LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

Virheilmoitusten raportointijärjestelmä PHP - ja Zend Framework 2 pohjaisessa verkkosovelluksessa

Case: eKeiretsu Oy

LAHDEN
AMMATTIKORKEAKOULU
Liiketalouden ala
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Syksy 2013
Niklas Juhonen

Lahden ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

JUHONEN, NIKLAS:

Virheilmoitusten raportointijärjestelmä PHP – ja Zend Framework 2 pohjaisessa verkkosovelluksessa
Case: eKeiretsu Oy

Tietojenkäsittelyn opinnäytetyö 37 sivua

Syksy 2013

TIIVISTELMÄ

Tämä opinnäytetyö käsittelee virheilmoitusten raportointijärjestelmän suunnitelman toteutusta. Päätutkimusongelmana haetaan vastausta siihen, miten PHP:ssä toteutetaan virheilmoitusten raportointijärjestelmä? Alaongelmina selvitetään, miten Zend Framework 2:n käyttö vaikuttaa järjestelmän toteutukseen sekä, miten järjestelmään kirjataan käyttäjien löytämät virheet? Tämä tutkimus on luonteeltaan kvalitatiivinen ja asetettuihin tutkimusongelmiin haetaan vastausta suunnittelutieteellisellä tutkimusmenetelmällä.

Tutkimuksen teoriaosuudessa käsitellään olemassa olevia teknologioita, virheilmoitusten käsittelyä, käyttäjien huomaamia virheitä sekä tietoturvaa. Nämä käsiteltävät aiheet ovat kohdeyrityksen kehittämän verkkosovelluksen määrittämiä. Verkkosovellus on tehty PHP verkko-ohjelmointikielellä sekä sen Zend Framework 2 ohjelmistokehyksen avulla.

Opinnäytetyön kohdeyritys on asettanut työn tutkimustehtäväksi tehdä suunnitelma virheilmoitusten raportointijärjestelmää varten. Tämä suunnitelma tehdään työn empiria osuudessa ja sitä tullaan käyttämään jatkossa tarkemman järjestelmäsuunnittelun pohjana.

Tämän työn tuloksena muodostui yleinen käsitys ja suunnitelma siitä, miten virheilmoitusten raportointijärjestelmä kannattaa toteuttaa PHP:ssä. Työssä myös havaittiin, että Zend Frameworkin käyttö helpottaa tällaisen järjestelmän toteuttamista sekä löydettiin vastaus sille, miten käyttäjien löytämät virheet voidaan kirjata.

Avainsanat: PHP, Zend Framework 2, virheilmoitus, poikkeus

Lahti University of Applied Sciences

Degree Programme in Information Technology

JUHONEN, NIKLAS:

Error message reporting system in a PHP and Zend Framework 2 based web-application
Case: eKeiretsu Oy

Bachelor's Thesis in Information Technology

37 pages

Autumn 2013

ABSTRACT

The purpose of this thesis is to develop a plan for an error message reporting system. The theoretical part of the thesis seeks an answer to the research question how to make an error message reporting system in PHP? Sub-questions try to find out whether the use of Zend Framework 2 affects the system design and also how to log user submitted error reports. This is a qualitative study and it seeks to answer the research problems using a design science research methodology.

The theoretical part of the thesis consists of existing technologies, the processing of error messages, methods to help users report errors and data security. The technologies analysed in this chapter are those used in the case company's web application. The web application is created using the PHP programming language and its Zend Framework 2 program framework.

The goal set by the case company for this thesis was to produce a plan for an error message reporting system. This plan was created in the empirical part of the thesis and it will be used as a basis for future, more detailed system planning.

This study resulted in a general answer to the question how to create an error message reporting system in PHP. The use of Zend Framework 2 was found to be beneficial in making the system more accessible and easier to develop. The thesis also resulted in a general suggestion as to how the application's users could submit their own error reports.

Key words: PHP, Zend Framework 2, error message, exception

SISÄLLYS

1 JOHDANTO

1.1 OPINNÄYTETYÖN AIHE

Tämän opinnäytetyön aiheena on suunnitella kohdeyrityksen kehittämään verkkosovellukseen virheilmoitusten raportointijärjestelmä. Opinnäytetyössä keskitytään ainoastaan virheilmoitusten raportointijärjestelmän suunnitteluun ja sen toteutus aloitetaan vasta myöhemmin. Aiheen selkeyttämiseksi opinnäytetyössä viitataan verkkosovellukseen termillä ”sovellus” ja tämän suunnitelman aiheena olevaan virheilmoitusten raportointijärjestelmään termillä ”järjestelmä”.

Työssä on käytännössä kolme vaihetta. Ensinnäkin tutkin olemassaolevia vaihtoehtoja toteuttaa järjestelmä. Seuraavaksi suunnittelen varsinaisen järjestelmän perustuen yrityksen asettamiin vaatimuksiin ja ensimmäisessä vaiheessa löytämiini vaihtoehtoihin. Lopuksi tiivistän sekä perustelen järjestelmän toteuttamismallin ja tehdyn valinnan.

Ensimmäisessa vaiheessa olen rajannut käytännössä kokonaan pois valmiit kaupalliset järjestelmät,koska kohdeyrityksestä löytyy tarvittavaa tietotaitoa kehittää järjestelmä itse. Kohdeyrityksen pyynnöstä olen kuitenkin työni teoriaosassa käsitellyt muutamaa myös kolmannen osapuolen kaupallista tuottetta, jotka mahdollistavat mallin jossa käyttäjät itse vastaavat virheilmoitusten muodostamisesta.

Järjestelmän tavoitteena on siirtää sovelluksen käyttäjien kohtaamat virheet tietokantaan, josta tieto siirtyy edelleen erilliseen virheilmoitusten hallintaohjelmaan.Virheilmoitustenhallintaohjelmasta sovelluksen kehittäjät pääsevät katsomaan syntyneen virheen tiedot, kuten esimerkiksi luokkaa jossa virhe tapahtui, rivinumeroa, virheen aiheuttanutta tietoa, virhekoodia ja mahdollisesti myös tietoja käyttäjän laitteistosta. Näiden tietojen avulla sovelluksen kehittäjä pystyy nopeasti paikallistamaan virheen sijainnin ja priorisoimaan vian merkityksen sekä selvittämään virheen uusiutumismahdollisuuden muille käyttäjille.

Työssä rajaan pois myös varsinaisen käyttöliittymäosuuden, sillä sovellukseen on tarkoitus kehittää ylläpitonäkymä ja rajapinta johon työni käyttöliittymä sen valmistuttua liitetään. Ylläpitonäkymä ja siihen liittyvä rajapinta suunnitellaan sovellukseen erikseen myöhemmin.

1.2 TOIMEKSIANTAJA JA TARVE TYÖLLE

Opinnäytetyön toimeksiantajana ja kohdeyrityksenä toimii Lahdessa vuoden 2013 keväällä perustettu IT-alan yritys eKeiretsu Oy. Yritys kehittää parhaillaan kunnille ja yrityksille suunnattua verkkopalvelua, joka on tarkoitus julkaista vuodenvaihteessa 2013-2014. Sovelluksen julkaisuun liittyen yritys tarvitsee järjestelmän, jolla sovelluksen kehittäjät saisivat mahdollisimman tehokkaasti selville käyttäjien kohtaamat virheet ja pääsisivät korjaamaan ne nopeasti.

Tehtäväni yrityksessä on toteuttaa tämä järjestelmä, jonka suunnitteluosuus muodostaa tämän opinnäytetyön. Järjestelmän toteuttaminen ja tässä suunnitelmassa esitettyjen asioiden lopullinen arviointi siirtyvät opinnäytetyön jälkeiselle ajalle.

1.3 MIKSI VIRHEILMOITUSTEN KÄSITTELY ON TÄRKEÄÄ?

Lieberman ja Fry esittävätartikkelissaan ajatuksen, että nykypäivän tietokonesovellukset eivät voi koskaan toimia täydellisesti. Niitä on käytännössä mahdotonta saada nykyteknologioilla ja ohjelmointikäytännöillä täysin virheettömiksi. (Lieberman & Fry 2001, 1,2)

Sovelluksessa tapahtuneista virheistä seuraa yleensä virheilmoitus joka näytetään käyttäjälle. Ohjelmointikielissä virheilmoitukset rakennetaan oletusarvoisesti niin, että sovelluksen kehittäjälle eli ohjelmoijalle annetaan mahdollisimman kattava ilmoitus siitä, mikä sovelluksessa ei toiminut ja miksi. Virheilmoitukset on kuitenkin tärkeää muuttaa sovelluksen julkaisun yhteydessä suppeammiksi, jotta sovelluksen tietoa ei vuoda ulkopuolisille, joka puolestaan aiheuttaisi yritykselle tietoturvariskin.

Kun käyttäjä törmää sovelluksessa virheeseen on todennäköistä, että sen viestin sisältö jää käyttäjälle epäselväksi. Käyttäjää voidaan ohjeistaa ilmoittamaan virheestä sovelluksen kehittäjille, mutta kehittäjän on puolestaan erittäin vaikea lähteä korjaamaan virhettä jos käyttäjä ei osaa sitä perusteellisesti kuvailla. Mikäli virheilmoituksia ei saada järjestelmässä jotenkin tehokkaasti kirjattua talteen niin ohjelmassa olevaa virhettä ei välttämättä saada korjattua ja sen käyttö voi muuttua turhauttavaksi.(Lieberman & Fry 2001, 2)

Ottaen huomioon mitä Lieberman ja Fry edellä esittävät ohjelmistoissa esiintyvistä virheistä ja niiden käsittelystä, yritysten olisikin ymmärrettävä että kyseessä on itseasiassa asiakaspalveluun, tuotteiden laatuun ja yrityksen kilpailukykyyn liittyvä asia. Tästä syystä virheilmoitusten käsittelyyn ja sen prosessin hallitsemiseen on syytä kiinnittää huomiota.

1.4 TYÖN TAVOITE JA VAATIMUKSET

Työni tärkeimpänä tavoitteena on tuottaa tutkimuksen ohessa kohdeyritykselle dokumentti, jota voidaan käyttää virheenkäsittelyjärjestelmän toteutuksen pohjana. Opinnäytetyön julkisen luonteen vuoksi siitä jätetään tarkoituksella pois seikkoja, joita kohdeyritys pitää luottamuksellisina. Tällaisiä ovat muun muassa tarkempi kuvaus vielä julkaisemattomasta verkkosovelluksesta sekä tuotteen tietokantarakenne.

Tässä työssä on tarkoitus antaa yleisempi kuva tutkimusaiheesta ja tuottaa suunnitelma, joka voisi olla toteuttamiskelpoinen laajemmallekin ryhmälle kuin pelkästään kohdeyritykselle. Kohdeyrityksen rooli näkyy kuitenkin työssäni mm. valittavien ratkaisumallien tarkastelussa, työlle asetetuissa rajauksissa ja niissä vaatimuksissa mitä teknologioille asetetaan.

1.5 TYÖN RAKENNE

Työni koostuu viidestä kappaleesta. Ensimmäisessa kappaleessa esitellään opinnäytetyön aihe, toimeksiantaja ja perusteet työlle sekä käsitellään työlle aseteltuja tavoitteita ja vaatimuksia.

Kappaleessa 2 esitellään käytettävät tutkimusmenetelmät, asetetaan tutkimusongelmat ja kuvataan tutkimusasetelmä sekä kuvataan aineiston keruusuunnitelma ja käytettävät lähteet.

Kappaleessa 3 esitellään aiheeseen liittyvä teoria, jossa käsitellään ohjelmointiteknologioita, virheilmoituksen muotoja ja erillaisten poikkeustilojen käsittelyä sekä tietoturvaa.

Kappaleessa 4 kuvataan suunnitelman toteutus eli valittu toteutustapa, virheilmoitusten kirjaamismalli sekä virhetietokannan kuvaus.

Työ päätetään kappaleessa 5 muodostamalla yhteenveto, jossa kerrotaan vastaukset tutkimusongelmiin sekä pohditaan saatuja tuloksia tulevaa järjestelmän toteutusta ajatellen.

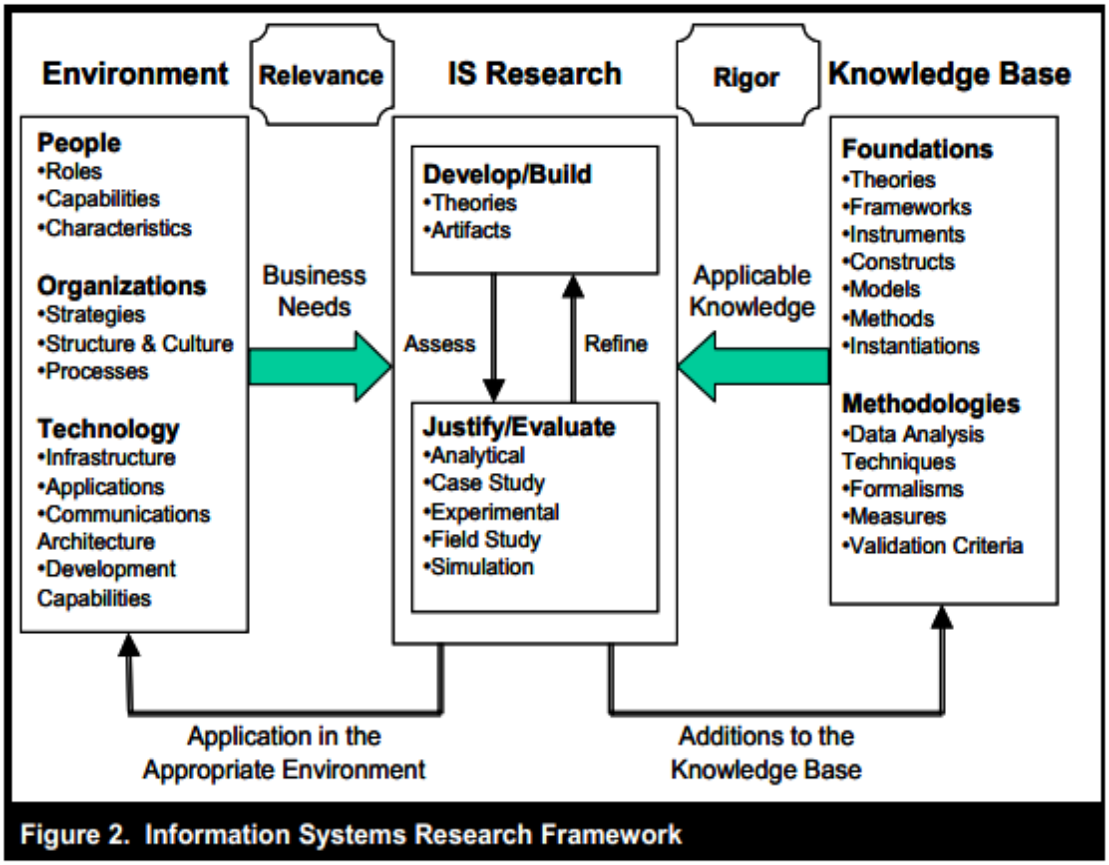
2 TUTKIMUSTEHTÄVÄ JA -MENETELMÄT

Opinnäytetyöni on suunnittelutieteellinen tutkimus, jonka pohjana on käytetty Hevnerin ym. (2004) esittämää tietojärjestelmien tutkimuskehystä. Tutkimukseni on luonteeltaan myös kvalitatiivinen eli laadullinen, jossa aineisto kerätään kirjallisista- ja internetlähteistä, sekä henkilöhaastattelusta. Haastateltava henkilö vastaa kohdeyrityksessä verkkosovelluksen kehittämisestä.

2.1 TUTKIMUSMENETELMÄ

Suunnittelutieteellinen tutkimusmenetelmä perustuu ongelmanratkaisu käytäntöön. Sillä tavoitellaan uusia innovaatioita jotka perustuvat ideoihin, käytäntöihin tai teknisiin ominaisuuksiin, joiden avulla erilaisia toimenpiteitä ja toimintoja tietojärjestelmiin voidaan tehokkaasti toteuttaa. Tällaisessa menetelmässä tutkijan luovuudella ja ongelmanratkaisukyvyllä on suuri merkitys tutkimuksen suorittamisessa. Tässä työssä menetelmällä pyritään etsimään kehitettävälle järjestelmälle ratkaisuja perustuen tarkoituksenmukaisuuteen, käyttäjän kokemaan käytännöllisyyteen ja hyötyyn sekä tiedon ja palvelun laatuun. (Hevner ym. 2004, 76.)

Hevner ym. (2004, 80) esittää kaavion, jonka mukaan suunnittelutieteellistä menetelmää on tarkoitus myös tässä tutkimuksessa soveltaa (KUVIO 1).



KUVIO 1: Suunnittelutieteellisen tutkimuksen kehys. (Hevner ym. 2004, 80).

Kaaviossa keskellä olevaan palkkiin voidaan sijoittaa tässä työssä kehitettävä suunnitelma. Kaavion vasen osa kuvaa työlle haettavia perusteita yrityksen ja sen liiketoiminnan tarpeista. On tärkeää ymmärtää kehitystyön ympäristötekijät, eli ihmiset jotka yrityksessä toimivat, organisaatiot ja niiden väliset toiminnat sekä käytettävissä olevat teknologiat, jotta tutkimustyö voisi onnistua ja tuottaa halutun tuloksen. (Hevner ym. 2004, 80.)

Työskentelen itse kohdeyrityksessä, joten tunnen organisaation ja sen ihmiset entuudestaan. Olen osallistunut myös sovelluksen kehitykseen sen alkuvaiheesta, joten myös käytetyt teknologiat, sovellus ja kehitysmenetelmät ovat minulle entuudestaan tuttuja. Olen myös itse yksi kohdehenkilöistä, joka tulee valmista järjestelmää ja sen hallinnointityökaluja käyttämään. Tämä luo minulle hyvän pohjan perustella suunnitelman toteuttamismahdollisuuksia.

Kaavion oikea puoli kuvaa kerättävää tietämystä, jossa tutkimukseen liitetään teoriat, kehykset, mallit, metodit ja erilaiset menetelmät tiedonkeräämiseen. Jotta

suunnitelma onnistuisi niin tarvitaan siis tietoa yrityksen liiketoimintatarpeista yhdistettynä saatavillaolevaan teoriaan, käytäntöihin ja metodeihin.

2.2 TUTKIMUSONGELMAT JA TUTKIMUSASETELMA

Myöhemmin tämän työn kappaleessa 3 kerrotaan, miksi teknisissä ratkaisuissa on päädytty PHP-ohjelmointikieleen ja sen Zend Framework 2 ohjelmistokehykseen. Nämä valinnat vaikuttavat opinnäytetyön tutkimuskysymysten asetteluun. Opinnäytetyön päätutkimusongelma on:

Miten PHP:ssä toteutetaan virheilmoitusten raportointijärjestelmä?

Päätutkimusongelmaa tarkennetaan kahdella alaongelmalla:

1. Miten Zend Framework 2:n käyttö vaikuttaa järjestelmän toteutukseen?
2. Miten järjestelmään kirjataan käyttäjien löytämät virheet?

Olen rajannut tästä opinnäytetyöstä pois muut PHP:n ohjelmistokehykset kuin Zend Framework 2:n. Olen kuitenkin pyrkinyt käsittelemään aihetta myös yleiseltä kannalta ja tuomaan esille myös muita ratkaisuja kuin vain kohdeyrityksen kannalta suotuisimman.

2.3 TUTKIMUSTEHTÄVÄ

Kohdeyritys on asettanut tutkimustehtäväkseni tutkia, millaisia vaihtoehtoja on olemassa virheiden raportointijärjestelmän toteuttamiseksi sekä mikä näistä mahdollisuuksista olisi yrityksen kannalta suotuisin. Yrityksen tavoitteena on saada käyttöön järjestelmä, joka tallentaa joko automaattisesti tai käyttäjien toimesta tietoa sovelluksen virheistä yrityksen tietokantaan. Virheiden tunnistamiseksi järjestelmän pitää kirjata tietokantaan virheilmoitus, joka sisältää mm. virhekoodin, virheen aiheuttaneen tiedoston nimen, virheen aiheuttaneen funktion nimen ja rivinumeron.

Lisäksi yritys on kiinnostunut käyttäjien laitteistoympäristöstä ja virheen aiheuttaneesta syötteestä tai tilanteesta. Nämä tiedot auttavat sovelluksen ohjelmoijaa mahdollisimman tehokkaasti tunnistamaan virhetilanteen

aiheuttaneen ongelman ja korjaamaan sen. Kuten aiemmin kappaleessa 1.3 todettiin, nykyteknologioilla ja -menetelmillä on lähes mahdotonta toteuttaa sovellusta joka olisi virheetön, vaikka testausta kuinka huolellisesti suoritettaisiinkin. Näin ollen on tärkeää, että tämä käyttäjän ja ohjelmoijan vuorovaikutus toimii tasolla joka hyödyttää kumpaakin toimijaa mahdollisimman paljon.

2.4 AINEISTON KERUUSUNNITELMA

Aineiston kerääminen suoritetaan ensisijaisesti kirjallisista- ja internetlähteistä. Työn tavoitteena on antaa kohdeyritykselle mahdollisimman paljon tietoa erilaisista toteuttamismahdollisuuksista, joka edellyttää aiheesta tuotettuun materiaaliin tutustumiseen mahdollisimman monipuolisesti. Pyrin tarkastelemaan ratkaisuja monipuolisesti huolimatta siitä, että työni aihe on määritelty kohdeyrityksen toimesta esimerkiksi käytettävien teknologioiden osalta etukäteen. Tämä aiheuttaa sen, että kaikkea lähteistä löytyvää informaatiota ei voi suoraan soveltaa tähän työhön. Tämän tiedon kerääminen ja siihen tutustuminen voi kuitenkin auttaa soveltamaan työssäni käsiteltyjä asioita mahdollisesti yleisemmälläkin tasolla.

Aineistoa kerätään myös kohdeyrityksen edustajaa haastattelemalla erityisesti järjestelmälle asetettavien vaatimusten selvittämisessä. Toteutettava järjestelmä on tarkoitettu yrityksen sisäiseen käyttöön ja koska sovellustakaan ei vielä ole julkaistu, en ole voinut työni lähdeaineistona käyttää asiakkaiden haastatteluja.

2.5 KIRJALLISUUSKATSAUS JA LÄHDEAINEISTO

Lähdeaineistona on käytetty pääsääntöisesti internetlähteitä, koska ohjelmoinnissa ja muussa tietotekniikkaan liittyvässä teoriassa tiedon uusiutuvuuson erittäin nopeaa. Pelkästään PHP:stä on vuoden 2013 aikana tähän mennessä julkaistu 13 uutta päivitystä ja versio on muuttunut 5.4.11:sta (17 tammikuuta 2013) 5.5.3:een (22 lokakuuta 2013) (The PHP Group 2013f). Näin ollen teorian täytyy jatkuvasti päivittyä mukana ja esimerkiksi vuonna 2010 julkaistu kirja ei välttämättä tietotekniikassa sisällä enää viimeisintä tietoa. Internetissä olevia teknologioiden

verkkomanuaaleja kuitenkin päivitetään aina tarvittaessa uuden päivityksen julkaisun yhteydessä.

Teknologioiden ja ohjelmointikielien omien verkkosivujen ja verkkomanuaalien lisäksi olen käyttänyt lähteenä myös aiheeseen liittyviä verkkoblogeja. Tämän tyyppisten lähteiden luotettavuutta ei välttämättä aina pysty perustelemaan. Lähteinä käyttämieni blogien kirjoittajat ovat kuitenkin tunnettuja PHP ja Zend Framework osaajia tai työkseen niitä käyttäviä henkilöitä, mutta silti esimerkeissä saattaa ilmetä kirjoitusvirheitä tai esimerkit saattavat olla tehty vanhoille ohjelmaversioille. Näin ollen on tärkeää perustella ja esimerkein todistaa näiden ratkaisujen toiminta työssä.

Kirjallisista lähteistä olen käyttänyt erityisesti George Schlossnaglen kirjoittamaa Advanced PHP Programming kirjaa (2008) sekä muutamia muita Zend Frameworkista, PHP:stä ja Ajaxista kertovia kirjoja. Olen myös haastatellut kohdeyrityksen pääinsinööriä ja nämä haastattelut ovat olleet pohjana suunnittelutyölleni.

Olen jakanut lähdeaineiston käytön siten, että teoriaosuudessa olen pääsääntöisesti käyttänyt edellämainittuja teknologioiden omia verkkomanuaaleja ja suunnitteluosiossa olen perustellut valintojani käytännön esimerkeillä, joissa lähteinä on käytetty myös blogeja sekä kohdeyrityksen edustajan haastatteluja. Jaotteluun päädyin siksi, että teknologiaan ja teoriaan en pysty itse juurikaan vaikuttamaan, joten pyrin välttämään niissä omaa pohdintaa. Suunnittelussa puolestaan oma pohdinta on tärkeää joten pohjustan sen sitä tukeviin esimerkkeihin sekä haastatteluihin.

Aineiston keruun päätyttyä ja niihin perehtymisen jälkeen pidimme kohdeyrityksen edustajien kanssa kehityspalaverin jossa esittelen löytämiäni ratkaisuja, jonka jälkeen yhdessä arvioimme tarjollaolevista vaihtoehdoista yritykselle sopivinta. Vaikka aineiston keräämisen ja analysoinnin päävastuu on ollut minulla niin kuvatunlaisen kehityspalaverin avulla pyrin saamaan yrityksen näkökulman esiin, jotta suunnitelmani olisi kohdeyrityksen kannalta mahdollisimman hyödyllinen.

3 AIHEESEEN LIITTYVÄ TEORIA

3.1 TEKNOLOGIAT

Tässä kappaleessa käsiteltävät teknologiat on valittu sillä perusteella, että kohdeyritys käyttää verkkosovelluksessaan PHP- ja Javascript ohjelmointikieliä sekä PHP:n Zend Framework 2 ohjelmistokehystä. Tällöin on perusteltua käsitellä pääsääntöisesti teknologioita.

3.1.1 Oliopohjainen ohjelmointi

Opinnäytetyöni perustuu varsinkin PHP:n osalta osittain oliopohjaiseen ohjelmointiin (Object Oriented Programming, OOP) ja jotta ratkaisuja olisi helpompi ymmärtää on ensin hyvä selventää hieman perusteita olioista (object), luokista (class) sekä metodeista (method/function).

1. Luokka
   Luokka on tietynlainen suunnitelma muodostettavalle oliolle, tai kokoelma erilaisia toimintoja. Se ei varsinaisesti itse tee mitään ohjelmassa, mutta se pitää sisällään tärkeitä ohjelman toimintoja. (Yaiser 2011.)

2. Olio
   Oliolla tarkoitetaan jotakin ohjelman ajon aikana lähdekoodissa tehtyä osaa, joka voi sisältää monenlaista informaatiota. Olio on käytännössä siis kokoelma informaatiota. Olio tehdään luokan sisällä, joten se on aina oman luokkansa ilmentymä. Yleensä olion sisältämää informaatiota käsitellään metodien avulla (Yaiser 2011.)

3. Metodi
   Olioita käsitteleviä funktioita nimitetään metodeiksi. Metodi on ohjelman lähdekoodissa oleva toiminto. Metodeja voidaan käyttää olioiden sisältämän informaation käsittelyyn, kuten esimerkiksi tässä kappaleessa myöhemmin virheilmoitusten kohdalla esittelen. (Yaiser 2011.)

3.1.2 PHP-ohjelmointikieli

PHP on avoimen lähdekoodin verkko-ohjelmointiin tarkoitettu ohjelmointikieli, jota voidaan käyttää suoraan verkkosivujen HTML lähdekoodin rinnalla lisäämällä sen väliin <?php ?> -liitäntämerkit (The PHP Group 2013d).

PHP lähdekoodi suoritetaan täysin palvelimen puolella, joten kaikki puhtaat PHP kutsut vaativat sivun päivityksen, jotta tulos voidaan näyttää verkkosivun tai sovelluksen käyttäjälle (The PHP Group 2013e). Tämä voidaan kuitenkin kiertää käyttämällä Javascript ohjelmointikieltä, joka puolestaan toimii selaimessa ja voi lähettää kutsuja palvelimen puolella toimivalle PHP:lle. Tämä onnistuu esimerkiksi Ajaxin avulla.

PHP:ssä on myös omat tietokantafunktionsa, joiden avulla voidaan tallentaa ja hakea tietoa esimerkiksi MySQL -tietokannoista. PHP on versiosta 5 lähtien tukenut oliopohjaista ohjelmointia. Se ei kuitenkaan ole täysin oliopohjainen kieli, vaan luo ainoastaan pohjan olioiden käyttämiselle sovellusarkitehtuurissa.

3.1.3 Zend Framework 2

Zend Framework 2 on PHP:lle rakennettu ohjelmistiokehys, joka tuo kehittäjälle valmiin pohjan Model-View-Controller tyyppisen ohjelman kehitystä varten. Tämä tarkoittaa sitä, että ohjelma jaetaan kolmeen osaan:

1. Model
   Model osa sisältää kaiken ohjelman toimintalogiikkaan liittyvän koodin, joka toimii sovelluksen taustalla. (Allen, Lo & Brown 2009, 19-20.)

2. View
   View osa esittää tämän osan käyttäjälle. Yleensä tämä tarkoittaa verkkosovelluksen HTML lähdekoodia, sekä Javascript funktioita. (Allen, Lo & Brown 2009, 19-20.)

3. Controller
   Controller osa hallitsee toimintoja ja kutsuja jotka kulkevat view osan ja model osan välillä. (Allen, Lo & Brown 2009, 19-20.)

Zendissä nämä kolme osaa sisällytetään vielä omiin moduuleihin. Oletuksena ohjelmassa Zendin front controller luonteen vuoksi on yksi moduuli, Application, jonka kautta kaikki ohjelmassa tapahtuvat toiminnot kulkevat (Allen, Lo & Brown 2009, 19). Moduuleita voi tehdä itse lisää ja jokaiselle erityiselle ohjelman toiminnolle, kuten esimerkiksi rekisteröitymiselle on hyvä tehdä oma moduulinsa jotta lähdekoodi pysyy selkeänä eikä yhdessä moduulissa tehdä liian montaa ohjelman asiaa. Tämä helpottaa toimintaa esimerkiksi juuri virheilmoitusten käsittelyssä kun rakenne on selkeä ja voidaan tunnistaa että missä osassa sovelluksen lähdekoodia virhe tapahtui.

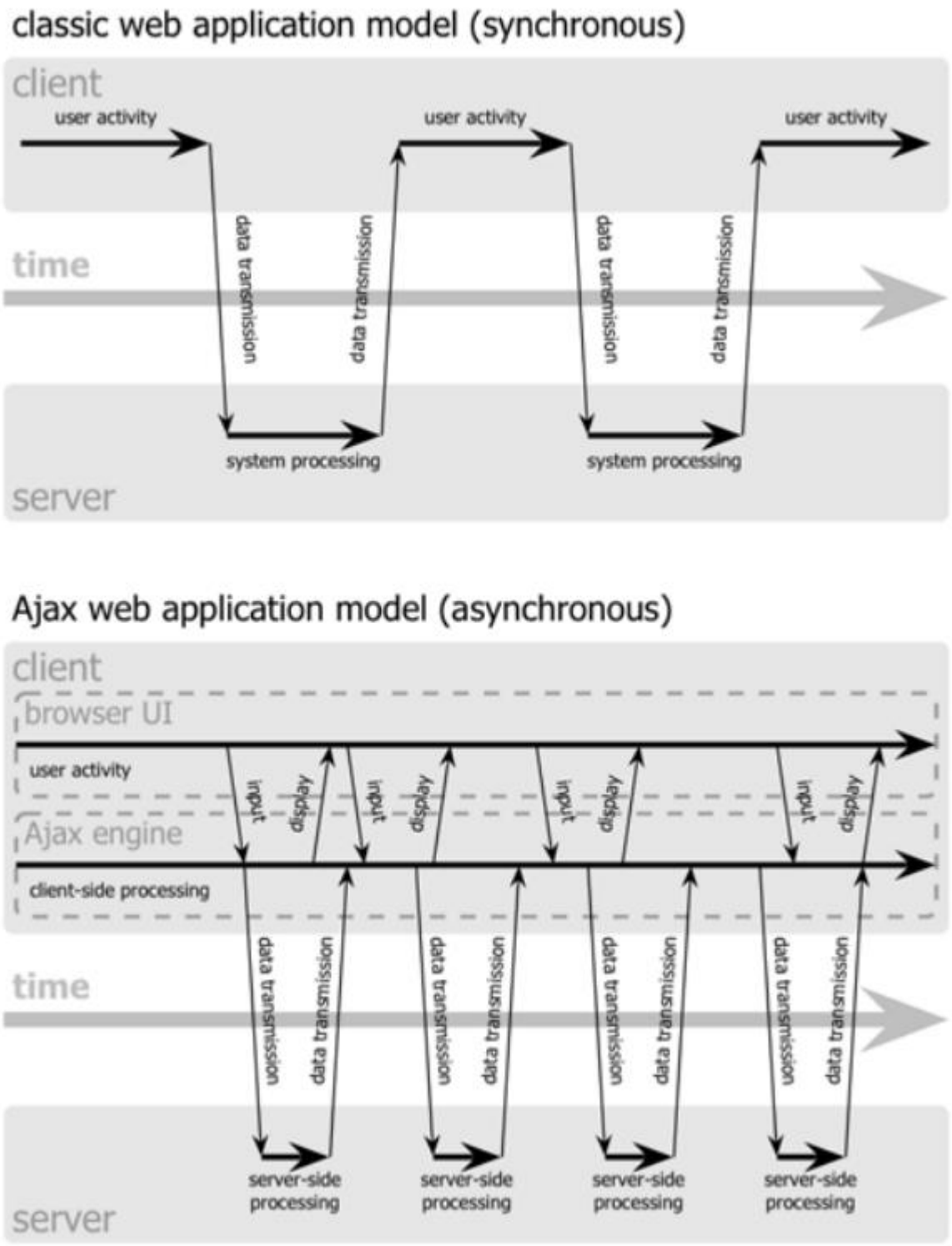
3.1.4 MySQL

MySQL on Oraclen kehittämä ja levittämä avoimen lähdekoodin tietokantojen hallinta järjestelmä (Oracle 2013). MySQL tietokantaan voi tallentaa tietoa ohjelmasta ja sitä voidaan hakea sieltä PHP:n avulla verkkosovelluksissa. Se on täysin ilmainen ja laajasti käytetty.

3.1.5 Javascript

Javascript on käyttäjän eli client puolen ohjelmointiin tarkoitettu ohjelmointikieli. Tämä tarkoittaa sitä, että toisin kuin PHP:ssä, Javascriptiä ei suoriteta palvelimella, vaan käyttäjän omassa selaimessa sivujen käytön aikana. Tämä mahdollistaa esimerkiksi sen, että sivujen sisältöä voidaan muuttaa ilman sivun päivitystä uudelleenhakua palvelimelta. Javascriptiin kuuluu sen oma jQuery kirjasto sekä Ajax -tiedon tallennus- ja hakumenetelmä.

Ajax (asynchronous javascript and xml) toimii siten, että käyttäjä lähettää selaimesta kutsun palvelimelle, mutta toisin kuin perinteisessä mallissa kutsu välitetään palvelimelle asynkroonisesti. Tällöin sekä palvelin, että selain toimivat samaan aikaan (Garret 2005, 2-3). Alla olevassa kuviossa 2 Garret kuvaa tämän toimintaperiaatteen.



KUVIO 2. Ajaxin toimintaperiaate. (Garret 2005, 3.)

Kuvion yläosassa on kuvattuna perinteinen malli, jossa kutsu kulkee synkroonisesti palvelimen ja selaimen välillä. Tämä tarkoittaa sitä, että selaimen pitää odottaa palvelimen vastausta ja ilmoitusta siitä, että kutsu on käsitelty ennenkuin sille annetaan lupa jatkaa. Kuvan alaosassa kutsu kulkee

asynkroonisesti, jolloin selain toimittaa kutsun Ajax-moottorille ja sille annetaan heti lupa jatkaa suoritusta. Ajax-moottori puolestaan välittää kutsun eteenpäin palvelimelle ja kun palvelin aikanaan vastaa moottorille takaisin toimittaa Ajax-tiedon taas selaimelle. Ajax siis eristää nämä kaksi toimijaa toisistaan mikä mahdollistaa sen, että kummatkin voivat toimia omaa tahtiaan toisesta välittämättä. Tämä mahdollistaa esimerkiksi tietojen tallentamisen tai hakemisen ilman sivun päivitystä. (Garret 2005, 1-3.)

3.2 MIKÄ ON VIRHEILMOITUS?

Virheilmoitus on ilmoitus, joka näytetään sovelluksen käyttäjälle, kun sovelluksen lähdekoodin suorituksessa tapahtuu jokin vakavaksi luokiteltu normaalista lähdekoodin suorittamisesta eroava tapahtuma. Näitä tapahtumia kutsutaan nimellä poikkeus (exception). Virheilmoitukset on oletusarvoisesti muotoiltu siten, että siitä näkee suoraan esimerkiksi tiedoston, luokan ja funktion jossa poikkeus on tapahtunut sekä poikkeuksen tyypin. Sovelluksen kehittäjä saa virheilmoituksesta suoraan tiedon mikä osio lähdekoodista tarvitsee korjausta.

Virheilmoituksia voidaan muuttaa myös itse, jolloin niissä voidaan ilmoittaa käyttäjälle, että jokin tapahtuma sovelluksessa epäonnistui. Samanaikaisesti taustalla voidaan suorittaa metodi joka lähettää tiedot virheilmoituksesta kehittäjälle. Myös virheilmoitusten poikkeustyyppejä on PHP:ssa mahdollista muokata itse jolloin virheilmoitusten aiheuttavia tapahtumia voidaan käsin lisätä tai poistaa ohjelmasta. Työssäni keskityn ainoastaan suorituksenaikaisten virheiden (run-time-error) käsittelyyn, sillä muut kuin suorituksen aikaiset virheet näkyvät kehittäjälle suoraan sovelluksen ohjelmoinnin yhteydessä, eikä niitä tarvitse siksi erikseen käsitellä.

Oliopohjaisessa ohjelmoinnissa ja myös PHP:ssä yleinen piirre on, että suorituksen aikaiset virheilmoitukset jaetaan eri alatyyppeihin joita kutsutaan poikkeuksiksi. Javascriptissä ja Ajaxissa toimintaperiaate on käytännössä sama. PHP:ssä näitä alatyyppejä on kolme: virhe eli error tai fatal error, ilmoitus eli notice sekä varoitus eli warning (PHP The Right Way 2013). Jokaiselle kolmelle tyypille on PHP:ssä, kuten monissa muissakin oliopohjaisissa ohjelmointikielissä olemassa omat metodit, joiden avulla voidaan tarttua ohjelman ajossa ainoastaan

tietyn tyyppiseen poikkeukseen. Seuraavaksi käsitellään virheen, varoituksen ja ilmoituksen eroja.

3.2.1 Virhe

Poikkeustyypeistä vakavin on virhe, jolla tarkoitetaan ohjelmassa tapahtuvia merkittäviä poikkeuksia. Toisin sanoen jos ohjelmassa esimerkiksi yritetään kutsua määrittelemätöntä luokkaa tai yritetään alustaa oliota olemattomasta luokasta, antaa ohjelma silloin virhe tyyppisen poikkeuksen (Schlossnagle 2004, 74). Kun virhe-tyyppinen poikkeus on annettu, ei ohjelma voi enää jatkaa suoritusta vaan käyttäjälle näytetään ilmoitus ja suoritus lopetetaan heti. Hyvin usein nämä poikkeukset johtuvat jostain virheestä ohjelman lähdekoodissa ja sen takia niihin on tärkeä puuttua heti (Ghai & Sagar 2009).

3.2.2 Varoitus

Varoitukset eivät ole yhtä vakavia kuin virheet, eivätkä johda PHP:ssä sovelluksen suorittamisen lopettamiseen (The PHP Group 2013b; Schlossnagle 2004, 74). Varoitukset esitetään oletusarvoisesti käyttöliittymässä kuten virheetkin, mutta ne voidaan kytkeä tarvittaessa pois päältä jolloin käyttäjä ei välttämättä edes huomaa että virhe tapahtui koska sovelluksen suorittamista ei lopeteta (Ghai & Sagar 2009). Näin sovelluksesta saadaan yhtenäisemmän oloinen jopa silloin kun kaikki ei aivan toimi niin kuin on tarkoitus. Vakavimmat varoitukset kuitenkin saattavat estää jonkin yksittäisen osan toiminnan sovelluksessa, joten varoituksia seuraava tila on hyvä pitää päällä ja kirjata varoitukset ylös.

3.2.3 Ilmoitus

Ilmoitus-tyyppiset poikkeukset ovat yleensä hyvin neutraaleja. Kun ohjelma antaa jostain tapahtumasta ilmoitus-tyyppisen poikkeuksen ei lähdekoodissa tai suorituksessa välttämättä ole käyttäjän tai ohjelmoijan mielestä edes mitään vikaa. Sovellus voi toimia käyttäjän näkökulmasta hyvin, mutta sovelluksen ajon aikana PHP huomaa esimerkiksi, että jotain tiettyä metodia ei käytetä aivan sille

ominaisella tavalla ja antaa tällöin ilmoituksen sovelluksen loki tiedostoon. Oletuksena ilmoituksia ei PHP:ssä kirjata ylös, mutta on kuitenkin suositeltavaa että sovelluksen kehityksen aikana niitä kirjattaisiin lokiin tiedostoon tai opinnäytetyöni tapauksessa virheilmoitustietokantaan (Ghai & Sagar 2009). Tällöin saadaan selville mitkä osat sovelluksessa eivät välttämättä toimi niinkuin olisi tehokkainta (Schlossnagle 2004, 74). Ilmoituksilla voidaan esimerkiksi ottaa ylös havainnot siitä, että jokin kriittinen toiminto ohjelmassa suoriutui onnistuneesti.

3.3 PHP:N POIKKEUSTEN KÄSITTELY

PHP:ssä, kuten monissa muissa olio-pohjaisissa ohjelmointikielissä on mahdollista käsitellä ohjelman poikkeustilanteita myös manuaalisesti. Poikkeustilalla tarkoitetaan tilaa johon ohjelma suorittaa itsensä kohdattuaan jonkin poikkeuksen, esimerkiksi virheen, ohjelman normaalissa ajossa. (The PHP Group 2013a.)

Kohdeyrityksen verkkosovelluksessa hyvin moni funktio on tällä hetkellä rakennettu siten, että mikäli ohjelman ajossa törmätään virheeseen tai mikäli funktiolle ei toimiteta oikean muotoista parametriä, niin se palauttaa boolean arvon *false*. Tämä estää sovelluksen kaatumisen pienissä käyttäjän polkuihin tai klikkailuihin liittyvissä virheissä sekä muissa funktioiden väärinkäytöissä, kun virheitä aiheuttavia funktioita ei edes suoriteta väärällä syötteellä. Ongelmana *false*-arvon palauttamisessa on kuitenkin se, että PHP kuten muutkaan ohjelmointikielet ei lue sitä poikkeukseksi tai virheeksi. Sovellus on kulkenut polkua joka johtaa lähdekoodin suorittamisen keskeytykseen, mutta sovelluksen kehittäjä ei saa tietoa siitä että jokin on mennyt vikaan. Seuraavassa kuviossa 3 havainnollistan esimerkillä miten *false*-arvon palauttaminen käytännössä toimii.

```
3   function calculate ($number){
4       if(!$number) return false;
5
6       else return 100 / $number;
7   }
8
    $number = 2 ;
10  echo "<p>Ensimmainen lasku: " .calculate($number)."</p>";
11
    $number = 4 ;
13  echo "<p>Toinen lasku: " .calculate($number)."</p>";
14
    $number = 0 ;
16  echo "<p>Kolmas lasku: " .calculate($number)."</p>";
17
```

Ensimmainen lasku: 50

Toinen lasku: 25

Kolmas lasku:

KUVIO3. *Return false* –esimerkki.

Ohjelma esimerkissä kutsutaan rivillä 3 olevaa *calculate* funktiota muuttujalla *$number* joka saa kolme eri arvoa: 2, 4 ja 0. Funktiossa tarkistetaan rivillä 4 onko sille parametrinä lähetettävä *$number* muuttuja olemassa tai onko se saanut arvon nolla. Tämä tarkistus vaaditaan koska funktio laskee rivillä 6 luvun 100 jaettuna annetulla parametrillä. Mikäli parametrillä olisi arvo 0 ja lasku suoritettaisiin niin ohjelma antaisi alla olevan kuvio 4:n mukaisen virheilmoituksen laskutoimituksen yhteydessä.

Warning: Division by zero in D:\wamp\www\OppariProjekti\index.php on line *6*

Call Stack

#	Time	Memory	Function	Location
1	0.0004	247984	{main}( )	..\index.php:0
2	0.0004	248688	calculate( )	..\index.php:16

KUVIO4. Wamp-palvelimen oletusmuotoinen virheilmoitus.

Nyt kuitenkin rivin 4 tarkistuksen ansiosta ohjelman lähdekoodi suoritetaan läpi normaalisti ja kuvion 3 alaosassa olevasta tulostuksesta voidaan huomioida että, kolmannen laskun tulostukselle ei ole annettu arvoa vaan lähdekoodin suorituksessa jakolaskua ei ole suoritettu koska parametrinä annettu muuttujua on ollut 0. Tällä tavalla voidaan helposti välttää epäselvien ja hämmentävien virheilmoitusten näyttäminen ohjelman käyttäjälle.

Ongelmana ohjelman kehittäjälle tässä on kuitenkin se, että ilman kyseistä tarkistusta hän olisi saanut tietää, että lähdekoodin suorituksessa on tapahtunut virhe. Jos lähdekoodiin liitetään tarkistus siitä onko parametri lähetetty arvolla 0 niin ohjelma saattaa näyttää toimivan käyttäjästä aivan oikein, vaikka suorituksessa on tapahtunut selkeä virhe; on yritetty jakaa nollalla. Järkevintä siis olisi virheiden ylöskirjaamista ajatellen käsitellä näitä virheitä ja niiden aiheuttamia poikkeuksia sovelluksen lähdekoodissa muulla tavoin kuin edellämainitulla tarkistuksella.

Näitä poikkeustiloja voidaan PHP:ssä hallita myös try, throw ja catch lohkoilla. Nämä lohkot toimivat siten, että jokin ohjelman toiminto, esimerkissäni jakolasku, laitetaan try/catch rakenteen try lohkon sisään ja sille annetaan catch vastalohko joka toteutuu, mikäli try lohkossa olevassa funktiokutsussa ilmenee jokin poikkeustila (Mynttinen 2009). Seuraavan sivun esimerkissä, kuviossa 5, on havainnollistettuna try, throw ja catch lohkojen käyttö.

```
3  function calculate ($number){
4      if(!$number){
5          throw new Exception("Nollalla ei voi jakaa!", 100);
6      }
7
8      else return 100 / $number;
9  }
10
11 $number = 2 ;
12 try{
13     echo "<p>Ensimmainen lasku: " .calculate($number)."</p>";
14 }
15  catch(Exception $e){
16     echo "Jokin meni pieleen: ". $e->getMessage();
17 }
18
19 $number-=2; //$numberista miinustetaan 2 jolloin siitä tulee 0
20 try{
21     echo "<p>Toinen lasku: " .calculate($number)."</p>";
22 }
23  catch(Exception $e){
24     echo "Jokin meni pieleen: ". $e->getMessage();
25 }
```

Ensimmainen lasku: 50

Jokin meni pieleen: Nollala ei voi jakaa!

KUVIO 5. Try, throw ja catch lohkojen käyttö PHP:ssä. (The PHP Group 2013a.)

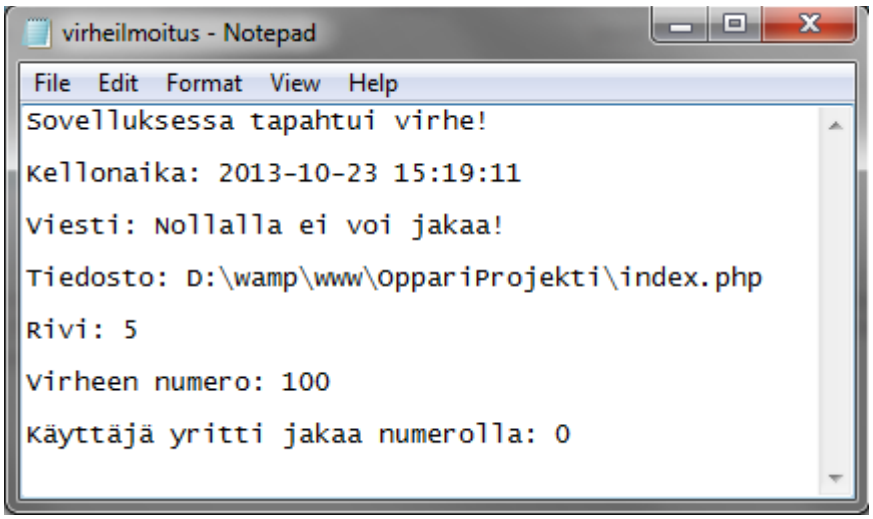
Ohjelma laskee luvun 100 jaettuna muuttujan *$number* arvoilla 2 ja 0. Edellisestä esimerkistä poiketen tälläkertaa *calculate* funktiossa rivilla 4 olevan tarkistuksen sisään on *false* arvon palautuksen sijasta asetettu throw-lohko. Mikäli ohjelma ajautuu tarkistuksen sisään, eli mikäli parametrinä lähetettyä *$numberia* ei ole olemassa tai se on nolla niin PHP antaa palautusarvona uuden *Exception*-luokan olion ja antaa sille viestiksi ”Nollalla ei voi jakaa!”

Kun rivillä 21 oleva laskutoimitusfunktion kutsu, eli nollalla jakamisen try lohko huomaa, että PHP palauttaa vastauksena funktion kutsuun poikkeuksen, suoritetaan ohjelma rivillä 23 olevaan catch-lohkoon, jossa parametrinä otetaan *Exception*-luokkaa oleva olio *$e*. Tämä *$e* on sama olio joka luodaan throw lohkossa rivillä 5. Lopulta catch-lohko tulostaa *getMessage()* metodilla käyttäjälle ilmoituksen ”Jokin meni pieleen: Nollalla ei voi jakaa!”

Catch lohkoa voidaan edelleen laajentaa tutkimusongelmani suuntaan. Olio *$e* sisältää taulukkona (array) koko virheilmoituksen, ja kuviossa 6 olen laajentanut ohjelman catch lohkoa siten, että oliosta eritellään tietoja omille riveilleen ja ne tallennetaan kuvion 7 *virheilmoitus.txt*-tekstitiedostoon joka sijaitsee samassa kansiossa kuin itse ohjelma.

```
19      $number-=2; //$numberista miinustetaan 2 jolloin siitä tulee 0
20      try{
21          echo "<p>Toinen lasku: " .calculate($number)."</p>";
22      }
23       catch(Exception $e){
24          echo "Jokin meni pieleen: ". $e->getMessage();
25
26          $file = 'virheilmoitus.txt'; // Tiedosto johon tiedot kirjataan.
27          $log = file_get_contents($file);
28          $log .= "Sovelluksessa tapahtui virhe!";
29          $log .= "Kellonaika: ". date('Y-m-d H:i:s');
30          $log .= "Viesti: " .$e->getMessage();
31          $log .= "Tiedosto: " .$e->getFile();
32          $log .= "Rivi: " .$e->getLine();
33          $log .= "Virheen numero: " .$e->getCode();
34          $log .= "Käyttäjä yritti jakaa numerolla: " .$number;
35          file_put_contents($file, $log);
36      }
```

KUVIO 6. Virheilmoituksen tallennus tekstitiedostoon. (The PHP Group 2013g.)



KUVIO 7. Tallennettu virheilmoitus.

Tekstitiedostoon tallennus voitaisiin samalla tapaa tehdä myös tietokantaan. Jos verrataan ylläolevaa kuviota 7 kuvion 4 virheilmoitukseen ja lisätään edelliseen vielä rivinvaihdot niin voidaan havainnoida miten virheilmoituksen informaation esittäminen selkeytyy kun se muotoillaan itse.

3.4 POIKKEUSTEN KÄSITTELY ZEND FRAMEWORK 2:SSA

Yrityksen verkkosovellus on rakennettu PHP:n Zend Framework 2 ohjelmistokehystä hyväksikäyttäen, joten käsiteltävissä teknologioissa täytyy ottaa myös huomioon se miten Zend Frameworkissa poikkeuksia käsitellään.

Zend Frameworkissa poikkeustenkäsittelyä on helpotettu tekemällä mahdolliseksi käsitellä koko ohjelman tapahtumia yhdessä paikassa (Allen, Lo & Brown 2013, 4). Kaikki Zendin MvcEvent tyyppiset tapahtumat kulkevat tiedostopolun *module/Application/module.php*:ssä sijaitsevan *onBootstratp()* funktion kautta. Käytännössä kaikki sovelluksen palvelimella tapahtuvat toiminnot ovat Zend Frameworkissa MvcEvent tyyppisiä tapahtumia. Näin ollen on mahdollista liittää kyseiseen funktioon poikkeustenkäsittelylause, joka kuuntelee koko ohjelman toimintoja ja jos ohjelmassa havaitaan jokin poikkeus niin se tulee funktioon ja sitä voidaan käsitellä siinä (Zend Technologies Ltd 2013). Tämän ansiosta ohjelmoijan ei tarvitse esimerkiksi lisätä jokaisen moduulin controlleriin erikseen omaa poikkeustenkäsittelyä tai jokaiseen funktioon erikseen, joten järjestelmän ja koodin ylläpito helpottuu huomattavasti. Kuviossa 8 on kuvattuna tämä Zendin tiedostorakenteesta löytyvän module.php tiedoston oletus *onBootstrap* –funktio.

```
10  namespace Application;
11
12  use Zend\Mvc\ModuleRouteListener;
13  use Zend\Mvc\MvcEvent;
14
15  class Module
16  {
17      public function onBootstrap(MvcEvent $e)
18      {
19          $eventManager        = $e->getApplication()->getEventManager();
20          $moduleRouteListener = new ModuleRouteListener();
21          $moduleRouteListener->attach($eventManager);
22      }
```

KUVIO 8. Zend Framework 2 onBootstrap -funktio.

Kuviossa 9 puolestaan tähän samaan funktioon on liitetty Rob Allenin esimerkin mukaisesti tapahtumakuuntelija, eli event handler. Sen tarkoituksena on ikäänkuin kuunnella sovelluksessa tapahtuvia kutsuja ja mikäli jossain ohjelman toiminnossa tapahtuu *dispatch.error* tyyppinen virhe, niin kuuntelija huomaa sen ja ajaa ohjelman rivin 20 anonyymiin funktioon. (Allen 2013.)

```
17      public function onBootstrap($e)
18      {
19          $eventManager = $e->getApplication()->getEventManager();
20          $eventManager->attach('dispatch.error', function($event){
21              $exception = $event->getResult()->exception;
22              if ($exception) {
23                  $sm = $event->getApplication()->getServiceManager();
24                  $service = $sm->get('Application\Service\ErrorHandling');
25                  $service->logException($exception);
26              }
27          });
28      }
```

KUVIO 9. Tapahtuman käsittelijä onBootstrap funktiossa. (Allen 2013.)

Tätä esimerkkiä voidaan jatkaa vielä Allenin esimerkkiä mukailemalla siten, että poikkeukset kirjattaisiin loki tiedostoon ja viesti esitettäisiin käyttäjälle. Zendissä siis on edellämainitun lisäksi mahdollista suorittaa poikkeustenkäsittely useammalla tavalla, kuten PHP:n omaa mallia mukaillen funktioissa tai controllereissa erikseen. Käytännössä kuitenkin helpointa on tehdä koko sovelluksen virheiden raportointi yhdessä paikassa.

3.5 JAVASCRIPT POIKKEUSTENKÄSITTELY

Hyvin suuri osa verkkosovelluksen selainpuolen lähdekoodista ja toiminnoista on tehty Javascriptiä sekä sen jQuery-kirjastoa ja Ajax-kutsuja käyttäen. Koska Javascript toimii täysin käyttäjän selaimessa, ei sen poikkeuksia voi käsitellä palvelimella samassa paikassa kuin PHP:n poikkeuksia. Javascriptin itsensä virheet tai poikkeukset eivät sinänsä ole sovelluksessa ongelma, sillä ne voivat pahimmillaankin vaikuttaa ainoastaan käyttäjälle esitettävän tiedon muotoiluun, mutta koska Ajax kommunikoi palvelimen ja selaimen välillä voi siinä tapahtua jokin poikkeus joka on selkeästi lähdekoodin looginen virhe. Näin ollen Ajax poikkeukset täytyy myös kirjata ylös. Javascriptin virheiden raportointi on todennäköisesti järkevämpi jättää käyttäjille. Tätä käsitellään työn seuraavassa osiossa.

Kuviossa 10 normaalimuotoinen jQueryllä tehty Ajax kysely pitää yleensä sisällään success ja error haarat. Nämä haarat toimivat samalla periaatteella kuin PHP:n try ja catch lohkot.

```
1  $.ajax({
2      type: "POST",
3      url: "testi.php",
4      success: function(){
5          // Kun ajax kutstu suoritetaan onnistuneesti.
6      },
7      error: function(XMLHttpRequest, textStatus, errorThrown) {
8          // Kun ajax kutsussa tapahtuu virhe
9      }
10 });
```

KUVIO 10. Ajaxin success ja error haarat.

Kun Ajax suoritetaan onnistuneesti niin sen success haaran sisällä oleva lähdekoodi toteutetaan. Mikäli puolestaan Ajaxissa tapahtuu virhe, niin suoritetaan error haarassa oleva lähdekoodi. Käytännössä siis Ajaxin virheilmoituksiin pääsee käsiksi samalla periaatteella kuin PHP:n virheilmoituksiin.

3.6 KÄYTTÄJÄN HUOMAAMAT VIRHEET

Vaikka voidaan olettaa, että suurin osa vakavista virheistä aiheuttaa ohjelmassa poikkeustapahtuman, joka johtaa automaattisiin toimenpiteisiin, ei välttämättä kaikki virheet tai väärät toiminnallisuudet ole sovelluksen mielestä virheitä vaikka käyttäjä voi huomata sovelluksen toimivan väärin. Näitä poikkeuksia ei luonnollisesti voi käsitellä tällöin automaattisesti vaan käyttäjällä täytyy olla mahdollisuus itse ilmoittaa virheestä tai myös kehitys ehdotuksista.

Haasteena käyttäjien itse tekemissä virheilmoituksissa yrityksen ja sovelluksen kehittäjien kannalta on se, miten käyttäjien ilmoittamista virheistä saadaan tarpeeksi tietoa, jotta niitä voidaan käyttää pohjana virheiden korjaamiselle. Mikäli käyttäjä täyttää esimerkiksi lomakkeen, joka lähetetään sovelluksen kehittäjille, on mahdollista, että hän kuvailee virhe tapahtumaa ja siihen johtanutta polkua ja käytettyä tietoa mahdollisesti vain yhdellä lauseella. Tällöin virheen korjaaminen saattaa olla erittäin vaikeaa.

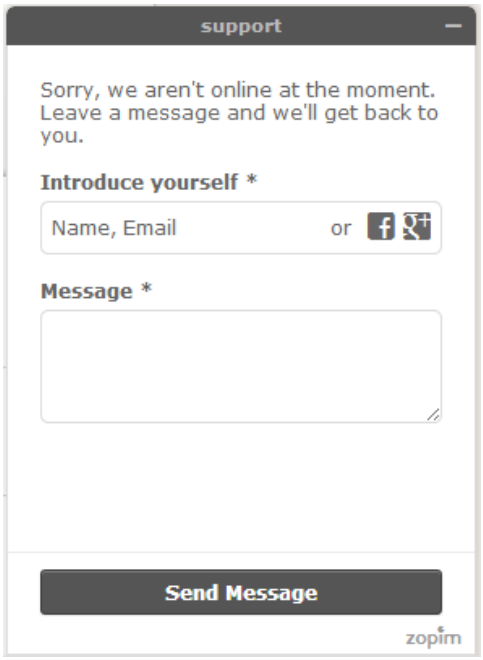
Google tarjoaa omaa Analytics-palvelua jonka kautta on mahdollista seurata sovelluksen käyttäjän käyttäytymistä verkkosivulla, ja näin ollen virheenilmoituslomakkeen lähetyksen yhteydessä on mahdollista liittää lomakkeeseen käyttäjän Analytics-tiedot. Tällöin käyttäjien lähettämistä tiedoista voidaan saadaa irti enemmän informaatiota. Ongelmana tässä kuitenkin on se että käyttäjä eivät välttämättä halua, että heidän tietojaan kerätään sivun käyttämisen yhteydessä.

Google Analyticsin ohella on myös mahdollista käyttää HTTP-evästeitä joita voidaan käyttää esimerkiksi käyttäjän verkkosovelluksessa kulkeman polun ja painamien nappien seuraamiseen (Microsoft 2013).

Yhtenä vaihtoehtona käyttäjien palautteen keräämiseksi on myös Zopim nimisen yrityksen tarjoama palvelu, jossa verkkosivulle liitetään suora viestittely ominaisuus, jonka kautta sivuston käyttäjä voi ottaa itse suoraan yhteyttä sovelluksen kehittäjiin tai asiakas palveluun (Zopim 2013).

Palvelu mahdollistaisi sen, että sovelluksen kehittäjä voi suoraan kysyä tarvittavaa lisätietoa asiakkaalta, jolloin vältetään edellä mainittu ongelma siitä,

miten saada tarpeeksi informaatiota käyttäjältä. Alla olevassa kuviossa 11 on esiteltynä Zopimin omilta verkkosivuilta esimerkki tälläisestä käyttäjäpalaute chatista.



KUVIO 11. Zopim käyttäjäpalauteikkuna. (Zopim 2013.)

Edellä esitetyn kaltainen lomake voidaan lisätä suoraan verkkosivuille esimerkiksi oikeaan alakulmaan pienenä linkkinä. Kun käyttäjä klikkaa linkkiä hänelle avautuu kuvassa esitelty laajempi yhteydenotto lomake, jonka kautta hän voi keskustella suorana yrityksen help desk henkilön kanssa tai jos yrityksen edustaja ei ole paikalla, voi siitä jättää myös sähköpostia.

3.7 TIETOTURVA

Yksi olennaisista syistä, miksi verkkosovellus saattaa tarvita itsetehtyjä poikkeusten käsittelyitä on se, että oletuksena PHP:n ja Zend Frameworkin sekä Javascriptin virheilmoituksissa näytetään niinsanottu stack trace eli polku, jota ohjelma on virheeseen päätynyt sekä luokka, metodi ja rivi, jossa virhe on tapahtunut. Nämä tiedot halutaan luonnollisesti pitää piilossa ulkopuolisilta, jotta ohjelman muuttujien, metodien, luokkien tai tiedostojen nimet eivät päädy vääriin käsiin. Näin tapahtuessa ulkopuolinen henkilö voi kalastella tietoa järjestelmästä

syöttämällä tarkoituksella väärää tietoa ohjelmaan ja ottamalla virheilmoituksista tarvittavat tiedot sovelluksen murtamiseen (Schlossnagle 2004, 76-77; The PHP Group 2013c).

Käyttäjälle halutaan siis virheilmoituksessa näyttää mielummin esimerkiksi teksti jossa kerrotaan virheen tapahtuneen ja pahoittelut kuin, että näytettäisiin oletusmuotoinen virheilmoitus. PHP:ssä virheilmoitus on mahdollista ottaa kokonaan pois käytöstä ohjelman julkaisun yhteydessä. Tällöin sovelluksesta tulee tietoturvallisempi, mutta koska virheet tyypillisesti vain lisääntyvät julkaisun yhteydessä, ei se välttämättä ole järkevää. Opinnäytetyöni tarkoitus onkin löytää ratkaisu juuri julkaistun sovelluksen virheilmoituksiin.

Tietoturvaan liittyy myös yrityksen vastuu siitä, mitä tietoja saa tallentaa. Esimerkiksi jos sisäänkirjautumisessa tapahtuu virhe, niin virheilmoituksista ei saa tallentaa salasanaa tai käyttäjän muita henkilökohtaisia tietoja. Käyttäjän tietojen luottamuksellisuus on siis myös huomioitava järjestelmää suunnitellessa. Toinen käyttäjän tietojen luottamuksellisuuteen liittyvä haaste on evästeiden ja muiden käyttäjän toimintoja seuraavien teknologioiden käyttö.

4 SUUNNITTELU

Tässä kappaleessa valitsen käsittelemistäni teknologioista parhaiten sopivat vaihtoehdot kohdeyritykseni sovellusta silmälläpitäen. Aiemmissa kappaleissa käsitellyistä teknologioista ja yrityksen toiveista muodostuvalle pohjalle on tarkoitus muodostaa tässä kappaleessa suunnitelma tavoitteena olevan järjestelmän toteuttamiseksi.

4.1 SOVELLUKSESSA KÄYTETYT TEKNOLOGIAT

Kuten kappaleessa 3 mainitsin, kohdeyrityksen kehittämä verkkosovellus on tehty PHP ja Zend Framework 2 ympäristöön ja suuri osa käyttöliittymästä on tehty Javascriptillä ja jQueryllä. Tietojen säilyttämiseen on käytetty MySQL pohjaista tietokantaa. Nämä käytössä jo olevat teknologiat määrittelevät hyvin pitkälle sen minkä tyyppiseen ratkaisuun suunnitelmassani päädyn. Yrityksen edustajan (eKeiretsu Oy 2013a) mukaan se, että siirtyisimme käyttämään jotain muuta PHP ohjelmistokehystä kuin Zend Framework ei vaikuta realistiselta vaihtoehdolta. Myöskään PHP:n vaihtaminen johonkin toiseen teknologiaan ei näytä tällähetkellä realistiselta joten nämä valinnat yrityksen puolelta toimivat suunnitelmani perusteena.

4.2 VALITTU TOTEUTUSTAPA

Aiemmin kappaleessa 3 toin esille sen, että kun sovellus siirretään julkiseen käyttöön, täytyy sen virheilmoitukset muuttaa kehitysversion ohjelmoijaa helpottavista ilmoituksista joko siistimpään ja tietoturvallisempaan muotoon, jossa käyttäjälle näytetään vain yksinkertainen ilmoitus siitä, että jokin ohjelman toiminto ei suoriutunut oikealla tavalla, tai virheilmoitukset tulisi ottaa pois käytöstä kokonaan. Koska opinnäytetöni aiheena on toteuttaa kohdeyritykselle järjestelmä, joka kirjaa julkaisun jälkeen edelleen virheitä ylös, jätän suunnittelussa huomioimatta kokonaan sen vaihtoehdon, että ilmoitukset otettaisiin pois käytöstä.

Ratkaisussa päädyttiin yhdessä yrityksen edustajan (eKeiretsu Oy 2013b) kanssa siihen, että PHP:n poikkeusten käsittely tullaan tekemään Zendin

module/Application/module.php tiedoston *onBootstrap* funktiossa. Tämä mahdollistaa sen, että koodia ei tarvitse sijoittaa eripuolille ohjelmaa jokaisen moduulin controlleriin erikseen, vaan kaikki voidaan tehdä yhdessä paikassa. Näin lähdekoodi pysyy helpommin hallittavana ja mikäli esimerkiksi virheilmoituksista halutaan lisää tai vähemmän tietoa on sen tekeminen helpompaa ja nopeampaa kuin muuttaa lähdekoodia erikseen jokaisessa controllerissa.

Mikäli poikkeustenkäsittely tehtäisiin ohjelmistokehyksestä riippumattomaksi niin käytännössä sovelluksen tärkeimmät toiminnot tulisi laittaa kappaleessa 3 esiteltyjen try-throw-catch lohkojen sisälle, jossa try-osio kutsuu ensin toimintoa ja catch-osioon olisi varmuuden varalta liitetty poikkeustenkäsittely. Zend helpottaa työtä tässä suhteessa siis huomattavasti, vaikka käytännössä ratkaisu on kummallakin menetelmällä ajatuksellisesti sama. Zendissä se tarvitsee vain tehdä ainoastaan yhdessä paikassa.

Javascriptissä puolestaan yritys tarvitsee tietoa ainoastaan Ajaxin antamista virheilmoituksista. Kun Ajax kutsu lähetetään palvelimelle niin kutsuun liitetään Ajaxin omat *success* ja *failure* funktiot. Ajaxin kutsuma PHP action controllerissa palauttaa muun tiedon ohessa muuttujan *success* joka saa arvon 1 silloin kun tieto palautetaan onnistuneesti. Tällöin kutsutaan *success* metodia. Jos taas tiedon haku tai lähetys epäonnistuu niin palautetaan success muuttujalle arvo 0, joka puolestaan kutsuu Ajaxin *failure* metodia. Tähän *failure* metodiin voidaan liittää Ajaxin poikkeusten käsittely.

Javascriptissä itsessään jQuery kirjastonsa kanssa ei sovelluksessa voi tapahtua suurta loogista virhettä vaan mahdolliset virheet niissä vaikuttavat vain sivun ulkoasuun ja käyttöliittymän elementtien asetteluun, joten näiden osalta päädyimme siihen tulokseen, että järkevintä olisi antaa käyttäjän itse ilmoittaa niistä. Käyttöliittymän ulkoasuvirheet on myös ohjelmoijalle suhteellisen helppo huomata joten en pidä niitä niin kriittisinä, että ne tarvitsisi tallentaa tietokantaan automaattisesti. Tämä rasittaisi sovellusta sekä tietokantaa turhilla kutsuilla.

Jotta käyttäjä voisi helposti ilmoittaa huomaamistaan virheistä, päädyin siihen tulokseen, että kappaleessa 3 esitellyn Zopim-palvelun kaltainen ratkaisu olisi järkevin. Palvelun käytön kokonaishinnasta ei vielä ole täyttä selvyyttä, mutta

ajatuksellisesti olisi käytännöllistä jos sovelluksen käyttäjä voisi suoraan ottaa yhteyyttä kehittäjiin. Tällöin kehittäjät voisivat kysellä tarkentavia kysymyksiä ja vältyttäisiin epämääräisiltä käyttäjäpalautteilta. Hyödyllistä olisi myös ottaa käyttöön Googlen Analytics palvelu, joka mahdollistaisi käyttäjien toiminnan seuraamisen sovelluksen sisällä. Tämä puolestaan helpottaisi sovelluksen jatkokehitystä siltä osin että saataisiin selville esimerkiksi ne sivut joilla käyttäjät viettävät eniten aikaa ja sivut joilla käyttäjät joutuvat klikkailemaan eniten. Käyttäjien tiedoista ei kuitenkaan saa missään nimessä kirjata ylös mitään henkilökohtaista tietoa, sillä ne ovat sekä Googlen käyttöehtojen vastaista, että monen maan lakien vastaista (Google 2013).

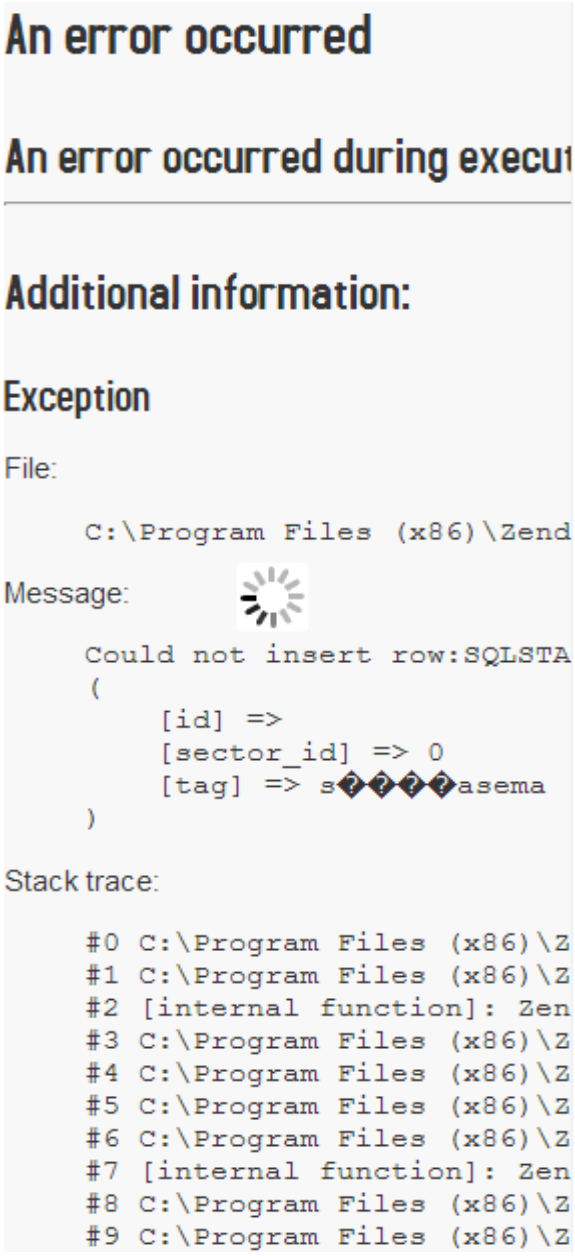
Haastatellessani kohdeyrityksen pääinsinööriä (eKeiretsu Oy 2013a) kävi ilmi, että yritys ei pelkästään halua tietoa virheilmoituksista, vaan sovelluksessa on tiettyjä toimintoja joiden toiminta on todella tärkeää ja halutaan varmistua siitä että ne todella toimivat. Näin ollen tulen liittämään kyseisiin toimintoihin samankaltaiset automaattiset lokifunktiot jotka kirjaavat ylös tiedot toiminnon suorituksesta myös silloin kun se toimii oikein. Tämä tulee kuitenkin todennäköisesti lisäämään tietokantaan kirjattavien ilmoitusten määrää hyvin paljon, joten järkevintä on pitää oikein suoritetuista toiminnoista lokia ainoastaan silloin kun sovellukseen on tehty jokin laajempi päivitys tai jos halutaan todella varmistaa sovelluksen toimiminen.

Tietokantarakenteessa päädyin siihen ratkaisuun että sovellusarkkitehtuurisesti kaikkein järkevin ratkaisu olisi tallentaa virheilmoituksista saatava tieto kokonaan eri tietokantaan kuin missä varsinaisen verkkosovelluksen tieto ja taulut sijaitsevat. Sovelluksen oma tietokanta tulisi todennäköisesti kuormittumaan liikaa täysin sovelluksen toimintaan liittymättömistä syistä mikäli käyttäisimme samaa tietokantaa ja tulevaisuudessa yritys tarvitsee jokatapauksessa uuden tietokannan erilaisia järjestelmänvalvojan ominaisuuksia varten.

4.3 MITÄ TIETOA ILMOITUKSISTA KIRJATAAN

Zend Frameworkin oletusvirheilmoituksessa näytetään virheestä aina virheen numerokoodi, tiedosto jossa virhe tapahtui, virheen viesti sekä polku jota pitkin virheeseen päädyttiin. Kuviossa 12 on kuvattuna yleinen Zend Frameworkin

virheilmoitus, joka on tapahtunut kohdeyrityksen verkkosovelluksessa. Kuten kuviosta voidaan huomata on Zend Frameworkin oletusvirheilmoitus suhteellisen epäselvä, joten siitäkin syystä on perusteltua kiinnittää siihen huomiota.



KUVIO 12. Virheilmoitus Zend Framework 2:ssa.

Jotta automaattisista virheilmoituksista saataisiin tarpeeksi kattava on siihen lisättävä vähintään yllä olevan virheilmoituksen tiedot. Haastatellessani yrityksen edustajaa päädyimme yhdessä siihen lopputulokseen että automaattisista ilmoituksista täytyy kerätä ylös oletuksena annettavien tietojen lisäksi myös esimerkiksi käyttäjän tunnusnumero (eKeiretsu Oy 2013a). Mikäli ongelma

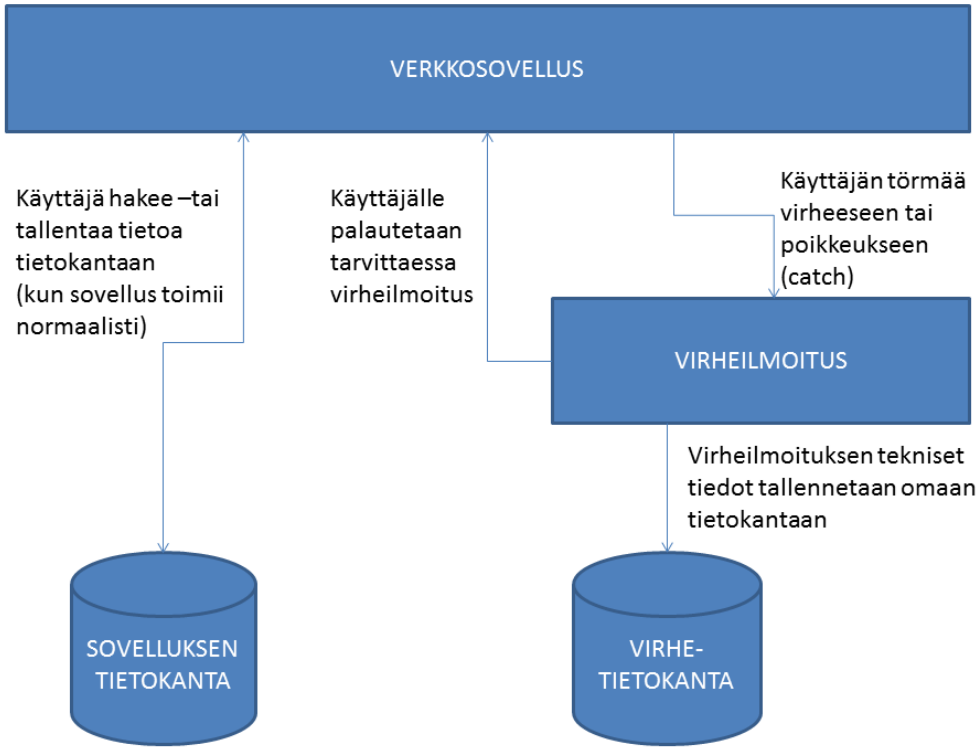
tapahtuisi sisäänkirjaamisessa tai missä tahansa muussa sovelluksen toiminnassa emme voi kirjata tietokantaan käyttäjien salasanoja.

Automaattisten virheilmoitusten keräämä tieto sinänsä on suhteellisen helppo määritellä ja kerätä, mutta toinen osa ongelmaa on se miten ja mitä tietoa käyttäjien tekemistä virheilmoituksista kerätään. Kappaleessa 3 mainitsin muutaman teknologian jota voidaan soveltaa käyttäjäpalautteen keräämiseen ja näistä vaihtoehdoista valitsimme Zopimin toteutettavaksi ratkaisuksi.

4.4 VIRHETIETOKANTA

Sovelluksen odotettu käyttäjäkunta tulee olemaan laaja. Tarkoituksenamme on kirjata ylös tietokantaan sekä virheet, varoitukset, että ilmoitukset. Tämä tarkoittaa sitä että tietokantaan tulee tapahtumia todennäköisesti paljon. Tässä riskinä on se että liiketoimintaan ja ohjelman toimintaan suoraan tarkoitettu tietokanta tulisi ylikuormittumaan hauista ja tallennuksista joilla ei ohjelman toimintaan ole suoraa merkitystä. Tästä syystä on järkevää tehdä virheilmoituksille oma ja täysin uusi tietokanta.

Tätä uutta tietokantaa puoltaa myös se, että sovellukseen ollaan myöhemmin vuoden 2013 lopun ja 2014 alun välillä tekemässä erillinen järjestelmänvalvojan käyttöliittymä ja siihen liittyvät toiminnot. Näihin toimintoihin kuuluu monia muita toimintoja kuin pelkästään virheilmoitusten hallinta, joten yritys tulee jokatapauksessa tarvitsemaan uutta täysin erillistä tietokantaa (eKeiretsu Oy 2013b). Näin verkkosovellus pystyy toimimaan täysin omassa ympäristössään ilman että järjestelmänvalvojan toiminnot rasittaisivat ja hidastaisivat sen toimintaa. Kuviossa 13 on kuvattuna erittäin yksinkertaistetusti virhetietokannan tarkoitus.



KUVIO 13. Virhetietokannan ja sovelluksen tietokannan ero.

Kun verkkosovelluksessa tehdään hakuja tai tallennuksia niin nämä toiminnot tehdään sovelluksen tietokantaan. Ainoastaan mikäli verkkosovelluksessa tapahtuu virhe käydään kuvan virheilmoitus haarassa, josta lopulta tieto tallennetaan virhetietokantaan. Näin vältytään sovelluksen tietokannan ylimääräiseltä kuormittamiselta.

5 YHTEENVETO

Sovelluksen virheilmoitusten tehokas raportointi on tärkeä, mutta todennäköisesti hieman ylenkatsottu osa verkkosovelluksen julkaisua ja käyttöä. Huonolla poikkeusten käsittelyllä ja niistä seuraavilla virheilmoituksella voidaan monella tapaa aiheuttaa haittaa sekä sovelluksen käyttäjille, että itse yritykselle joka sovellusta kehittää. Sovellus joka antaa käyttäjälle usein epämääräisiltä tuntuvia ilmoituksia virheistä muodostaa sovelluksesta yrityksen kannalta epäedullisen kuvan. Yritykselle puolestaan huonosti toteutettu poikkeusten käsittely voi aiheuttaa todellista liiketoiminnallista haittaa. Virheilmoituksissa saatetaan näyttää luottamuksellista tietoa sovelluksen toiminnasta ja sen metodeista, luokista, muuttujista ja muista haavoittuvuuksista. Käyttäjä voi halutessaan ja osatessaan käyttää näitä tietoturva-aukkoja hyökkäyksiin sovellusta kohtaan mikä voi johtaa todellisiin liiketoiminnalisiin haittoihin kuten palvelunestoon tai käyttäjien tietojen ja salasanojen kalasteluun. Huono virheiden käsittely voi myös aiheuttaa sovelluksen ohjelmoijalle tai ohjelmoijille ylimäärästä työtä ja viivästyttää sivun tai sovelluksen valmistumista kun vihreiden syitä joudutaan etsimään tarkemmin.

Virheilmoitusten raportointijärjestelmän toteutukseen on monia erilaisia vaihtoehtoja joista ei voida poimia yhtä oikeaa ratkaisua joka sopisi kaikkiin tilanteisiin. Kohdeyrityksessä ja opinnäytetyössäni lopullisen suunnan suunnitelmalleni antoi se, että sovellus on kehitetty PHP:llä ja sen Zend Framework 2 ohjelmistokehyksellä. Mikäli käytössä olisi esimerkiksi pelkkä PHP ilman erillistä ohjelmistokehystä tai jokin toinen ohjelmistokehys Zendin tilalla niin ratkaisu johon päädyin olisi ollut lähdekoodia ajatellen erilainen, mutta silti todennökäisesti toiminnallisesti samankaltainen. Uskon siis että löytämääni ratkaisua voidaan käyttää pohjana virheilmoitusten raportointijärjestelmän suunnitteluun myös yleisesti vaikka välttämättä käsitellyt teknologiat eivät vastaisi jonkin toisen sovelluksen teknologioita.

Tässä työssä päätutkimusongelmana oli selvittää miten PHP:ssä toteutetaan virheilmoitusten raportointi järjestelmä? Järjestelmän voi toteuttaa joko ennakoiden estämällä funktioiden suoritus virheellisellä syötteellä, jolloin ilmoitus virheestä annetaan ennen kuin virhe tapahtuu, tai niin että virheiden annetaan tapahtua ja järjestelmä raportoi tapahtuneita virheitä. Päädyin siihen

lopputulokseen, että yleisesti ottaen tutkimusongelmaa kannattaa lähestyä siitä näkökulmasta, että virheiden annetaan tapahtua hallitusti try ja catch lohkojen sisällä. Tämän lähestymistavan yksi selkeä hyöty on siinä, että kaikki tarvittava tieto virheistä välittyy varmasti sovelluksen kehittäjälle. Toisessa lähestymistavassa välttämättä tieto ei välity kun funktiota ei edes anneta suorittaa virheeseen asti.

Alaongelmina tutkittiin lisäksi miten Zend Framework 2:n käyttö vaikuttaa järjestelmän toteutukseen ja miten järjestelmään kirjataan käyttäjien löytämät virheet? Zend Framework 2:n käyttö PHP:n yhteydessä mahdollistaa sen, että kaikkia sovelluksen virheitä voidaan käsitellä yhdessä paikassa. Tällöin päätutkimuskysymyksessä löydettyä ratkaisua ei tarvitse jakaa eripuolille sovellusta, vaan sovelluksen rakenne pysyy loogisena ja selkeänä. Käyttäjille puolestaan on järkevää antaa mahdollisuus lähettää omia virheilmoituksia mm. Zopimin kaltaista käyttäjäpalaute chattia hyödyntäen. Tällöin yrityksen edustaja voi olla suoraan yhteydessä käyttäjään ja tarvittaessa pyytämään lisäinformaatiota virheestä, sekä omasta puolestaan neuvomaan käyttäjää sovelluksen käytössä. Tämän kaltainen asiakaspalvelu on tärkeää hyvien käyttäjäkokemuksien tarjoamiseksi.

Yleisenä nyrkkisääntönä virheilmoitusten ja poikkeusten käsittelystä voidaan siis todeta, että suurin hyöty sovelluksen kehittäjille saadaan silloin kun tietoa virheilmoituksista kerätään mahdollisimman yksityiskohtaisesti ja useasta eri lähteestä. Täytyy kuitenkin huomioida, että tämä aiheuttaa tiettyjä lisävaatimuksia tietokannalle joten tiedon tallentaminen täytyy suunnitella huolellisesti. Käyttäjälle puolestaan ei tule koskaan näyttää virheilmoituksen sisältöä vaan jokin erillinen pahoittelu viesti ja vaikkapa linkki takaisin etusivulle. Tämä parantaa sovelluksen käytettävyyttä, sekä luo verkkosovelluksesta tietoturvallisemman.

LÄHTEET

Allen, R. 2013. Simple logging of ZF2 errors.[Viitattu 3.10.2013]. Akrabat.com Saatavissa: http://akrabat.com/zend-framework-2/simple-logging-of-zf2-exceptions/

Allen, R., Lo, N. & Brown, S. 2009. Zend Framework in Action. Shelter Island: Manning Publications.

eKeiretsu Oy. 2013a. Pääinsinööri. Ekeiretsu Oy. Haastattelu 1.10.2013

eKeiretsu Oy. 2013b. Pääinsinööri. Ekeiretsu Oy. Haastattelu 9.10.2013

Garret, J.J. 2005. AJAX: A New Approach to Web Applications. [Viitattu 16.10.2013]. Adaptive Path. Saatavissa: http://experiencezen.com/wp-content/uploads/2007/04/adaptive-path-ajax-a-new-approach-to-web-applications1.pdf

Ghai, R. & Sagar, V. 2009. What are the different types of errors in PHP? [Viitattu 9.10.2013]. CareerRide. Saatavissa:http://www.careerride.com/PHP-types-errors.aspx

Google. 2013. GOOGLE ANALYTICS TERMS OF SERVICE. [Viitattu 25.10.2013]. Google. Saatavissa: http://www.google.com/analytics/terms/us.html

Hevner, A.R., March, S.T., Park, J. & Ram, S. 2004. Design Science in Information Systems Research. MIS Quarterly. vol. 28 no. 1, pp. 75-105.

Lieberman, H & Fry, C. 2001. Will Software ever work?.Communications of the ACM 44.3.1-3. Saatavissa: http://web.media.mit.edu/~lieber/Publications/Software_Work.pdf

Microsoft. 2013. HTTP Cookies. [Viitattu 22.10.2013]. Microsoft. Saatavissa: http://msdn.microsoft.com/en-us/library/windows/desktop/aa384321(v=vs.85).aspx

Mynttinen, T. 2009. Poikkeustenkäsittelylohkot. [Viitattu 21.10.2013]. Mikkelin ammattikorkeakoulu. Saatavissa: http://cna.mikkeliamk.fi/Public/MynttinenTimo/Internet%20sovelluskehitys/PHP-perusteet/Poikkeusten%20k%C3%A4sittely/

Oracle. 2013. What is MySQL? [Viitattu 1.10.2013]. MySQL.com. Saatavissa: http://dev.mysql.com/doc/refman/4.1/en/what-is-mysql.html

PHP The Right Way. 2013. Errors and Exceptions. [Viitattu 2.10.2013]. PHP The Right Way. Saatavissa: http://www.phptherightway.com/#errors_and_exceptions

Schlossnagle, G. 2003. Advanced PHP Programming.Indianapolis: Sams Publishing.

The PHP Group. 2013a. Exceptions. [Viitattu 30.9.2013]. PHP.net. Saatavissa:http://php.net/manual/en/language.exceptions.php

The PHP Group. 2013b. Error Handling: Predefined Constants. [Viitattu 5.10.2013]. PHP.net. Saatavissa:http://us2.php.net/manual/en/errorfunc.constants.php#errorfunc.constants.errorlevels.e-error

The PHP Group. 2013c. Error Reporting. [Viitattu 13.10.2013]. PHP.net. Saatavissa:http://php.net/manual/en/security.errors.php

The PHP Group. 2013d. What is PHP? [Viitattu 29.9.2013]. PHP.net. Saatavissa:http://php.net/manual/en/intro-whatis.php

The PHP Group. 2013e. What can PHP do? [Viitattu 29.9.2013]. PHP.net. Saatavissa:http://php.net/manual/en/intro-whatcando.php

The PHP Group. 2013f. Unsupported historical releases. [Viitattu 9.10.2013]. PHP.net.Saatavissa:http://php.net/releases/index.php

The PHP Group. 2013g. file_put_contents. [Viitattu 18.10.2013]. PHP.net. Saatavissa:http://php.net/manual/en/function.file-put-contents.php

Yaiser, M. 2011. Object-oriented programming concepts: Objects and classes. [Viitattu 24.10.2013]. Adobe. Saatavissa:http://www.adobe.com/devnet/actionscript/learning/oop-concepts/objects-and-classes.html

Zend Technologies Ltd. 2013. Introduction to the MVC Layer. [Viitattu 16.10.2013]. Zend Technologies Ltd. Saatavissa: http://framework.zend.com/manual/2.1/en/modules/zend.mvc.intro.html

Zopim. 2013. Zopim Product Features. [Viitattu 10.10.2013]. Zopim. Saatavissa: https://www.zopim.com/product