3  <script>
4  $(document).ready(function() {
5      $(function() {
6          $("#alkupvm").datapicker();
7      });
8      $(function() {
9          $("#loppupvm").datapicker();
10     });
11 });
12 </script>
13 <?php
14
15 $datapicker = '<h4>Haku</h4>';
16 $datapicker .= '<form action="#" method="post">';
17 $datapicker .= 'Alkupvm: <input id="alkupvm" name="alkupvm" type="text" size="8"></input>';
18 $datapicker .= 'Loppupvm: <input id="loppupvm" name="loppupvm" type="text" size="8"></input>';
19 $datapicker .= '<button type="submit">Hae</button>';
20 $datapicker .= '</form>';
21
22 if($_POST['alkupvm'] != null && $_POST['loppupvm'] != null) {
23     $startdate = $_POST['alkupvm'];
24     $enddate = $_POST['loppupvm'];
25     list($startm, $startd, $starty) = split('/',$startdate);
26     list($endm, $endd, $endy) = split('/',$enddate);
```

Kuva 17. Toiminnallisuus päivämäärän valitsemiseen.

Päivämäärän syöttäminen voidaan toteuttaa pelkästään tavallisella input–kentällä, tai siinä voidaan käyttää lisänä JavaScriptin jQuery–kirjaston päivämäärän valitsijaa. Kuvassa 17 toteutettuna jQuery päivämäärän valitsija. $datepicker–

muuttujaan tallennetaan html–form, jossa kaksi kenttää tiedon syöttämiseen ja painike jolla tieto välitetään eteenpäin. Riveillä 4-11 jQuery toiminnallisuus joka on liitettty rivien 17 ja 18 input–kenttiin id–tunnusten perusteella. Kun käyttäjä valitsee aktiiviseksi alkupvm- tai loppupvm–kentän, aukeaa kalenterinäkymä josta voi valita halutun päivämäärän. Kalenterinäkymä haetaan JavaScript–lähdetiedostoista, jotka on määritelty riveillä kaksi ja kolme. Lopuksi vielä tarkistetaan, että käyttäjä on syöttänyt sekä alkupäivämäärän että loppupäivämäärän. Riveillä 23 ja 24 otetaan syötetyt päivämäärät vastaan ja tallennetaan ne muuttujiin. Päivämäärää joudutaan usein muokkaamaan, joko muuttamaan esitysmuotoa vastaamaan eri standardiin tai sitten halutaan pilkkoa päivä, kuukausi ja vuosi erilleen. Päivämäärän käsittelyyn on useita eri tapoja ja kuvassa 17 riveillä 25-26 on yksi tapa. Siinä muuttujaan tallennettu päivämäärä pilkotaan kolmeen erilliseen muuttujaan eli päivämäärään, kuukauteen ja vuoteen. PHP:n split–funktio erottaa annetun parametrin perusteella osat toisistaan ja list–funktion avulla annetaan muuttujat joihin osat tallennetaan. Päivämäärien pilkkomisen jälkeen kuukaudenpäivä, kuukausi ja vuosi on kaikki eri muuttujiin tallennettu. Päivämäärän tallentaminen useaan muuttujaan on usein pakollinen välivaihe kun halutaan vertailla päivämääriä tai tallentaa tietokantaan. Esimerkiksi Euroopassa käytetty päivämäärän esitys-standardi eroaa tietokantojen käyttämästä standardista.

Kuvassa 17 otetaan siis vastaan käyttäjän syöttämät päivämäärät jQuery–kirjastoa apuna käyttäen ja erotellaan kuukauden päivä, kuukausi ja vuosi eri muuttujiin. Syötettyjen päivämäärien avulla on tarkoitus hakea tietokannasta ne niput, joissa aloitus- ja lopetus- päivämäärä sopii hakuehtoihin.

```
1  $bundlecreated = $bundledata->created;
2  list($bundley, $bundlem, $bundled) = split('[- ]',$bundlecreated);
3  $bundletime = $bundley.$bundlem.$bundled;
4  $start = $starty.$startm.$startd;
5  $end = $endy.$endm.$endd;
6  if($bundletime >= $start && $bundletime <= $end) {
```

Kuva 18. Tietokannasta haetun päivämäärän käsittely.

Kuvassa 18 rivillä yksi tallennetaan $bundlecreated–muuttujaan tietokannasta haetussa $bundledata–objektissa oleva created–sarakkeen päivämäärä. Seuraavaksi päivämäärä jaetaan kolmeen muuttujaan, muuttujat yhdistetään uudelleen ja verrataan saatua arvoa syötettyihin arvoihin. Päivämäärät siis muutetaan kokonaisluvuiksi, jolloin niitä on helppo vertailla keskenään. Kokonaisluku muodostetaan vuodesta, kuukaudesta ja päivästä. Vuosi on eniten merkitsevä ja päivä vähiten merkitsevä. Kun halutaan hakea päivämäärät tietyltä aikaväliltä, haku on helppoa muodostettujen kokonaislukujen perusteella. Rivillä kuusi verrataan tietokannan päivämäärästä muodostettua kokonaislukua $start ja $end–muuttujien kokonaislukuihin. Jos if-lauseen ehto toteutuu, voidaan nippujen tiedot tallentaa esittämistä varten.

Kun käyttäjä on hakenut päivämäärien perusteella nippuja, näytetään käyttäjälle hakutulosta vastaavat niput joista käyttäjä voi valita haluamansa. Käyttäjän valitsemasta nipusta luodaan raportti. Raportti luodaan Word–tiedostoon, jonka käyttäjä voi tallentaa haluamaansa sijaintiin. Tiedot nipusta ja siihen liittyvistä hälytyksistä haetaan tietokannasta.

```
1  header("Content-type: application/vnd.ms-word");
2  header("Content-Disposition: attachment;Filename=alert_report".date(d-m-Y H:i:s').".doc");
```

Kuva 19. Word–dokumentin luonti.

Kuvassa 19 Word–dokumentin luontiin tarvittava header–määrittely. Määrittelyssä annettu dokumentin nimi on aina yksiköllinen, koska nimessä on päivämäärä ja kellonaika mukana. Itse sisältö raporttiin tallennetaan html–tageja käyttäen. Raportin ulkomuotoa voidaan siis helposti muokata, kuten normaalia web–sivun sisältöä.

5.6 Esitettävän tiedot tulostus

Aiemmissa esimerkeissä on tietokannasta haettua dataa tallennettu apumuuttujiin myöhempää käyttöä varten. Apumuuttujien käyttö parantaa koodin luettavuutta ja helpottaa myöhempää muokkausta. Apumuuttujien käyttö on monessa tilanteessa

pakollistakin, koska tietokannasta haettua dataa ei pystytä esittämään suoraan. Tietokannasta haettavan datan määrä vaihtelee, ja vastaanotettava data joudutaan tallentamaan silmukoiden avulla. Silmukoiden käyttö mahdollistaa dynaamisen tiedon tallennuksen, koska silmukkaa toistetaan niin kauan kun dataa on jäljellä. Silmukoissa voidaan myös esittää erilaisia ehtoja ja näin vaikuttaa apumuuttujaan tallentuvaan dataan. Yhdessä silmukassa voidaan tallentaa dataa moneen apumuuttujaan ja käyttää erilaisia ehtolauseita eri apumuuttujien kohdalla.

```
1  echo '
2  <div><table border="0">
3      <tr>
4          <th width="500px"><h1>Raportin tulostus</h1></th>
5          <th width="500px"></th>
6      </tr>
7      <tr>
8          <td>'.$datapicker.'</td>
9          <td>'.$box_content.'</td>
10         <td>'.$box_contentC.'</td>
11     </tr>
12 </table></div>';
```

Kuva 20. Esitettävän tiedon tulostus.

Kuvassa 20 esitettävän tiedon tulostus, jossa on kolme apumuuttujaa. Runkona tiedon esittämisessä on taulukko jota on helppo muokata haluttuun muotoon. $datepicker–apumuuttujaan on tallennettu päivämäärien syöttökentät sekä hae–painike. Tähän apumuuttujaan tallennetaan sisältöä aina kun käyttäjä valitsee raportin tulostuksen. Kun käyttäjä on valinnut päivämäärät ja painaa hae–painiketta, hakuehtoja vastaava tieto tallennetaan $box_content–apumuuttujaan ja hakutulokset näytetään käyttäjälle. $box_contentC–apumuuttujaan tallennetaan uuden nipun luontiin tarvittavat toiminnallisuudet. $box_contentC sisältää tiedot hälytyksestä jota ollaan viemässä nippuun, kentät nipun tietojen syöttämiseen sekä tallenna–painike. Kaikki kolme apumuuttujaa sisältää siis tietoa eri käyttötarkoituksiin ja niihin lisätään sisältöä tarpeen vaatiessa. Jos käyttäjä esimerkiksi valitsee raportin tulostuksen, vain muuttujaan $datepicker tallennetaan sisältöä ja käyttäjä näkee

vain raporttien hakuun tarkoitetut toiminnallisuudet. Kun käyttäjä hakee nippuja, tallennetaan hakutulokset apumuuttujaan $box_content ja käyttäjä näkee hakua vastaavat niput. Jos käyttäjä valitsee luo uusi nippu, lisätään $box_contentC–muuttujaan sisältöä jolla käyttäjä voi luoda uuden nipun. Käyttäjälle näkyvät toiminnallisuudet esitetään käyttäjän toimenpiteiden mukaan ja samassa koodissa voidaan esittää erilaista sisältöä tarpeen mukaan.

5.7 Tietokannan toteutus

Anvian reagointijärjestelmä käyttää viittä eri tietokantataulua hälytys- ja nipputietojen tallentamiseen. Anvian reagointijärjestelmän toiminnallisuuksilla haetaan ja tallennetaan tietoa tietokantatauluihin. Reagx_alerts -tietokantatauluun ulkopuolinen sovellus päivittää hälytysdataa ja tästä taulusta Anvian reagointijärjestelmä hakee hälytykset. Muita tietokantatauluja tarvitaan nippujen tallentamiseen sekä raportointia varten. Hälytyksiä tallennetaan kolmeen eri tietokantatauluun ja siksi näissä tauluissa osa sarakkeista on samannimisiä ja niissä on sama tietotyyppi. Ensin ulkopuolinen sovellus lisää uuden hälytyksen Reagx_alerts–tauluun, ja reagx_alerts–tauluun. Lisätyt hälytykset haetaan Anvian reagointijärjestelmän hälytysnäkymälle. Jos käyttäjä luo uuden nipun reagx_alerts–taulun hälytykselle, kopioidaan hälytyksen tiedot reagx_bundle_alerts–tauluun. Samalla kun käyttäjä luo nipun, tallentuu hälytys vielä reagx_report_alerts–tauluun raportointia varten.

```
CREATE TABLE `reagx_bundle_alerts` (
  `id` int(10) NOT NULL auto_increment,
  `alert_id` int(10) NOT NULL,
  `bundle_id` int(10) NOT NULL,
  `creator` varchar(25) default NULL,
  `created` timestamp NOT NULL default CURRENT_TIMESTAMP,
  `forced` tinyint(1) default NULL,
  `alertID` varchar(45) NOT NULL,
  `startTime` datetime NOT NULL,
  `priority` varchar(25) NOT NULL,
  `equipment` varchar(45) NOT NULL,
  `location` varchar(45) NOT NULL,
  `alertText` varchar(200) NOT NULL,
  `eqType` varchar(45) NOT NULL,
  `notedWho` varchar(45) default NULL,
  `notedTime` datetime default NULL,
  `company` varchar(45) NOT NULL,
  `monitoringSystem` varchar(45) NOT NULL,
  `status` varchar(15) default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=303 DEFAULT CHARSET=latin1
```

Kuva 21. Luontilause MySQL–tietokantaan.

Kuvassa 21 tietokantataulun luontilause jolla luotiin reagx_bundle_alerts–taulu. Reagx_bundle_alerts–taulussa on yksilöllinen id–numero, kuten kaikissa muissakin Anvian reagointijärjestelmän käyttämissä tietokantatauluissa. Yksilöllisellä id–numerolla taulujen rivejä on helppo yhdistää toisiinsa kun tietoa haetaan tietokannasta. Reagx_bundle_alerts–taulussa on useita eri id–numeroja tallennettu, on taulun oma yksilöllinen id, sekä kaksi id:tä toisista tauluista. Alert_id on reagx_alerts–taulun id ja bundle_id on reagx_bundle–taulun id. Kun käytössä on kolme eri id–numeroa, nippujen ja hälytysten yhdistäminen on helppoa. Kaikki taulun rivit joissa on sama bundle_id kuuluu siis samaan nippuun. Taulun neljäs id, alertID, tulee toisesta järjestelmästä eikä sitä käytetä Anvian reagointijärjestelmässä tietokantataulujen yhdistämiseen. Hälytyksiin täytyy tallentaa myös tieto siitä kuka hälytyksen on lisännyt nippuun ja koska. Creator–sarakkeeseen tallennetaan tieto siitä kuka hälytyksen on siirtänyt nippuun ja created–sarakkeeseen siirtoaika. Raegx_bundle_alerts–tauluun täytyy myös tallentaa hälytyksen tiedot, koska nipuissa olevat hälytykset haetaan tästä taulusta. Sarakkeet joihin tallennetaan hälytyksen tietoja, on kopioitu suoraan reagx_alerts–taulusta. Koska hälytykset kopioidaan suoraan, myös tietokantataulun sarakkeet ja sarakkeiden tietotyypit voi ja pitääkin olla samat. Reagx_bundle_alerts–taulu sisältää siis täysin samat

tiedot hälytyksistä mitä reagx_alerts–taulukin, mutta lisäksi tietoa siitä mihin nippuun hälytys kuuluu sekä kuka hälytyksen on siirtänyt nippuun ja koska hälytys on siirretty nippuun.

6 TESTAUS

6.1 Ensimmäinen vaihe

Anvian reagointijärjestelmään toteutettuja toiminnallisuuksia kehitettiin yksi kerrallaan. Ensimmäinen vaihe oli saada hälytysdata haettua tietokannasta ja esitettyä se hälytysnäkymällä. Kun hälytysdata saatiin haettua näkymälle, voitiin suorittaa ensimmäinen testaus. Tässä vaiheessa hälytysnäkymä ei päivittynyt vielä asynkronisesti, vaan koko sivu päivittyi määräajoin. Tietokantataulun rivejä ja hälytysnäkymää vertaamalla pystyi kuitenkin näkemään että hälytysdata päivittyy oikein. Kun nähtiin että hälytysdata päivittyy, voitiin todeta että rajapinta Anvian reagointijärjestelmän ja tietokannan välillä on oikein määritelty. Myös Anvian reagointijärjestelmän hälytysnäkymän ajastettu päivitys todettiin toimivaksi, koska hälytysnäkymään päivittyi tietokannassa tapahtuvat muutokset.

6.2 Asynkroninen päivitys

Kun ensimmäisen vaiheen toteutus ja testaus oli saatu päätökseen, voitiin lähteä toteuttamaan asynkronista hälytysnäkymän päivitystä. Koska testauksen ensimmäisessä vaiheessa todettiin rajapintakutsut ja tietokanta toimiviksi, voitiin asynkronisen päivityksen testauksessa luottaa niiden toimivuuteen. Asynkronisessa päivityksessä vain erikseen määrätty osa näkymästä päivittyy ja testaus voitiin jakaa osiin. Anvian reagointijärjestelmä käyttää samaa rajapintaa critical-, major-, minor- ja warning-hälytysten hakemiseen, mutta asynkronisessa näkymän päivityksessä testaus voitiin jakaa hälytysten kriittisyyden perusteella osiin. Ensimmäisenä toteutettiin critical-hälytysten asynkroninen päivitys ja päivityksen testaus. Critical-hälytykset päivittyivät ajallaan näkymään eikä testauksessa ilmennyt puutteita toiminnan suhteen. Kun critical-hälytysten päivitys oli todettu toimivaksi, voitiin toteuttaa major, minor ja warning–hälytysten päivitys samalla periaatteella. Seuraavana voitiin testata koko hälytysnäkymän päivitys asynkronisesti. Testaus suoritettiin vertaamalla tietokannassa tapahtuvia muutoksia hälytysnäkymän muutoksiin. Testauksen tässäkään vaiheessa ei todettu mitään isompaa on-

gelmaa hälytysnäkymän päivityksessä ja voitiin todeta asynkronisen päivityksen toimivan.

6.3 Niputuksen testaus ja lopullinen testaus

Anvian reagointijärjestelmän hälytysnäkymän hälytyksiä voi siirtää olemassa olevaan nippuun tai sitten luoda kokonaan uusi nippu hälytykselle. Kun hälytykselle luodaan uusi nippu, vaatii se monta eri toiminnallisuutta, ensimmäisenä tiedot uudesta nipusta pitää tallentaa tietokantaan ja liittää haluttu hälytys nippuun. Testauksen kannalta nipun luonnissa on monta eri toiminnallisuutta jotka pitää todeta toimiviksi. Uutta nippua luotaessa näkymään aukeaa popup–ikkuna, johon voi syöttää uuden nipun tiedot. Uuden nipun luontia testattaessa todettiin popup–ikkunan toiminnallisuudessa virhe, kun käyttäjä valitsi tallenna, jäi popup–ikkuna tyhjänä auki näkymään. Tämä tilanne oli helppo todeta virheeksi toiminnallisuudessa, koska käyttäjä joutui joka kerta erikseen sulkemaan tyhjän popup–ikkunan. Ennen kuin popup–ikkunan toiminnallisuutta alettiin muuttamaan, piti päättää halutaanko siihen vain lisätä sisältöä vai sulkea kokonaan. Todettiin että mitään sisältöä ei tarvitse enää esittää vaan popup–ikkuna voidaan sulkea ja palata takaisin hälytysnäkymään. Kun toiminnallisuus oli muutettu, voitiin todeta popup–ikkunan sulkeutuvan niin kuin haluttiinkin. Nipussa olevat toiminnallisuus-painikkeet oli aluksi sijoitettu erilliseen valikkoon, mutta testauksessa todettiin niille paremmaksi paikaksi nipun päänäkymä. Nipuissa olevien hälytysten taustaväri todettiin päivittyvän hälytyksen tilan mukaan, paitsi silloin kun hälytys on poistunut. Anvian reagointijärjestelmän päänäkymällä hälytyksen taustaväri muuttuu vihreäksi kun hälytys on poistumassa ja lopulta se poistuu näkymästä. Nipuissa olevat hälytykset sen sijaan todettiin jäävän nippuun aktiivisena, vaikka hälytys on poistunut. Nipuissa olevien hälytysten taustaväri päätettiin asettaa vihreäksi kun hälytys ei ole enää aktiivinen. Näin poistuneet hälytykset näkyvät vielä nipuissa, mutta niiden taustaväri on vihreä ja siitä ne tunnistaa poistuneiksi.

Kun Anvian reagointijärjestelmässä oli saatu hälytysdatan esitys ja niputuksen toiminnallisuudet toteutettua ja testattua niiden toimivuus, voitiin suorittaa loppu-

testaus kaikille toteutetuille toiminnallisuuksille. Lopputestauksessa haluttiin varmistaa Anvian reagointijärjestelmän toimivuus kaikissa tilanteissa ja mahdollisesti muuttaa vielä toiminnallisuuksia parempaan suuntaan. Kun lopullinen testaus aloitettiin, oli Anvian reagointijärjestelmässä nipun tilan tallennus vielä manuaalinen ja käyttäjän oli tehtävä tallennus aina muutoksen jälkeen. Manuaalinen nipun tallennus toimi kyllä hyvin, mutta käytettävyyden kannalta se ei vastannut haluttua lopputulosta. Niinpä päätettiin manuaalisen tallennuksen rinnalle toteuttaa automaattinen tallennus, joka tallentaa nipun tilan aina muutoksen jälkeen. Nipun automaattinen tallennus oli ainoa iso muutos toiminnallisuuksiin joka tehtiin lopputestauksen aikana.

7 YHTEENVETO

Tämän opinnäytetyön aiheena oleva Anvian reagointijärjestelmä on laaja kokonaisuus ja valmiina se sisältää monia toiminnallisuuksia sekä kommunikoi useiden eri järjestelmien kanssa. Opinnäytetyötä suunniteltaessa aihetta rajattiin sopivan kokoiseksi ja päätettiin mitä toiminnallisuuksia toteutettaisiin. Keskeisimmät tavoitteet oli saada eri järjestelmien hälytysdata koottua yhteen sekä hälytysnäkymän toimiva päivitys. Myös niputukseen liittyvät toiminnallisuudet, raportointi ja kommunikointi toisiin järjestelmiin haluttiin toteuttaa osana opinnäytetyötä. Osana opinnäytetyötä piti myös suunnitella ja toteuttaa Anvian reagointijärjestelmän käyttämät tietokantataulut. Tietokantatauluihin tallentuu tieto hälytyksistä ja luoduista nipuista.

Anvian reagointijärjestelmään saatiin haettua hälytysdataa niistä järjestelmistä, joista vaatimusten määrittelyvaiheessa haluttiinkin. Myös hälytysnäkymän asynkroninen päivitys saatiin toteutettua ja hälytysnäkymän päivitys toimi ajastetusti kuten haluttiin. Hälytysten niputtaminen vaati monia eri toiminnallisuuksia ja niitä toteutettiin vaiheittain. Ensimmäisenä lähdettiin toteuttamaan niitä toiminnallisuuksia, jotka vaatimusmäärittelyissä oli esitetty. Kun niputus saatiin toimimaan ja päästiin testausvaiheeseen, voitiin todeta joidenkin toiminnallisuuksien vaativan muutoksia. Tehtyjen muutosten jälkeen voitiin tehdä uusi testaus ja todeta toiminnallisuuksien olevan vaatimusten mukaiset. Tietokannan suunnittelu tehtiin Anvian reagointijärjestelmän vaatimusmäärittelyjen pohjalta. Kun toiminnallisuuksia saatiin toteutettua ja testausvaiheessa niihin tehtiin muutoksia, myös tietokantaan jouduttiin tekemään muutoksia.

Anvian reagointijärjestelmän toteutuksessa haastavinta oli hälytysten järjestäminen hälytysnäkymään sekä niputuksen toiminnallisuuksien toteutus. Koska hälytys järjestetään hälytysnäkymälle usein eri muuttujan perusteella, vaati sen toteutus useita eri ehtolauseita ja oli tarkoin mietittävä missä osassa ohjelmaa toiminto voidaan suorittaa. Myös tietokantaan tallennuksessa oli haasteita. Monet tietokan-

taan tallennukset jouduttiin tekemään useaan eri tauluun ja linkittämään tallennettuja rivejä id–numeroiden perusteella. Hälytysten niputtamisessa täytyi ottaa huomioon monta eri tekijä. Kun hälytys siirretään nippuun, se pitää poistua hälytysnäkymältä ja taas palautua sinne jos hälytys poistetaan nipusta. Nippujen esitys hälytysnäkymällä vaati omat toiminnallisuudet ja niiden toteutus oli haastavaa. Nipuissa piti olla painikkeita eri toimintoihin ja nippuja piti pystyä avaamaan sekä sulkemaan. Nipuissa olevat hälytykset piti järjestää kriittisyyden ja tilan mukaan sekä hälytysten taustavärin piti muuttua hälytyksen tilan mukaan. Asynkroninen hälytysnäkymän päivitys vaati useita eri määrityksiä toimiakseen mutta kun yksi näkymän osa saatiin toteutettua, oli helppo toteuttaa muutkin osat.

8 JOHTOPÄÄTÖKSET

Anvian reagointijärjestelmälle alussa asetetut tavoitteet toteutuivat pääosin ja kokonaisuutta ajatellen projekti sujui hyvin. Jo projektin alkuvaiheessa oli tiedossa että kaikkia toiminnallisuuksia ei voida Anvian reagointijärjestelmään toteuttaa tässä opinnäytetyössä vaan osa toiminnallisuuksista toteutetaan myöhemmässä vaiheessa.

Tässä projektissa toteutettiin Anvian reagointijärjestelmän hälytysnäkymä sekä toiminnallisuudet niputukseen ja raportointiin. Hälytysnäkymässä tärkeimpiä toteutettuja toimintoja on näkymän päivitys asynkronisesti sekä hälytysten ryhmittely ja taustavärien määrittely. Myös eri toiminnallisuuspainikkeiden toteutus hälytysriveille ja niihin rakennetut toiminnallisuudet ovat tärkeitä ominaisuuksia. Hälytysten niputukseen toteutettiin useita eri toimintoja. Jokaisesta hälytyksestä käyttäjä voi valita luo uusi nippu ja kun nippu on luotu, siirtyy valittu hälytys nippuun. Käyttäjä voi myös valita jokaisen hälytyksen kohdalta ja siirtää kyseisen hälytyksen jo olemassa olevaan nippuun. Hälytyksen voi myös poistaa nipusta tai siirtää toiseen nippuun. Jos käyttäjä poistaa nipusta hälytyksen joka on vielä aktiivinen, palautuu se takaisin hälytysnäkymään. Aina kun käyttäjä tekee muutoksia nippuihin, tallentuu nipun tila automaattisesti muutoksen jälkeen. Nipun tilan pystyy tallentamaan myös manuaalisesti ja jokaisessa nipussa on painike nipun sulkemiseen. Koska nipussa voi olla kymmeniä hälytyksiä, on jokaisessa nipussa toiminnallisuus nipun pienentämiseen ja suurentamiseen. Kun nippu on pienennetty, näkyy siitä vain nimi ja toiminnallisuuspainikkeet mutta kun nipun suurentaa tulee kaikki nipussa olevat hälytykset näkyviin. Nippujen tallennettuja tiloja pystyy myöhemmin tarkastelemaan raporteista joita voi luoda Anvian reagointijärjestelmän raportointityökalulla. Käyttäjä voi hakea päivämäärän perusteella nippuja ja luoda valitusta nipusta raportin. Raportissa näkyy nipun tiedot sekä kaikki siitä tallennetut tilat.

Tässä työssä ei toteutettu kaikkia Anvian reagointijärjestelmään suunniteltuja toiminnallisuuksia. Työn suunnitteluvaiheessa aihetta rajattiin opinnäytetyöhön sopivaksi ja arvioitiin mitkä toiminnallisuudet voidaan jättää myöhempään toteutusvaiheeseen. Jotta Anvian reagointijärjestelmään tuleviin hälytyksiin voitaisiin reagoida mahdollisimman nopeasti, tarvitaan toiminnallisuuksia joilla tieto hälytyksistä saadaan ilmoitettua eteenpäin. Osa näistä toiminnallisuuksista jätettiin myöhempään toteutusvaiheeseen, kuten tekstiviestin lähetys ja intranet–ilmoitus. Tekstiviestin lähetys–toiminnallisuudella voidaan lähettää viesti vian korjaajalle, viestissä näkyy hälytyksen keskeisimmät tiedot. Intranet–ilmoituksella taas voidaan lähettää ilmoitus hälytyksestä yrityksen intranet–sivuille. Yksi keskeinen Anvian reagointijärjestelmän ominaisuus jota ei vielä tässä vaiheessa toteutettu liittyy raporttien tulostukseen. Tällä hetkellä Anvian reagointijärjestelmässä on toiminnallisuudet luotujen hälytysnippujen tallennukseen sekä raportin luonti tallennetuista nipuista. Raporteista halutaan kuitenkin nähdä asiakasmäärät joita senhetkinen tilanne koskee. Tämä ominaisuus on helposti lisättävissä raportointityökaluun, mutta asiakasmäärien haku on haastavaa eikä siihen ole vielä toiminnallisuuksia olemassa. Tässä opinnäytetyössä toteutettiin monta Anvian reagointijärjestelmälle asetettua toiminnallisuutta ja niillä voidaan seurata hälytyksiä sekä reagoida niihin tarvittaessa. Monia toiminnallisuuksia jäi vielä toteuttamatta Reagointijärjestelmässä ja jatkokehitettävää on vielä paljon. Jo tätä opinnäytetyötä suunniteltaessa oli tiedossa että kaikkia toiminnallisuuksia ei pystytä toteuttamaan ja että toteuttamatta jääneet toiminnallisuudet tullaan toteuttamaan myöhemmässä vaiheessa.

LÄHTEET

/1/ Gilmore, J. & Kuvaja, A. 2005. PHP & MySQL : Tehokas hallinta. Suomennettu painos. Helsinki. Readme.fi

/2/ Smith, D. & Negrino, T. 2007. Javascript– Tehokas hallinta. Suomennettu painos. Helsinki. readme.fi