OULUN AMMATTIKORKEAKOULU

Timo Asumaniemi

VERKKOHALLINTASOVELLUKSEN SUUNNITTELU JA TOTEUTUS MYPOSE OY:LLE

VERKKOHALLINTASOVELLUKSEN SUUNNITTELU JA TOTEUTUS MYPOSE OY:LLE

Timo Asumaniemi
Opinnäytetyö
Syksy 2014
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikka, Ohjelmistokehitys

Tekijä(t): Timo Asumaniemi
Opinnäytetyön nimi: Verkkohallintasovelluksen suunnittelu ja toteutus Mypose Oy:lle
Työn ohjaaja: Juha Alakärppä
Työn valmistumislukukausi- ja vuosi: Syksy 2014 Sivumäärä: 45 + 1

Opinnäytetyön aiheena oli suunnitella ja toteuttaa verkkohallintasovellus Mypose Oy:lle. Pohjana työlle oli entinen verkkohallinta, jonka rakenteellisuus todettiin projektin aloitusvaiheessa hieman kankeaksi jatkokehityksen kannalta. Työn tavoitteena oli saada vastaavat toiminnollisuudet uudelle verkkohallinnalle, joita projektin aloitushetkellä oli edellisessä verkkohallinnassa. Uuden verkkohallinnan tuli olla sellainen, että uusien ominaisuuksien lisääminen tai muokkaaminen tapahtuisi huomattavasti nopeammin ja vaivattomammin. Verkkohallinnan avulla mm. muokataan Mypose-laitteiden sisältöä ja luetaan statistiikka. Työ tuli siis välittömästi todelliseen käyttöön yrityksessä.

Projektin apuna käytin Tinypm-työkalua projektinhallintaan. Totetus tapahtui Linux-ympäristössä käyttäen Netbeans- ja Qt-ohjelmointiympäristöjä. Tietokantojen suunnittelussa käytettiin MySQL-Workbench ohjelmistoa.

Projekti onnistui kokonaisuudessaan hyvin ja jatkokehitys projektin parissa lähti etenemään opinnäytetyön päätyttyä vauhdikkaasti. Tässä raportissa käydään läpi pääpiirteittäin verkkohallinnan toimintaperiaatteet. Liikesalaisuussyistä toteutuksen esittelyssä ei käydä läpi tarkkaa lähdekoodia, vaan lyhyillä ohjelmakoodiesimerkeillä havainnollistetaan käytettyjä teknologioita ja menetelmiä.

Asiasanat: verkkohallinta, verkkopalvelu, HTML, PHP, jQuery, JavaScript, Ajax

ABSTRACT

Oulu University of Applied Sciences
Software development

Author(s): Timo Asumaniemi
Title of thesis: Design and implementation of web management service for Mypose Oy
Supervisor(s): Juha Alakärppä
Term and year when the thesis was submitted: Autumun 2014 Number of pages: 45 + 1

Objective for this thesis was to design and implement new web management service for Mypose Oy. Starting point for this project was the old web management, which was a bit too complicated to support new feature requirements for Mypose devices. The goal for the new web management was that it should have the same basic functionality than the old one. Customizing and applying new features to new web management should happen easily in the future. Also the new web management should give foundations to support new upcoming features for Mypose devices. This work was adapted to real usage right away.

For project management I used Tinypm-software. For the programming I used Netbeans and Qt integrated development environments. For designing databases, I used MySQL Workbench.

Project succeeded very well and the development continues fast at Mypose. This document gives general information about functionality of the new web management. For trade secret reasons I won't display actual source code at any point. I show short program examples to show some methods and technologies I used in development.

Keywords: Webmanagement, Web service, Database, PHP, jQuery, JavaScript, HTML

ALKULAUSE

Haluan kiittää koko Mypose Oy:n henkilökuntaa luottamuksesta ja hyvästä kannustuksesta projektin parissa. Erityiset kiitokset toimitusjohtaja Lassi Anttoselle, jonka kannustava asenne ja hyvä palaute auttoivat merkittävästi projektin edistymisessä. Iso lisäkiitos kuuluu myös kollegalleni Juho Juntuselle, jonka avustuksella selvisin monesta ohjelmointiongelmasta nopeasti. Lisäksi opinnäytetyön ohjaajani Juha Alakärppä auttoi kannustavasti kehittämään insinöörin taitojani.

Oulussa 27.11.2014

Timo Asumaniemi

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena oli suunnitella ja toteuttaa uusi verkkohallintasovellus Mypose Oy:lle. Työhön kuuluivat pääpiirteittään uusien vaatimusten määrittely, uuden verkkohallinnan suunnittelu ja ohjelmointi sekä testaus. Tarve projektille oli hyvin selvä heti projektin alkuvaiheessa. Vaikka yrityksen entinen verkkohallinta toimi sinällään erinomaisesti, sen muokkaaminen, laajentaminen ja uusien ominaisuuksien lisääminen osoittautui jo projektin esitutkimusvaiheessa hieman sekavaksi ja aikaa vieväksi. Muun muassa tästä johtuen tilaajan kanssa päätettiin, että lähden suunnittelemaan uuden verkkohallinnan lähes puhtaalta pöydältä.

Lisähaastetta projektille asetti se, että ennen uuden verkkohallinnan toteutusta täytyi uusi palvelin konfiguroida ja alustaa projektia varten. Lisäksi entinen tietokantarakenne ei sopinut sellaisenaan uuden verkkohallinnan ominaisuuksille, joten myös tietokantojen suunnittelu ja toteutus kuuluivat projektiin. Projektin aloituspäätöksestä lähtien on Mypose-sovellukselle tullut jatkuvasti uusia ominaisuuksia, jotka täytyi ottaa huomioon myös tietokantoja suunnitellessa. Monipuolisen tietokannan pohjalta on mahdollista tehdä verkkohallinnan puolelle käytännöllisiä työkaluja, joiden avulla voidaan helpottaa niin ohjelmistosuunnittelijoiden, myyjien kuin asiakkaittenkin työskentelyä Mypose-laitteiden parissa. Uusi verkkohallinta ei siis enää ole vain Myposen henkilökunnan työkalu, vaan myös sen asiakkaiden työkalu.

Verkkohallinta on todella laaja kokonaisuus ja sitä jatkokehitetään yrityksessä varmasti vielä pitkän aikaa. Tärkeää oli kuitenkin tehdä linjaveto, missä vaiheessa opinnäytetyön osalta toteutus on riittävän pitkällä. Tärkeimpänä vaatimuksena oli saada ne ominaisuudet toimintaan uudessa verkkohallinnassa, jotka olivat vanhassakin verkkohallinnassa. Vaikka ominaisuudet päällepäin tällä hetkellä vastaavatkin entistä verkkohallintaa, rakenteellisten muutosten takia suurin ero löytyy verkkohallinnan taustan eli ns. back-endin puolelta.

Tämän dokumentin nimeämiskäytännöstä on hyvä esitellä muutama asia, sillä muuten jotkin nimitykset saattavat sekoittaa lukijaa. Kun dokumentissa mainitaan *Mypose*, *Mypose Oy* tai *Mypose*-yritys, tarkoitetaan näillä yritystä. Kun taas puhutaan *Mypose-laitteesta*, tarkoitetaan sillä yrityksen toimitettavaa laitetta. *Mypose-sovellus* taas tarkoittaa puolestaan *Mypose-laitteella* olevaa sovellusta, jota loppukäyttäjä varsinaisesti käyttää *Mypose-laitteella*. *Verkkohallintasovellus* ja *verkkohallinta* tarkoittavat isoa kokonaisuutta, verkkosovellusta, joka koostuu useasta komponentista.

Esitellyt ohjelmakoodit eivät sisällä liikesalaisuus- ja tietoturvasyistä tarkkoja kopioita varsinaisesta lähdekoodista. Sen sijaan olen tehnyt omalle palvelimelleni lukuisia lyhyitä esimerkkisovelluksia, jotka antavat hyvän kuvan, millä lailla datan käsittely verkkohallinnassa toimii.

2 MYPOSE-LAITTEEN TOIMINTAPERIAATE

Mypose-laite on nykyaikainen valokuvauslaite, jonka avulla käyttäjä voi ottaa itsestään kuvia ja jakaa niitä sosiaaliseen mediaan tai lähettää sähköpostilla kavereilleen. Käyttäjä voi myös koristaa kuvansa hauskoilla lisäelementeillä. Henkilö, joka käyttää laitetta, on ns. loppukäyttäjä. Asiakas, joka vuokraa Mypose-laitteen yritykselleen tai vaikkapa johonkin tapahtumaan, voi rakentaa kuvan päälle erilaisia markkinointikampanjoita. Kun loppukäyttäjä jakaa omalle Facebook-sivulleen kuvansa, jossa on vaikkapa jonkun tunnetun brändin logo tai mainos, tulee yritykselle tai brändille näkyvyyttä todella paljon. Kuvassa 1 on esimerkki Mypose-laitteen käytöstä.

KUVA 1. Mypose-laite käytössä

Yrityksillä on myös mahdollisuus sisällyttää kuvaan markkinointiviesti. On hyvin selvää, että markkinointimateriaalia eri kampanjoille pitää pystyä vaihtamaan nopeasti ja helposti. Materiaalin vaihto ja kampanjan päivittäminen pitäisi onnistua myös laitteen vuokraajalta itseltään. Lisäksi Mypose-sovelluksen ominaisuuksia täytyy pystyä laittamaan päälle ja pois ilman laitteen tai sovelluksen uudelleenkäynnistämistä. Esimerkiksi jossakin tapahtumassa laitteen vuokraaja voi haluta, että kuvan voi jakaa ainoastaan Facebookin kautta. Näitä ominaisuuksia voi muokata ja lisätä verkkohallintasovelluksen kautta. Verkkohallinnan näkyvä puoli on web-sivu, johon käyttäjä kirjautuu ja voi

esimerkiksi vaihtaa kampanjakuvia, jotka lisätään varsinaisen kuvan päälle, tai muuttaa sähköpostiin liitettävän markkinointiviestin sisältöä. Lisäksi verkkohallinnan sivulla käyttäjä voi lukea laitteen statistiikkaa, esimerkiksi tarkastaa, montako kuvaa on jaettu Facebookiin kampanjan aikana. Kuvassa 2 on esimerkkinäkymä Mypose-laitteen kampanjamateriaalista.

KUVA 2. Esimerkki Mypose-laitteen kampanjamateriaalista

3 VERKKOHALLINNAN KOMPONENTIT

Verkkohallinta kokonaisuudessaan koostuu oikeastaan kolmesta pääelementistä: palvelin, käyttäjän laite sekä Mypose-laite. Käyttäjän laitteella tarkoitetaan lähinnä käyttäjän tietokonetta ja sen selainta, ja se kuvastaa siten rajapintaa käyttäjän ja verkkohallintasovelluksen välillä. Palvelimella tarkoitetaan tässä kohtaa karkeasti kaikkia komponentteja, jotka sijaitsevat palvelimella ja muodostavat ytimen verkkohallinnalle. Ensimmäisessä osassa käydään läpi verkkohallinnan rakenteellista osuutta ja sitä, miten nämä palvelimella olevat osat kommunikoivat keskenään. Nämä osat muodostavat verkkohallinnan taustan. Tausta tarkoittaa yleisesti jonkin sovelluksen tai kokonaisuuden ytimessä olevia komponentteja, jotka eivät ole loppukäyttäjälle näkyvissä. Edusta, eli front-end, on tässä tapauksessa verkkohallinnan web-sivu, joka on nimenomaan loppukäyttäjälle se näkyvissä oleva puoli ja muodostaa käyttöliittymän.

3.1 Verkkohallinnan arkkitehtuuri

3.1.1 Palvelin

Palvelin on fyysinen laite, joka pitää sisällään verkkohallinnan kaiken toiminnollisuuden. Palvelimella sijaitsevat kaikki data ja sitä käyttävät palvelut, sekä tietenkin verkkohallinnan web-sivu. Mypose-laitteet keskustelevat palvelimen kanssa verkkohallinnan kautta. Turvallisuussyistä verkkohallinnasta ei suoraan voida muuttaa Mypose-laitteiden sisältöä. Verkkohallinnan kautta muokataan tai lisätään dataa, joka tallennetaan palvelimelle, ja Mypose laitteet hakevat itse päivitetyn datan suoraan palvelimelta. Datan lisäksi palvelimen, verkkohallinnan ja Mypose-sovelluksen välillä liikkuu mm. kuvia ja konfiguraatiotiedostoja. Ennen kuin varsinaista verkkohallintasovellusta pystyttiin toteuttamaan, oli palvelin muokattava sellaiseksi että se vastasi verkkohallinnan vaatimusmäärittelyjä. Uusi verkkohallinta tuli siis käyttöön uudelle palvelimelle, jolloin täytyi varmistaa, että palvelimelta löytyivät tarvittavat sovellukset ja kirjastot, sekä tarvittaessa asentaa puuttuvat osat.

Käytössä oleva palvelin toimii Linux-ympäristössä. Linux-käyttöjärjestelmän käyttäminen palvelimissa on todella suosittua sen luotettavuuden takia. Palvelimen konfigurointiin löytyy todella paljon hyviä teoksia, joista Linux Bible 2010 Edition (Negus 2009) osoittautui todella hyväksi käsikirjaksi tässäkin projektissa.

3.1.2 Tietokanta

Olemassa olevat tietokannat kaipasivat aika paljon rakenteellisia muutoksia, joten tietokantataulut suunniteltiin uusiksi ja niitä tehtiin jonkin verran lisää. Lisäksi tietokannan rakenteellisuus muutettiin relaatiotietokannaksi. Ympäristössä, jossa laitteiden hallinta ja statistiikan keruu koski alle 20:tä laitetta, entinen tietokanta-arkkitehtuuri palveli mainiosti. Liiketoiminnan jatkuvan kasvun, sekä lisääntyvien laitemäärien myötä tilaajan kanssa päätettiin päivittää myös tietokantojen perustoiminnot. Eritoten huomiota täytyi kiinnittää siihen, että uutta verkkohallintaa tulisivat käyttämään asiakkaat entistä enemmän myyjien tai muun henkilökunnan sijaan.

Sujuva ja mahdollisimman helppo verkkohallinnan käyttö tarvitsee rinnalleen mahdollisimman kattavan ja järkevästi toimivan tietokannan. Tietokannan muuttaminen relaatiotietokannaksi antaa hyvän pohjan tulevaisuudelle, sillä tässä vaiheessa ei vielä täysin ole tiedossa, minkälaisia työkaluja verkkohallintaan täytyy lisätä. Kun verkkohallintaan tarvitaan työkalu, jolla voidaan esimerkiksi listata asiakkaan käytössä olleet laitteet tiettynä ajanjaksona, on sellainen huomattavasti helpompaa ja nopeampaa toteuttaa, jos tietokanta on valmiiksi hyvin linkitetty esimerkiksi asiakastunnusten, laitesopimusnumeroiden ja statistiikkataulujen välillä.

Koska verkkohallintasovelluksen ohjelmointikielenä on PHP, oli luonnollista käyttää sen tarjoamaa rajapintaa tietokannan käsittelyyn. Tietokannan käsittelyyn on olemassa lukuisa eri tapoja ja oikean ratkaisun löytäminen vaatii kokemusta. Tässä projektissa ratkaisujen löytämisten helpottamiseksi käytettiin mm. Expert PHP and MySQL teosta (Rochkind 2013).

3.1.3 Verkkohallinnan sisäiset pienoissovellukset

Koska palvelimella olevia tietokantoja käyttävät verkkohallinta, Mypose-sovellukset ja myös muut PHP-sovellukset, tehtiin tulevaa ohjelmointi- ja suunnittelutyötä helpottamaan PHP-pienoissovel-

lus. Pienoissovellus toimii eräänlaisena rajapintana tietokantakyselyjen ja niitä kutsuvan sovelluksen välillä. Tämä rajapinta oli täysin oma-aloitteisesti tehty ja suunniteltu lisäsovellus vaatimusmäärittelyjen päälle. Koska tiedossa oli, että verkkohallintaa tullaan muokkaamaan ja jatkokehittämään tulevaisuudessa paljon, on perusteltua paloitella verkkohallinnan toiminnollisuuksia helpommin käsiteltäviin osiin. Samalla tietokannan käsittelyn tärkeimmät osat saadaan suojattua.

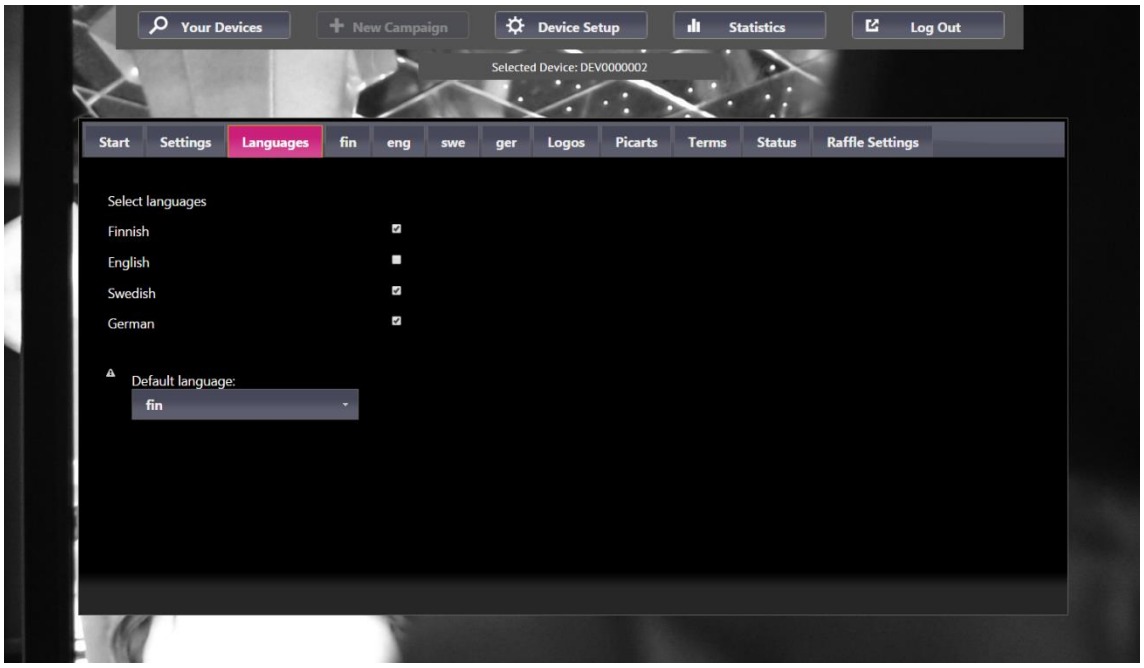
Esimerkkinä rajapinnan toiminnasta voisi olla hyvinkin tyypillinen tilanne, jossa rajapintaa kutsuva sovellus tarvitsee jonkin laitteen senhetkisen kampanjan kampanjatunnuksen. Koska tällaista tyypillistä tietoa tarvitsee niin verkkohallinta kuin Mypose-sovelluskin, löytyy rajapinnasta funktio, joka välittää kutsun tietokantoja käsittelevälle luokalle. Tämä luokka puolestaan palauttaa datan takaisin rajapinnalle ja rajapinta taas palauttaa datan sitä kutsuneelle sovellukselle. Koska rajapinnan tätä ominaisuutta kutsutaan HTTP-protokollan POST-pyyntöjen avulla, voivat rajapintaa käyttää lukuisat eri alustoilla toimivat sovellukset. Tällaista sovellusta kutsutaan monesti myös englanninkielisellä termillä web service eli verkkopalvelu. PHP-pohjaiset verkkopalvelut ovat hyvin yleinen tapa, kun halutaan päästä käsiksi palvelimella sijaitsevaan dataan. Oikean tavan löytämiseksi käytettiin apuna PHP-verkkopalveluista kertovaa teosta PHP Web Services (Mitchell, PHP Web Services 2013).

Tietokantojen lisäksi oleellinen muokattava data on konfiguraatiotiedostot. Koska konfiguraatiotiedostojen sisältö saattaa tulevaisuudessa muuttua hyvinkin paljon, niiden käsittelyä varten täytyi keksiä jotakin uutta. Jos ajatellaan tilannetta, jossa Mypose-sovellukseen on toteutettu uusi ominaisuus, esimerkiksi mahdollisuus jakaa kuva Twitteriin, tulisi tämän ominaisuuden tarvittavien komponenttien olla muokattavissa tietyssä konfiguraatiotiedostossa. Lisäksi nämä tarvittavat komponentit pitäisi lisätä itse konfiguraatiotiedostoon, jotta niiden muokkaaminen olisi edes mahdollista. Web-sivun pitäisi toisaalta pysyä automaattisesti mukana näissä vaihdoksissa. Edellinen verkkohallinta oli tällaisissa tapauksissa hyvinkin kankea, sillä pienenkin asian lisääminen vaati muutoksia moneen paikkaan, jolloin prosessista tuli aikaa vievää ja virheherkkää. Asia ratkaistiin sillä, että konfiguraatiotiedostojen käsittelyyn tehtiin myös oma PHP-pienoissovellus, joka rakenteellisesti toimii samankaltaisesti kuin tietokantoja käsittelevä PHP-rajapintasovelluskin. Tälle pienoissovellukselle lähetetään samaan tapaan POST-pyyntöjä ja sovellus palauttaa sitten tarvittaessa dataa tai kuittauksen prosessin onnistumisesta.

3.1.4 Verkkohallinnan käyttöliittymä

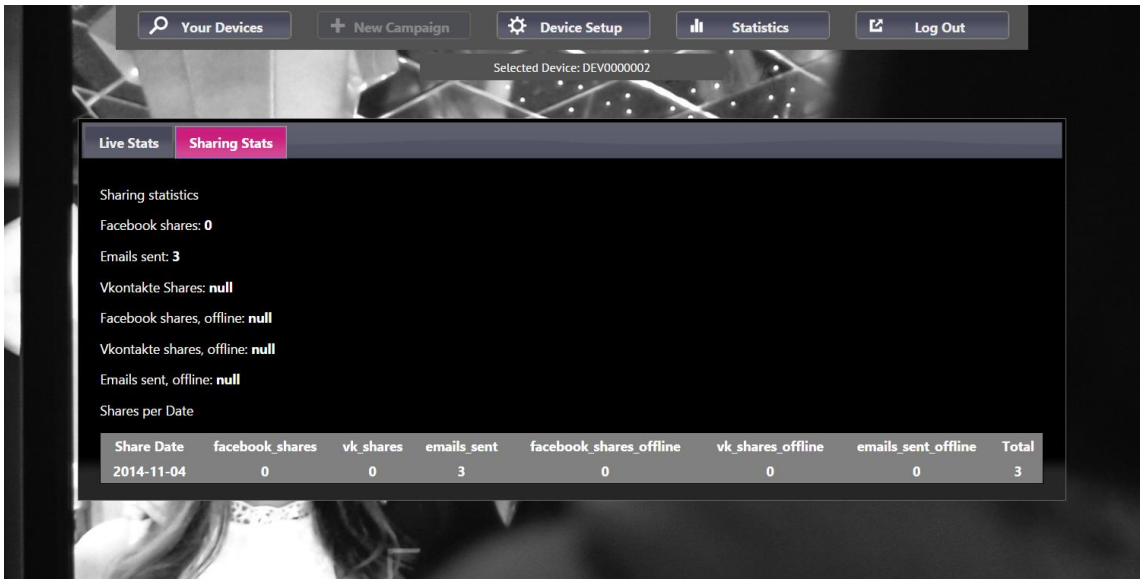
Verkkohallinnan käyttäjätilit jakautuvat tällä hetkellä kahteen päätasoon, admin- ja asiakastasoon. Vaikka vaatimusmäärittelyn mukaan verkkohallintaa varten tarvittiin kaksi tasoa, on kaikki palvelinpuolen toteutukset suunniteltava siten, että eri tasoja voidaan tulevaisuudessa lisätä tai poistaa helposti. Esimerkiksi hyvin kattavan laitehallinnan tarjoava kehittäjä-taso on tulevaisuudessa hyvinkin tarpeellinen.

Käyttöliittymä on toteutettu HTML-kielellä, joka käyttää apunaan CSS-tyylitiedostoja, JavaScript-tiedostoja, jQueryä, Ajax-kutsuja jne. Käyttöliittymä koostuu kahdesta pääkomponentista, työkalupalkista ja välilehdillä varustetusta sisältöikkunasta. Kuvassa 3 on esimerkki, jossa käyttäjä on muokkaamassa kielivalintaa.



KUVA 3. Kielivalinnan muokkaus

Työkalupalkista löytyy tällä hetkellä mahdollisuudet valita laite, muuttaa laitteen asetuksia, luoda uusi kampanja tai lukea statistiikkaa. Kuvassa 4 käyttäjä on tarkastelemassa statistiikkaa.



KUVA 4. Statistiikan luku verkkohallinnasta

3.3 Admin-taso

Admin-tason tili on tarkoitettu ensisijaisesti henkilökunnan käyttöön. Se sisältää laajimmat hallintamahdollisuudet sekä pääsyn kaikkiin Mypose-laitteisiin. Admin-taso on luonnollisesti toteutukseltaan monimutkaisempi ja vaatii huomattavasti enemmän aikaa niin toteutuksen kuin myös testauksen osalta. Tässä opinnäytetyössä riitti, että admin-taso eroaa asiakastasosta sillä, että admintasolla pääsee käsiksi kaikkiin laitteisiin. Jatkokehityksen alla on työkalujen ohjelmointi admin-tasolle. Tällainen työkalu voisi olla esimerkiksi konfiguraatiotiedoston tekstieditori tai vaikkapa käyttäjätilien laajempi hallinta.

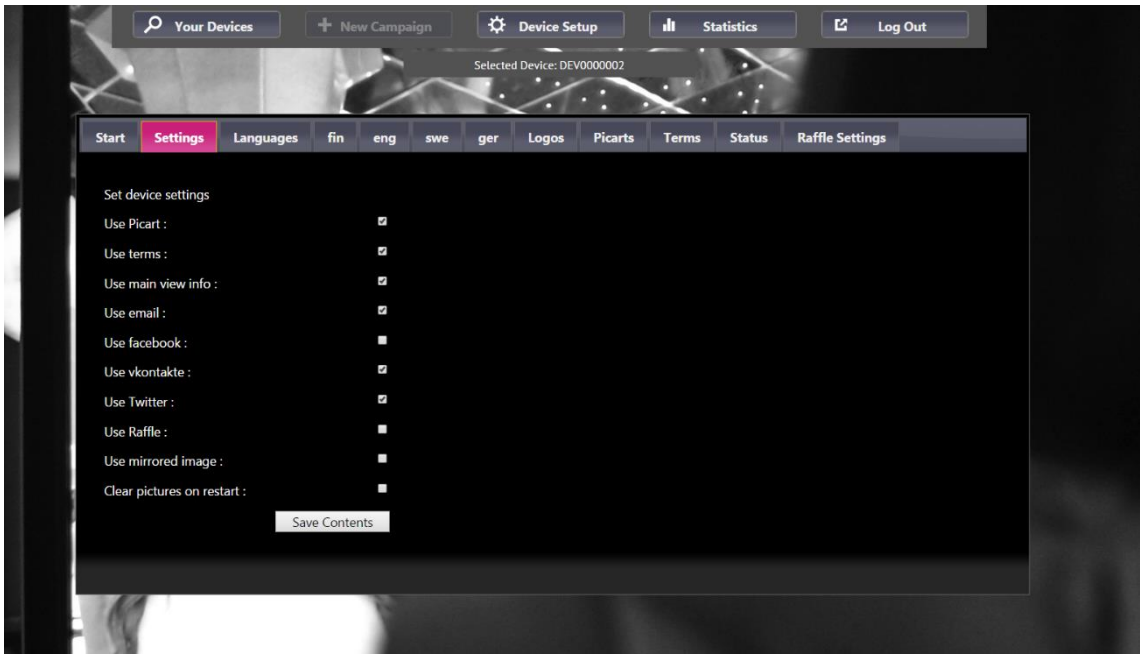
3.4 Asiakastaso

Asiakastaso on tarkoitettu yhden tai useamman laitteen vuokranneelle asiakkaalle. Kun asiakas on kirjautunut sivulleen, on hänen mahdollista tarkastella vain ja ainoastaan hänen vuokrasopimuksissaan olevia laitteita. Edellisessä verkkohallinnassa saattoi tulla tilanne vastaan, jossa asiakas pystyi tarkastelemaan laitteen edellisen käyttäjän statistiikkaa. Varsinaista haittaahan ei tästä ollut, mutta tämä täytyi muuttaa uuteen verkkohallintaan. Uuden verkkohallinnan myötä asiakas näkee statistiikan hänelle kuuluvan laitteen, vain hänen itsensä luoman kampanjan ajalta. Lisäksi ominaisuus, jolla käyttäjä pystyy tarkastelemaan esimerkiksi aikaisemmin vuokraamiensa laitteiden statistiikkaa, toteutettiin projektin loppupuolella. Tämä ominaisuus on käyttöliittymää vailla valmis.

Tämä ominaisuus tulee olemaan todella tärkeä, sillä tällä hetkellä Mypose-laitteita on tullut paljon lisää ja ne liikkuvat nopeasti paikasta toiseen. Jälkeenpäin statistiikan tarkastelu kampanjoittain on todella huomattava parannus edellisen statistiikan keruun päälle.

Statistiikan lukemisen lisäksi asiakas pystyy luomaan uuden kampanjan laitteelleen verkkohallinnasta. Kampanjan luontiin liittyvät toiminnot on uudessa verkkohallinnassa paloiteltu pienempiin osiin, jolloin verkkohallinnan käyttäminen on huomattavasti helpompaa ja selkeämpää. Uusia ominaisuuksia on tullut niin paljon, että paloittelu on todella tarpeellinen. Nyt asiakas voi mm. määritellä tarkemman sijainnin laitteelle, antaa nimen kampanjalle, määritellä kampanjan keston jne.

Pienenä parannuksena entiseen verkkohallintaan nähden on myös se, että uuden verkkohallinnan web-sivun sisältö muokkautuu automaattisesti sen mukaan, mitä komponentteja asiakas valitsee kampanjan muokkaamisen yhteydessä. Esimerkiksi jos asiakas valitsee, että laitteella käytetään vain suomen kieltä, ei kampanjan muokkaustyökalussa ole esimerkiksi tekstikenttiä sähköpostin sisällön muokkaamiseksi muille kuin suomen kielelle. Mikäli asiakas haluaa, ettei esimerkiksi sähköpostilähetystä oteta käyttöön, piilotetaan silloin näkyvistä sähköpostin sisällön muokkaamiseen liittyvät komponentit. Kuvassa 5 on esimerkki, miten komponenttien asetusten vaihtaminen tapahtuu verkkohallinnasta.



KUVA 5. Komponenttien valinta

Kun web-sivu osaa muokkautua tällä tavalla tarpeen mukaan, on tulevaisuudessa esimerkiksi uusien ominaisuuksien käyttöönotto huomattavasti nopeampaa kokonaisuudessaan, kun niiden vaatimia pieniä lisäyksiä ei tarvitse käsin tehdä kuin korkeintaan muutamaan paikkaan, mieluiten yhteen. Lisäksi verkkohallinnan käyttökokemus paranee, jos kaikki mahdolliset muokattavat asiat eivät ole koko ajan näkyvillä. Asiakas voi myös ladata kuvan päälle laitettavia logoja tai kuvan päälle lisättäviä hauskoja elementtejä. Tämäkin ominaisuus on eroteltu verkkohallinnassa kahteen eri osaan, ja lataamisen jälkeen asiakas näkee kuvat ennen varsinaista palvelimelle lataamista.

3.5 Mypose-laitteen hallinta

Turvallisuussyistä verkkohallinnan kautta ei voida suoraan päivittää Mypose-laitteen sisältöä. Tämä estää myös vahinkotapaukset, joissa esimerkiksi keskeneräiset muutokset päivittyisivät näkyviin kesken jonkin tärkeän tapahtuman. Mypose-sovelluksessa on asetusvalikossa painike, jota painamalla laite hakee uudet asetukset palvelimelta. Lisäksi kehitteillä on automaattinen päivitystoiminto, jonka avulla laite osaa päivittää asetukset tarpeen vaatiessa. Kun uudet päivitykset on ladattu, Mypose-sovellus päivittää uudet muutokset voimaan välittömästi. Jatkokehityksen alla on myös laitestatuksen päivittäminen esimerkiksi tällaisten päivitysten yhteydessä.

Laitteelta lähtee statistiikkapäivityspyyntö aina, kun kuva otetaan tai kun kuva jaetaan johonkin sosiaaliseen mediaan. Osa päivityspyynnöistä lähtee suoraan Mypose-sovelluksesta, josta pyyntö voidaan lähettää palvelimella sijaitsevalle rajapintasovellukselle. Osa päivityspyynnöistä lähtee verkkosovelluksen kautta, jossa esimerkiksi tapahtuu jonkin sosiaalisen media ohjelmointirajapinnan mukaisia pyyntöjä jo valmiina. Näistä pyynnöistä voidaan helposti varmistaa jakamisen onnistuminen. Tällaisen varmistuksen jälkeen statistiikat päivittyvät luotettavasti. Tämän tyyppinen verkkosovellus (kirjoitettu esimerkiksi PHP-kielellä) lähettää samalla tavalla päivityspyynnön palvelimella olevalle rajapintasovellukselle kuin Mypose-sovelluskin.

4 TOTEUTUS

Tämän luvun tarkoitus on antaa tarkempi esittely verkkohallintasovelluksen sisäisistä toiminnollisuuksista. Liikesalaisuus- ja tietoturvasyistä samaa lähdekoodia, mitä palvelimella sijaitsee, ei esitellä. Toiminnollisuudet havainnollistetaan kuitenkin lyhyillä toimivilla ohjelmakoodiesimerkeillä.

Tämän kaltaiselle verkkohallintasovellukselle ei voida käyttää mitään valmista pohjaa. Toteutustapa ja käytettävä teknologia on osattava suunnitella omien taitojen ja käytössä olevien resurssien mukaan. Kokonaisuuden pilkkominen pienempiin komponentteihin auttaa suunnittelu- ja toteutustyössä. Pienemmissä komponenteissa voi sitten hyvinkin käyttää sellaisia perustoiminnallisuuksia, joita löytyy varmasti internetistä valmiiksi tehtynä. Se miten valmiiksi tehtyjä komponentteja sidotaan yhteen turvallisesti, on yleensä ohjelmistosuunnittelijan vastuulla. Tässä työssä monet ratkaisut ovat tehty siten, että ratkaisuun haettiin monta eri vaihtoehtoja joista sitten kasattiin tarpeeseen sopiva paketti omaa tietotaitoa hyväksikäyttäen. Lähteinä ratkaisuihin käytettiin eri teknologioiden ja kielien omia dokumentaatioita ja esimerkiksi alan keskustelupalstoja. Eniten käytetty keskustelupalsta oli todella suosittu Stackoverflow (Stackoverflow. 2014). Esimerkkinä hyvästä ohjelmointirajapinnan dokumentaatiosta on jQueryn kotisivut (jQuery API 2014). Pääsivulta löytyy suoraan viittaukset ja linkit kirjaston funktioitten tarkempiin dokumentaatioihin.

Verkkohallinnan käyttöliittymä on toteutettu HTML-websivun pohjalle. Käyttöliittymäelementtien implementointiin käytettiin jQuery UI:n komponentteja. Tähän ratkaisuun päädyttiin, koska jQueryn käyttö oli tuttua entuudestaan. Toisaalta jQuery UI:n dokumentaatio on monipuolinen ja sen kotisivuilta (jQuery UI 1.11 API Documentation 2014) löytyy hyviä esimerkkejä. Kotisivujen päänäkymässä on linkki ”Widgets”, jota seuraamalla pääsee käsiksi käyttöliittymäkomponenttien dokumentaatioihin. jQuery UI:n käyttäminen nopeutti koko projektin toteutusta, sillä erilaisten käyttöliittymäkomponenttien toimintojen käsin kirjoittamiseen menee yllättävän paljon aikaa, ja tässäkin projektissa näitä komponentteja on melko paljon. Lisäksi jQuery UI:n kotisivuilla on näppärä työkalu yksinkertaisten teemojen suunnitteluun. Tämä työkalu tunnetaan yleisesti nimellä Themeroller (ThemeRoller. 2014). Sen avulla voidaan graafista työkalua hyväksikäyttäen valita jokaiselle käyttöliittymäkomponentille ulkoasu ja tämän jälkeen tallentaa valinnat css-tyylitiedostoina ja JavaScript-tiedostoina. Nämä sitten liitetään sivulle HTML-koodissa. Samalla tallennuksen yhteydessä voi asettaa paketille nimiavaruuden. Tämän luvun esimerkkisovelluksissa on käytetty apuna

PHP:n manuaalia, joka löytyy PHP:n kotisivuilta (PHP Manual 2014). Seuraamalla linkkiä ”Language References” löytyy kaikki tarvittava perustieto PHP-kielestä.

4.1 Ajax-kutsut PHP-sovelluksen ja web-sivun välilllä

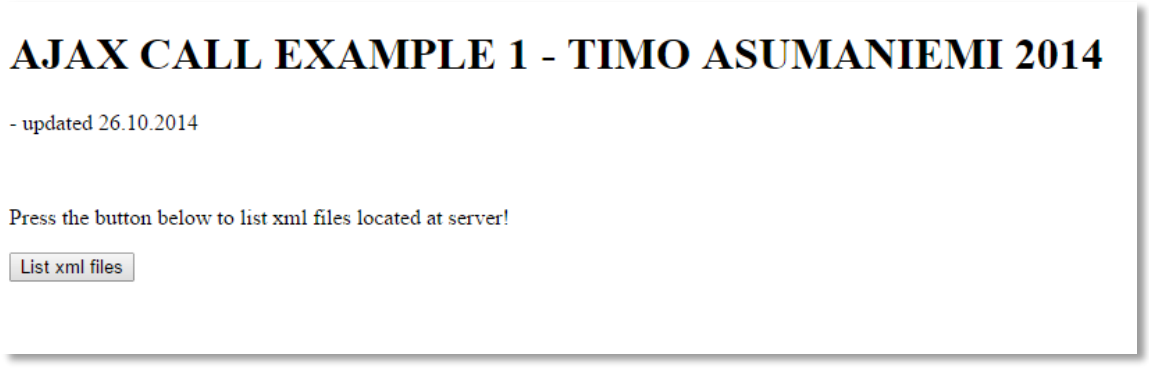
Seuraava esimerkki osoittaa, kuinka yksinkertaisimmillaan voidaan web-sivulta kutsua palvelimella olevaa PHP-sovellusta. Tämä PHP-sovellus palauttaa sitten jotakin palvelimella sijaitsevaa dataa web-sivulle. Kutsut palvelimelle tapahtuvat Ajax-kutsuilla, jotka vaativat toimiakseen jQuery-kirjastojen lisäämisen sivulle. Ajax-kutsut ovat erittäin käyttökelpoisia asynkronisen luonteensa takia. Niiden avulla pitkäkestoisia kutsuja voidaan suorittaa taustalla ilman, että käyttäjä joutuu odottamaan niiden valmistumista. Esimerkkisovellus on kokeiltavissa alla olevassa osoitteessa.

http://timo.devfever.com/oppari/ajax_example_1.

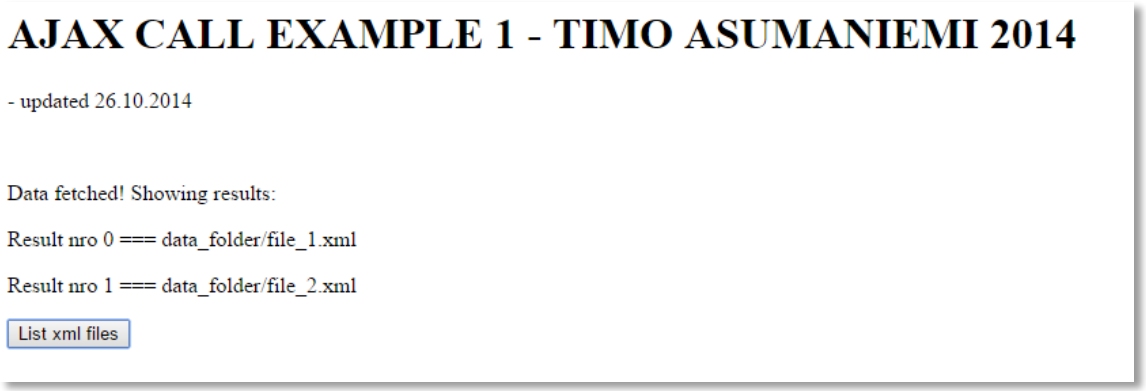
Esimerkkisovellus *ajax_example_1* koostuu seuraavista tiedostoista jotka sijaitsevat palvelimella *ajax_example_1*-kansion juuressa:

- index.html
- ajax_handler.js
- data_handler.php
- data_folder/file_1.xml
- data_folder/file_2.xml.

Kuvissa 6 ja 7 on esitelty sovelluksen toimintaa. Nappia *list xml files* painamalla JavaScript-tiedostossa oleva *fetchData*-funktio lähettää Ajax-kutsun palvelimelle, joka palauttaa listan tietyssä datakansiossa olevista xml-tiedostoista.



KUVA 6. Ajax call example 1



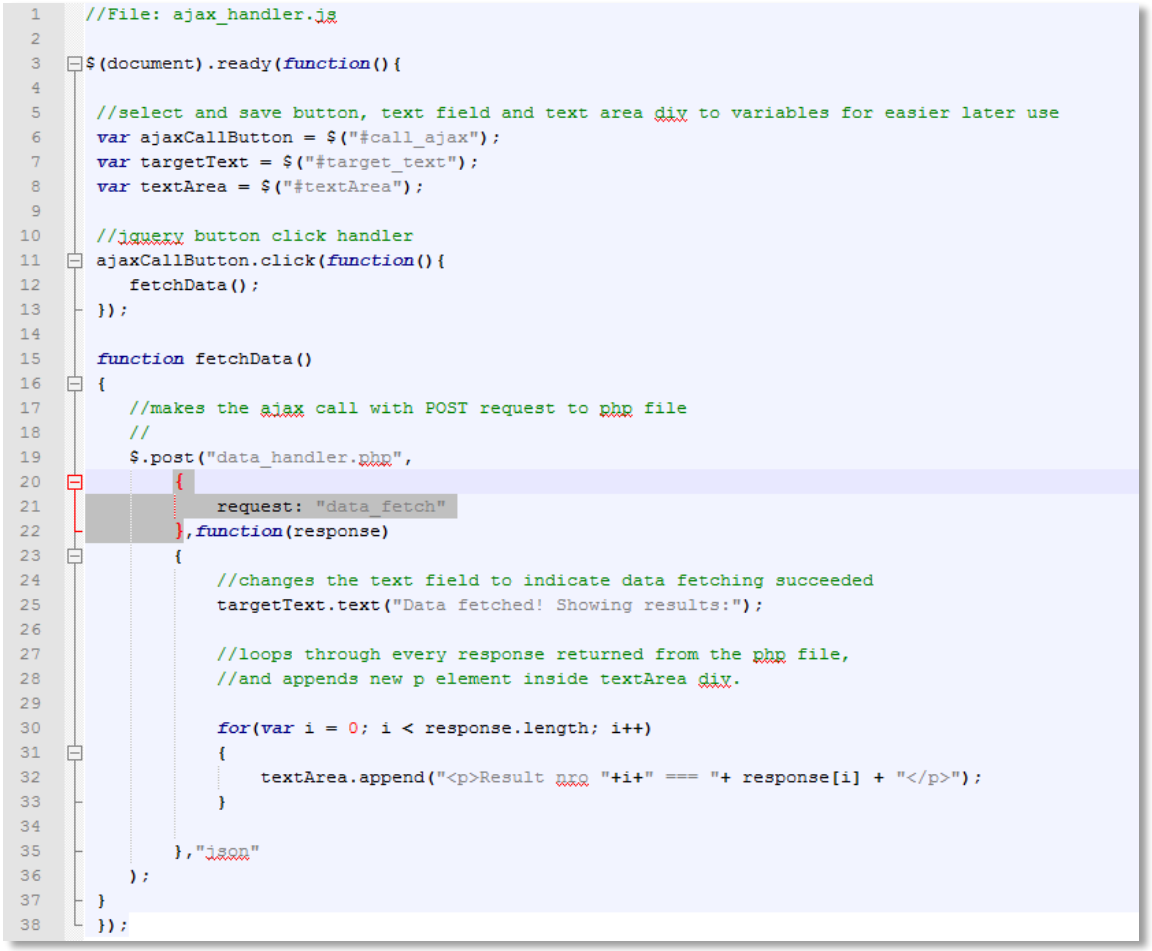
KUVA 7. Ajax call example 2. Tiedostolistaus suoritettu

Kuvan 8 koodiesimerkissä on *index.html*-sivun lähdekoodi sellaisenaan.

```
1  <!-- AJAX CALL EXAMPLE 1--
2  File: index.html
3  -->
4
5  <html lang="en">
6      <head>
7          <meta charset="utf-8">
8
9          <!-- In order to use shortened ajax calls,
10         add jquery library as below -->
11
12         <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
13         </script>
14         <script src="ajax_handler.js"></script>
15
16         <title>ajax_call_ex_1</title>
17 </head>
18
19 <body>
20     <h1>AJAX CALL EXAMPLE 1 - TIMO ASUMANIEMI 2014</h1>
21         <p> - updated 26.10.2014</p>
22         <br>
23
24     <div id="textArea">
25         <p id="target_text">Press the button below to list xml files located at server!</p>
26     </div>
27
28     <button id="call_ajax">List xml files</button>
29 </body>
30 </html>
```

KUVA 8. index.html

Rivillä 12 liitetään jQuery-kirjasto sovellukselle. Tämän kirjaston avulla voidaan mm. yksinkertaistaa Ajax-kutsuja. Rivillä 14 liitetään mukaan JavaScript-tiedosto, jossa suoritetaan tämä kutsu. Kuvassa 9 on esiteltynä edellä mainittu *Ajax_handler.js*-tiedosto.



```
1  //File: ajax_handler.js
2
3  $(document).ready(function(){
4
5     //select and save button, text field and text area div to variables for easier later use
6     var ajaxCallButton = $("#call_ajax");
7     var targetText = $("#target_text");
8     var textArea = $("#textArea");
9
10    //jquery button click handler
11    ajaxCallButton.click(function(){
12       fetchData();
13    });
14
15    function fetchData()
16    {
17       //makes the ajax call with POST request to php file
18       //
19       $.post("data_handler.php",
20          {
21             request: "data_fetch"
22          },function(response)
23          {
24             //changes the text field to indicate data fetching succeeded
25             targetText.text("Data fetched! Showing results:");
26
27             //loops through every response returned from the php file,
28             //and appends new p element inside textArea div.
29
30             for(var i = 0; i < response.length; i++)
31             {
32                textArea.append("<p>Result nro "+i+" === "+ response[i] + "</p>");
33             }
34
35          },"json"
36       );
37    }
38 });
```

KUVA 9. Ajax_handler.js

Korostetuilla riveillä on esitettynä, kuinka helposti voidaan lähettää parametrina tietoa Ajax-kutsun yhteydessä. Rivillä 22 on funktio, jota toimii callback-funktiona eli takaisinkutsufunktiona. Tämä tarkoittaa sitä, että odotetaan jonkin tehtävän päättymistä, jonka jälkeen ohjelma siirtyy suorittamaan tätä takaisinkutsufunktiota. Edellä olevassa kuvassa rivin 22 funktioon siirrytään, kun PHP-sivun toiminnot ovat suoritettu ja vastaus lähetetty. Vastaus voidaan välittää parametrina vastaanottavalle funktiolle. Riville 35 lisätty optio mahdollistaa Json-tyyppisen datan käsittelyn funktion sisällä. Parametri *response* sisältää numeerisesti indeksoidun taulukon, jonka jokaisen indeksin arvona on tässä tapauksessa tiedoston suhteellinen polku palvelimella. Jokainen tietue lisätään HTML-sivun *textArea*-elementtiin tekstikenttänä. Kuvassa 10 esitellään *data_handler.php*-tiedoston sisältö.

```php
<?php

//store POST variable for later use
$request = $_POST['request'];
// folder to search files from
$dataFolderPath = "data_folder";

//if post variable is correct, proceed
if($request = "data_fetch")
{
    //glob is handy way for listing or searching certain files in folder.
    //inside curly braces you can idenitify file type.
    $filesInDirectory = glob("$dataFolderPath/*.{xml}",GLOB_BRACE);

    //$filesInDirectory variable is now an array, indexed by integers,
    //so it's easier to parse later with javascript after encoding it to json.
    echo json_encode($filesInDirectory);
}
else
{
    echo "invalid POST parameter";
}
?>
```

KUVA 10. data_handler.php

Vaikka tämä on hyvin yksinkertaistettu esimerkki siitä, miten web-sivulta päästään käsiksi palvelimella olevaan dataan, antaa se hyvän näkemyksen kuinka edellä mainittuja tapoja käyttäen voidaan rakentaa hyvinkin monimutkaisia pyyntöjä. Tämän tapaisia pyyntöjä käytetään myös verkkohallintasovelluksessa todella paljon. Esimerkiksi samankaltaista tapaa voisi käyttää vaikkapa haettaessa jonkin tietyn laitteen kampanjaan asetetut png-kuvat tai haettaessa konfiguraatiotiedoston polku lataamista varten.

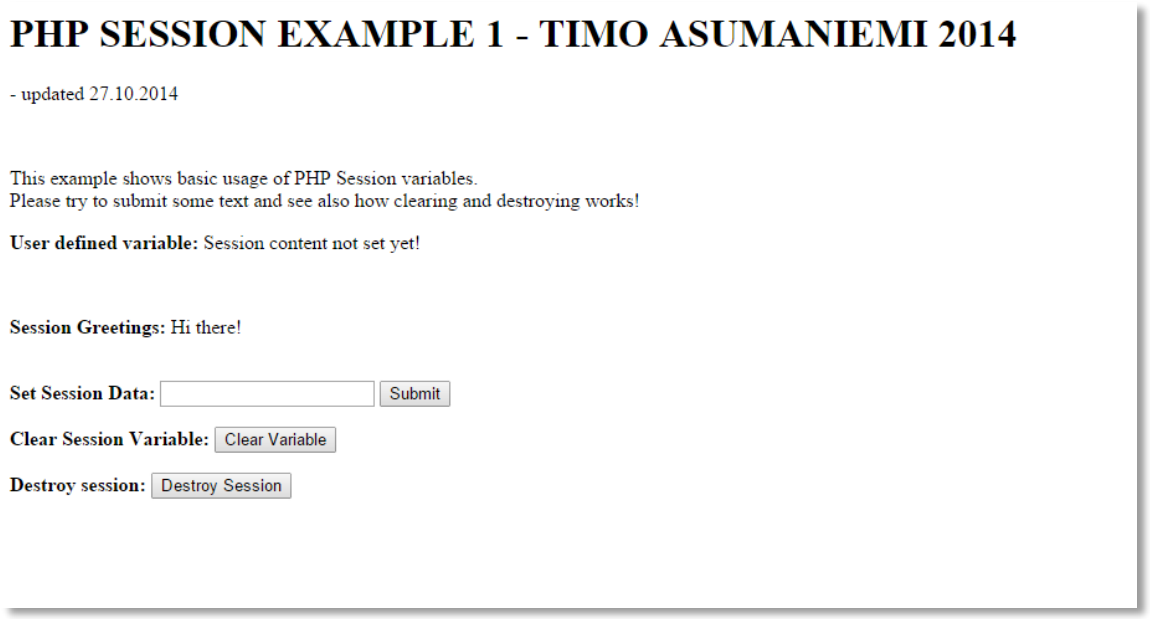
4.2 PHP-istuntojen käyttö väliaikaisen datan tallentamisessa

PHP-koodissa muuttujia voidaan käyttää vain PHP-koodin suorituksen loppuun. Ne eivät ole sen jälkeen voimassa. Istuntoja voidaan käyttää, jos halutaan tallentaa dataa muuttujiin joiden pitää säilyä vaikkapa sivulta toiselle. Yleisin käyttötapa istuntojen käytölle on käyttäjän tunnistaminen jonkin kirjautumissivun tai ruudun jälkeen. Oletuksena istuntomuuttujat säilyvät niin kauan kunnes käyttäjä sulkee selaimen. Istuntomuuttujat tallennetaan globaalin PHP-muuttujan *$_SESSION* alle. Jotta istuntomuuttujia voidaan käyttää, täytyy PHP-sivun ensimmäisellä rivillä kutsua metodia *session_start()*.

Seuraava yksinkertainen esimerkki näyttää, miten istuntomuuttujat toimivat käytännössä. Esimerkkiä voi kokeilla osoitteessa

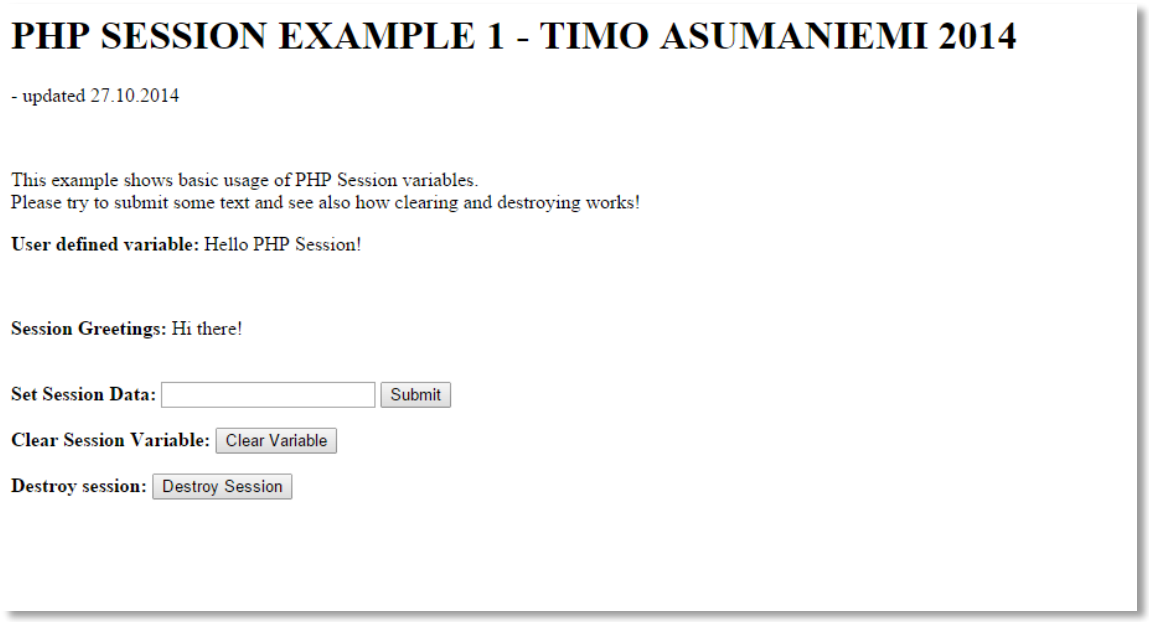
http://timo.devfever.com/oppari/php_session_example_1/index.php.

Esimerkkisovellus koostuu yhdestä tiedostosta *index.php*, joka sijaitsee palvelimella kansion *php_session_example_1* juuressa. Sovelluksessa käytetään kahta istuntomuuttujaa. *greetings*-muuttuja asetetaan istunnon käynnistyksen yhteydessä ja pitää sisältää tervehdysviestin. *content_text*-muuttujaan tallennetaan käyttäjän syöttämä tekstikenttä. Nappia *Clear Variable* painamalla sovellus poistaa *content_text*-muuttujan sisällön. Nappia *Destroy Session* painamalla sovellus tyhjentää kaikki istuntomuuttujat ja tuhoaa istunnon. Molemmat napit lähettävät pyynnön samaiselle *index.php*-tiedostolle, jolloin sivu päivittyy. Samalla tavalla pyynnön voisi lähettää muullekin PHP-sivulle (tai ASP-sivulle). Tällöin täytyy muistaa vain laittaa ensimmäiselle riville kutsu *session_start()*, jotta isuntoa voidaan hyödyntää. Kuvassa 11 on sivun perusnäkymä.



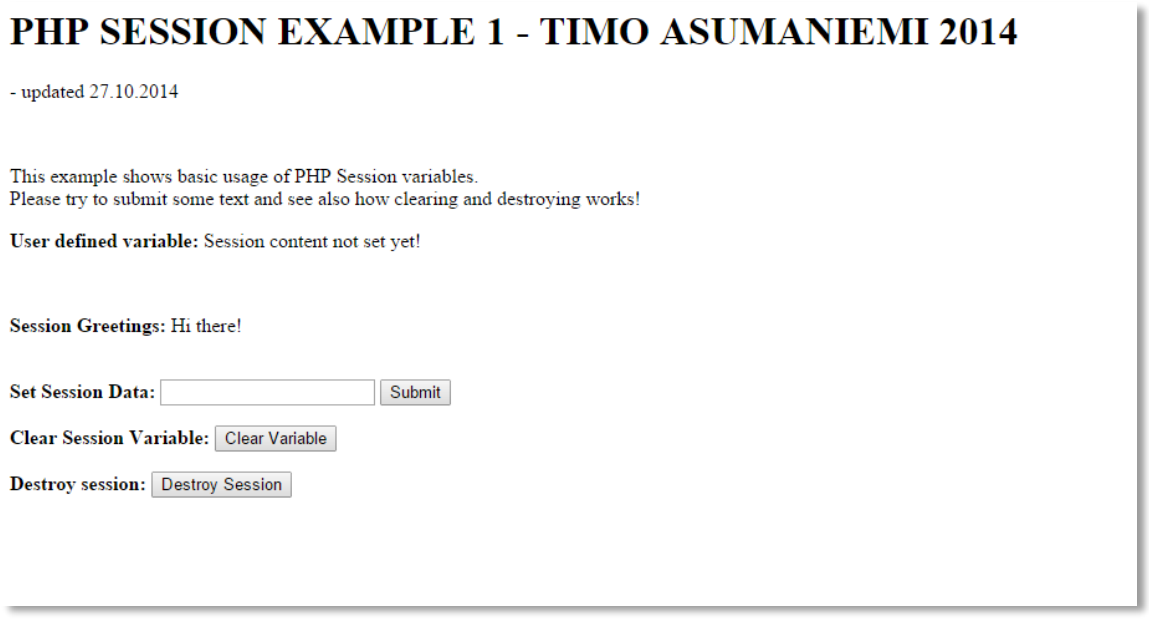
KUVA 11. PHP-istunto, perusnäkymä.

Kuvassa 12 nähdään, mitä tapahtuu, kun käyttäjä on syöttänyt tekstiä tekstikenttää ja lähettänyt pyynnön.



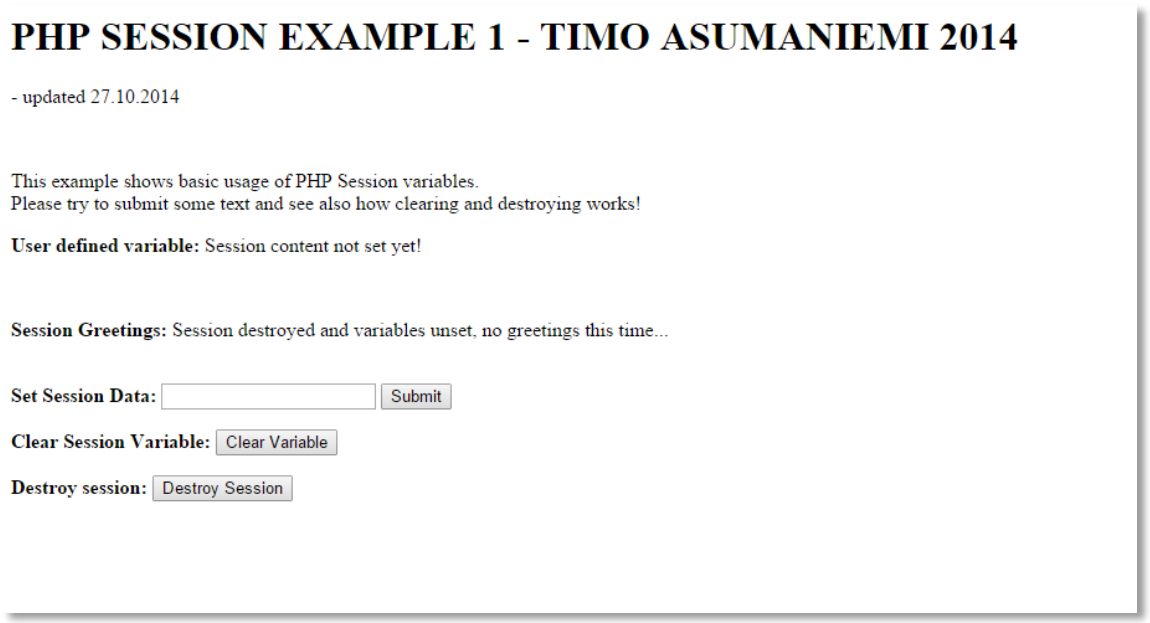
KUVA 12. PHP-istunto, teksti lähtetty

Kuvassa 13 nähdään, mitä tapahtuu kun *Clear Variable*-nappia on painettu.



KUVA 13. PHP-istunto, istuntomuuttuja tuhottu

Lopuksi kuvassa 14 nähdään, mitä tapahtuu kun *Destroy Session*-nappia on painettu.

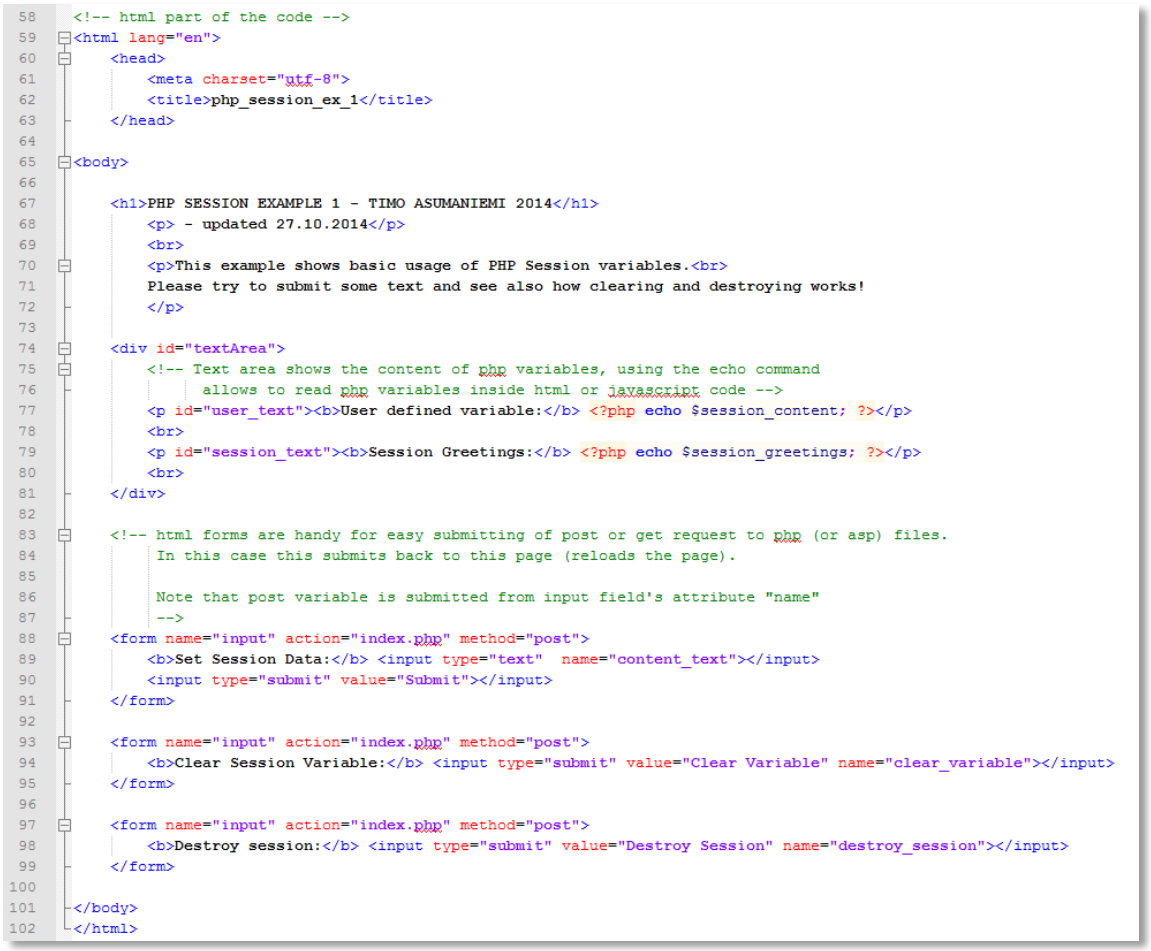


KUVA 14. PHP-istunto, istunto tuhottu

Kuvissa 15 ja 16 on esiteltynä edellä mainitun esimerkin lähdekoodi kahdessa osassa. Molemmat kuvat kuuluvat samaan tiedostoon *index.php*, mutta kuvassa 15 on tiedoston PHP-osio ja toisessa kuvassa 16 on tiedoston HTML-osio.

```php
<!-- PHP SESSION EXAMPLE 1--
File: index.php
-->

<!-- php part of the code -->

<?php
//To use session put session_start at the very beginning of php file
session_start();

//set predefined content for 'greetings' session variable
$_SESSION['greetings'] = "Hi there!";

//if request is to clear variable, unset the session variable so it deletes the content of the variable
if($_POST['clear_variable'] === "Clear Variable")
{
    unset($_SESSION['content']);
}

//if request is to destroy session do the following:
if($_POST['destroy_session'] === "Destroy Session")
{
    // removes all session variables
    session_unset();

    // destroy the session
    session_destroy();
}

//set the user defined text to session variable (only if it's not empty!)
if($_POST['content_text'] !== "")
{
    $_SESSION['content'] = $_POST['content_text'];
}

//if session variable has content stored, store it for later use
if(isset($_SESSION['content']))
{
    $session_content = $_SESSION['content'];
}
else //show message that session variable is empty
{
    $session_content = "Session content not set yet!";
}

//if default greeting is set to session variable 'greetings' save it for later use
if(isset($_SESSION['greetings']))
{
    $session_greetings = $_SESSION['greetings'];
}
else //if default session 'greetings' variable is destroyed show message about it
{
    $session_greetings = "Session destroyed and variables unset, no greetings this time...";
}

?>
```

KUVA 15 PHP-istuntomuuttujien käyttö. Sivun index.php PHP-osio.



```
<!-- html part of the code -->
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>php_session_ex_1</title>
    </head>

<body>

    <h1>PHP SESSION EXAMPLE 1 - TIMO ASUMANIEMI 2014</h1>
        <p> - updated 27.10.2014</p>
        <br>
        <p>This example shows basic usage of PHP Session variables.<br>
        Please try to submit some text and see also how clearing and destroying works!
        </p>

    <div id="textArea">
        <!-- Text area shows the content of php variables, using the echo command
             allows to read php variables inside html or javascript code -->
        <p id="user_text"><b>User defined variable:</b> <?php echo $session_content; ?></p>
        <br>
        <p id="session_text"><b>Session Greetings:</b> <?php echo $session_greetings; ?></p>
        <br>
    </div>

    <!-- html forms are handy for easy submitting of post or get request to php (or asp) files.
         In this case this submits back to this page (reloads the page).

         Note that post variable is submitted from input field's attribute "name"
         -->
    <form name="input" action="index.php" method="post">
        <b>Set Session Data:</b> <input type="text"  name="content_text"></input>
        <input type="submit" value="Submit"></input>
    </form>

    <form name="input" action="index.php" method="post">
        <b>Clear Session Variable:</b> <input type="submit" value="Clear Variable" name="clear_variable"></input>
    </form>

    <form name="input" action="index.php" method="post">
        <b>Destroy session:</b> <input type="submit" value="Destroy Session" name="destroy_session"></input>
    </form>

</body>
</html>
```

KUVA 16. PHP-istuntomuuttujien käyttö. Sivun index.php HTML-osio.

4.3 Tietokantojen käsittely PHP-sovelluksesta

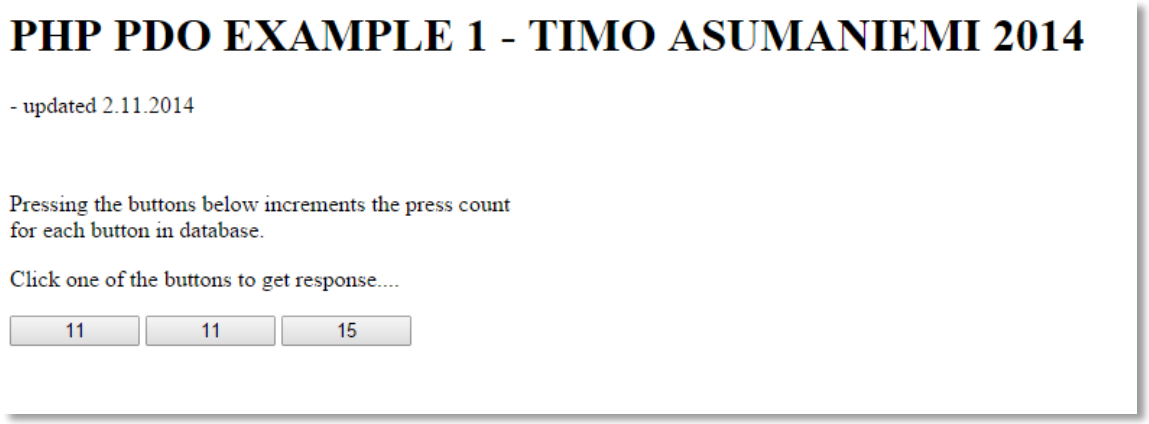
Tietokantojen käsittelyä varten toteutettiin siis PHP-rajapinta, joka helpottaa datavirran ohjaamista tietokannasta sitä kutsuvalle sovellukselle. Liikesalaisuussyistä en esittele tarkasti rajapinnan toimintoja tai lähdekoodia, vaan havainnollistan sen keskeisimmät toiminnat esimerkkien avulla. Hyvin nykyaikainen tapa käyttää MySQL-tietokantaa PHP:llä on käyttää apuna PDO-rajapintaa. Rajapinta on ollut mukana PHP:n versiosta 5.1 eteenpäin. PDO-rajapinnan avulla on helppo valmistella tietokantakyselyt ja sitoa joitakin arvoja muuttujiin. Tällä tavoin pystytään suojautumaan mm. SQL-injektioita vastaan. SQL-injektio on tekniikka, jolla käyttäjä voi injektoida SQL-komentoja SQL-kyselyjen seassa, esimerkiksi web-sivulla olevan tekstikentän avulla, joka on osana SQL-kyselyä. Tällä tavoin hakkerit voivat vaikkapa tuhota tai muuttaa jonkin tietokannan taulun sisällön huolimattomasti toteutetun koodin takia. Apuna tietokantojen käsittelyssä käytettiin MySQL:n dokumentaatiota (MySQL Reference Manual, 2014). Sivun päänäkymässä olevan otsikon ”Table of Contents” alta löytyy kaikki tarvittava perustieto MySQL-kielestä.

Seuraava esimerkki koostuu kolmesta komponentista, HTML-sivusta, PHP-tiedostoista ja MySQL-tietokannasta. Tietokantojen käsittely on jaettu kahteen komponenttiin, *main.php* ja *db_handler.php*. Websivu koostuu *index.html*-pääsivusta sekä *ajax_handler.js*-tiedostosta. Esimerkissä on web-sivulla kolme nappia, joita painamalla tietokantaan päivittyy kunkin napin painokerrat ja nämä näytetään käyttäjälle napin tekstikentässä välittömästi.

Tietokannan ja websivun välillä rajanpintana toimii *main.php*-tiedosto. Web-sivulta lähtee tuttuun tapaan Ajax-kutsu palvelimelle ja nimenomaan *main.php*-tiedostolle. Tämä tiedosto käsittelee halutun tehtävän ja tekee sen perusteella kutsun tietokantoja varsinaisesti käsittelevälle *db_handler.php*-sovellukselle. Ennen tehtävän kutsumista luodaan tietokantaluokasta olio. Kyseisen luokan konstruktorissa luodaan pohjalle uusi yhteys tietokantaan esitäytetyillä tiedoilla ja asetuksilla valmiiksi. Tietokantaluokka sisältää julkisia funktioita, jotka suorittavat halutun tietokantakyselyn. Funktion paluuarvona voidaan välittää dataa tai virhetilanteen sattuessa viesti. Tämä PHP-rajapintasovellus palauttaa datan sitä kutsuneelle web-sivulle tuttuun tapaan Json-tyyppisenä objektina. Tätä esimerkkiä voi kokeilla osoitteessa

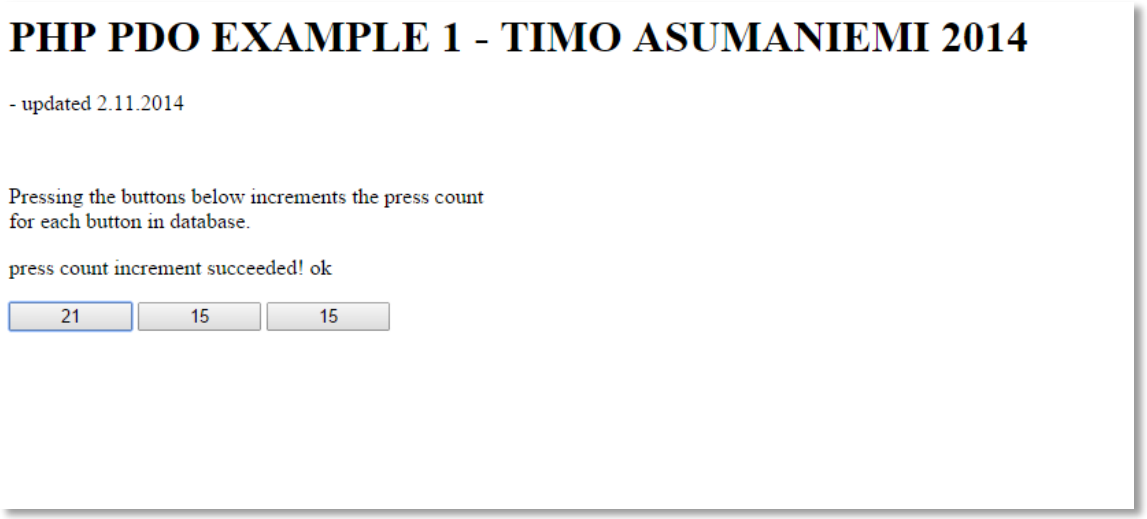
http://timo.devfever.com/oppari/php_pdo_example_1/index.html.

Kuvassa 17 on ensimmäinen näkymä, kun käyttäjä on avannut sivun.



KUVA 17. Tietokantojen käsittely web-sivulta, päänäkymä

Kuvassa 18 käyttäjä on painanut nappeja ja tulokset näkyvät nappien tekstikentässä.



KUVA 18. Tietokantojen käsittely web-sivulta

Kuvien 17 ja 18 esimerkki koostuu seuraavista tiedostoista, jotka sijaitsevat palvelimella *php_pdo_example_1*-kansion juuressa.

- index.html

- ajax_handler.js

- main.php

- db_handler.php

Kuvassa 19 on esiteltynä *index.html*-tiedosto.

```
<!-- PHP PDO EXAMPLE 1--
File: index.html
Requires: main.php,db_handler.php
-->
<html lang="en">
    <head>
        <meta charset="utf-8">

        <!-- In order to use shortened ajax calls,
        add jquery library as below -->

        <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js">
        </script>

        <title>php_pdo_ex_1</title>
</head>

<body>
    <h1>PHP PDO EXAMPLE 1 - TIMO ASUMANIEMI 2014</h1>
        <p> - updated 2.11.2014</p>
        <br>

    <!-- info text area -->
    <div id="textArea">
        <p id="target_text">Pressing the buttons below increments the press count <br> for each button in database.</p>
        <p id="response_message">Click one of the buttons to get response....</p>
    </div>


    <div style="display:inline-block;">
        <button class="press_button" style="width:90px" id="first_button">First Button</button>
    </div>

    <div style="display:inline-block;">
        <button class="press_button" style="width:90px" id="second_button">Second Button</button>
    </div>

    <div style="display:inline-block;">
        <button class="press_button" style="width:90px" id="third_button">Third Button</button>
    </div>

</body>
<!-- include the script file to handle button clicks -->
<script src="ajax_handler.js"></script>
</html>
```

KUVA 19. index.html PHP_PDO_EXAMPLE_1

Rivistä 23 eteenpäin luodaan sivulle info-tekstit ja painonapit. Nappien tapahtumien käsittely tapahtuu *ajax_handler.js*-tiedostossa, josta lähdekoodi esitelty kuvassa 20.

```
1
2   //click handler for all buttons
3   $(".press_button").click(function(){
4       update_press_count(this.id);
5   });
6
7
8   //show current situation at startup
9   $(".press_button").each(function(){
10      show_press_count(this.id);
11  });
12
13
14  function update_press_count(button_id)
15  {
16      //this function makes the actual post call to main.php to
17      //update the database
18      console.log("button " + button_id + " pressed");
19      $.post('main.php',{
20          action: "increment_button_press",
21          button: button_id
22      },function(response)
23      {
24          $("#response_message").text(response.result);
25          //call the function to show the current result on buttons text field
26          show_press_count(button_id);
27      },"json"
28      );
29  }
30
31  function show_press_count(button_id)
32  {
33      //this function calls the same main.php fileCreatedDate
34      //but using actions we tell it to just get the current
35      //press count by requested button's id
36      $.post('main.php',{
37          action: "get_press_count",
38          button: button_id
39      },function(response)
40      {
41          //show the result directly on this button
42          $("#"+button_id).text(response.result);
43      },"json"
44      );
45  }
```

KUVA 20. ajax_handler.js

Koska kaikilla napeilla on sama luokka, voidaan jQueryä hyväksikäyttäen asettaa funktiokutsu yhdellä kertaa, kuten riveillä 3–5 nähdään. Funktio *update_press_count* ottaa siis vastaan parametrina halutun napin id:n ja välittää sen Ajax-kutsun yhteydessä *main.php*-sovellukselle, joka välittää kyseisen päivityspyynnön eteenpäin. Vastauksena palvelimelta tulee kyseisen napin tämän hetkiset painokerrat lukuina, ja ne näytetään suoraan web-sivulla olevan napin tekstikentässä. Kuvassa 21 on esitelty rajapintasovelluksen *main.php* lähdekoodi.

```php
<?php

//create an array of valid actions
$listed_actions = array();
$listed_actions[] = "increment_button_press";
$listed_actions[] = "get_press_count";

//filter the posted string variable (FILTER_SANITIZE_STRING removes unwanted characters in posted string)
$action_requested = filter_input(INPUT_POST, 'action', FILTER_SANITIZE_STRING);

// "include" database handler php file
require_once 'db_handler.php';

//Check that requested action is in list of valid actions
if(in_array($action_requested, $listed_actions,true))
{
    $action = $action_requested;
    $buttonID = filter_input(INPUT_POST, 'button',FILTER_SANITIZE_STRING);
    //make new instance of DatabaseHandler Class
    $db_conn = new DatabaseHandler();

    //switch through all valid actions and make corresponding actions
    switch($action)
    {
        case "increment_button_press":
                //call the database handlers public function to make the actual database query
                $response = $db_conn->incrementButtonPress($buttonID);

                if($response === "ok")
                {
                    $returnMessage = array(result => "press count increment succeeded!");
                }
                else
                {
                    $returnMessage = array(result => "press count increment failed! $response");
                }

                echo json_encode($returnMessage);
        break;

        case "get_press_count":
                $pressCount = $db_conn->getPressCount($buttonID);
                $returnPressCount = array( result => $pressCount);
                echo json_encode($returnPressCount);
        break;

    }
}
else
{
    //if invalid action is requested, do nothing and return some error message
    $returnMessage = array(result => "invalid main action");
    echo json_encode($returnMessage);
}

?>
```

KUVA 21. Rajapintasovellus main.php

Kuvassa 21 esitellyssä koodissa on muutama uusi asia edellisiin esimerkkeihin nähden. Huomioitavaa on se, kuinka POST-parametri suodatetaan ennen sen varsinaista käyttöä. PHP:n avulla suodatus tapahtuu *filter_input()*-komennon avulla kolmella parametrilla. Ensimmäinen parametri kertoo suodatettavan datan tyypin. Tässä tapauksessa kyseessä on POST-pyyntö, eli parametriksi asetetaan *INPUT_POST,* kuten rivillä 18 nähdään. Toinen parametri on muuttujan nimi, joka siis vastaa suoraan edellisessä esimerkissä käytettyä *$_POST['action']*-komentoa. Viimeinen parametri määrää minkälaisesta suodattimesta on kyse. Tässä esimerkissä käytetty *FILTER_SANITIZE_STRING* poistaa merkkijonossa olevat erikoismerkit tai suodattaa ne pois näkyvistä. Esimerkiksi jos välitetty merkkijono olisi muotoa *" Testi teksti "*, olisi se suodatuksen jälkeen *"Testi*

teksti”. Tämä on yksi tapa ehkäistä haitallisten parametrien välitystä palvelimelle. Kun haluttu tehtävä on sallittujen tehtävien listalla (kuvan 21 rivit 4–5), voidaan varsinaisia tietokantakyselyjä tekevästä luokasta luoda olio (rivi 20). Luokasta löytyy funktio *incrementButtonPress*, jolle välitetään POST-parametrina lähetetty napin id. Kuvassa 22 esitellään tietokantakyselyt suorittava tiedosto *db_handler.php.*

```php
<?php

//Declare DataBaseHandler class
class DatabaseHandler
{
    //actual connection should be used only by this class
    private $db_connection;

    //At the constructor, db connection is established by some default settings

    public function __construct()
    {
        $database_user = [redacted];
        $database_password = [redacted];

        //sets some PDO settings for error handling and utf 8 encoding
        $database_options = array(PDO::ATTR_PERSISTENT => false, PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES 'utf8'");

        //establish default db connection
        try
        {
            $this->db_connection = new PDO('mysql:host=localhost; dbname=php_mysql_ex_1_testdb', $database_user, $database_password, $database_options);
        }
        catch(PDOException $exception)
        {
            return "database updating failed $exception";
        }
    }

    public function incrementButtonPress($button_id)
    {
        if($this->db_connection !== null)
        {
            try
            {
                //make prepared sql queries to prevent sql injection attacks
                // wanted value can be bind to temporary identificator (this case it's :button)
                $sql_query = $this->db_connection->prepare('UPDATE button_presses SET button_pressed = button_pressed + 1 WHERE button_name = :button');
                $sql_query->bindValue(":button",$button_id,PDO::PARAM_STR);

                if($sql_query->execute())
                {
                    return "ok";
                }
                else
                {
                    return "database updating failed";
                }
            }
            catch(PDOException $exception)
            {
                return "SQL Query error," . $exception;
            }
        }
        else
        {
            return "database_connection not set up!";
        }
    }

    public function getPressCount($button_id)
    {
        if($this->db_connection !== null)
            {
                try
                {
                    $sql_query = $this->db_connection->prepare('SELECT button_pressed FROM button_presses WHERE button_name = :button');
                    $sql_query->bindValue(":button",$button_id,PDO::PARAM_STR);

                    if($sql_query->execute())
                    {
                        $press_count = $sql_query->fetch();
                        return $press_count['button_pressed'];
                    }
                    else
                    {
                        return "database updating failed";
                    }
                }
                catch(PDOException $exception)
                {
                    return "SQL Query error," . $exception;
                }
            }

        else
        {
            return "database_connection not set up!";
        }
    }
}


?>
```

KUVA 22. Tietokannan käsittelijä db_handler.php

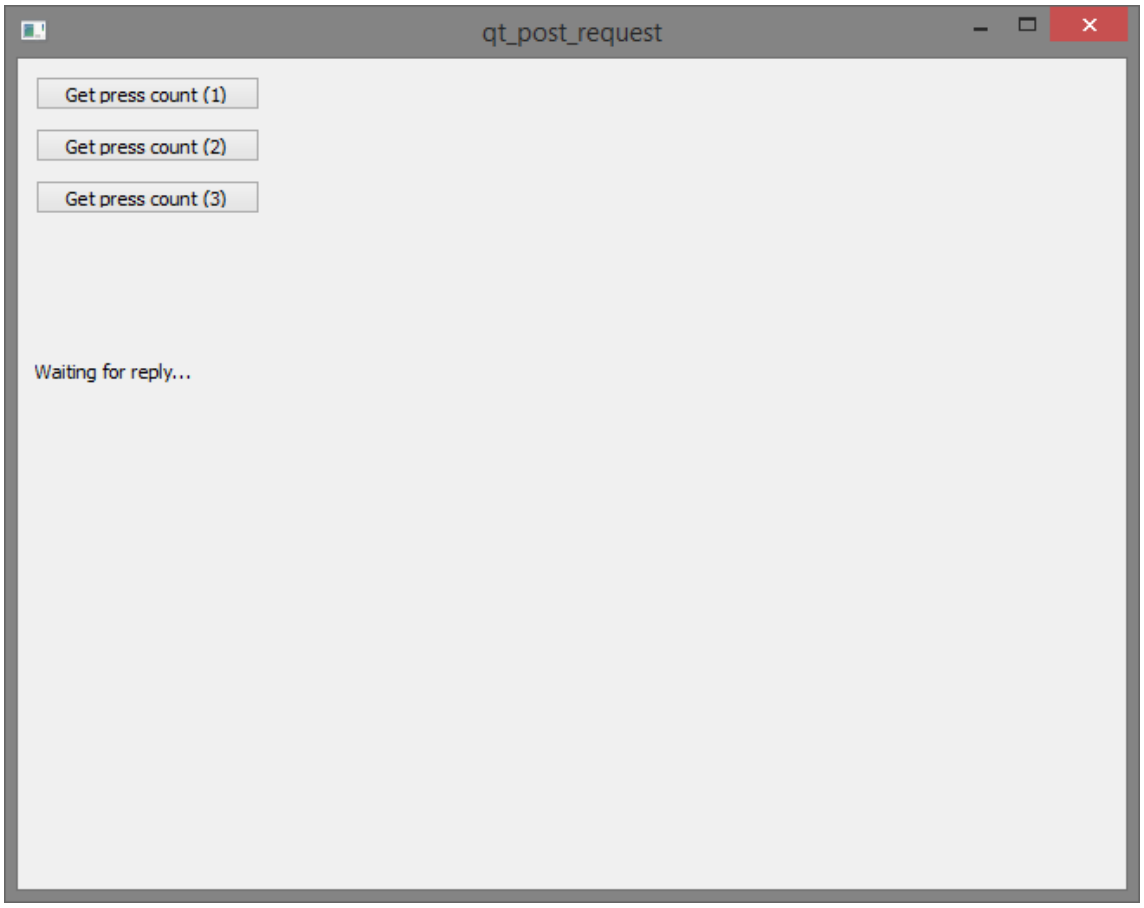
Tiedosto koostuu siis luokasta *DatabaseHandler*. Sen konstruktorissa alustetaan ja luodaan uusi yhteys tietokantaan. PDO-rajapinnan avulla voidaan alustaa yhteys parametreilla, joiden avulla voidaan määrittää käyttäjätunnus, salasana ja PDO:n asetuksia. Virheiden käsittelyä varten on lisätty parametri *PDO::ERRMODE_EXCEPTION* (kuvan 22 rivillä 17), joka auttaa virheen analysoimisessa huomattavasti. Sen avulla voidaan try-catch periaatteella napata virheet ja *PDOException* näyttää missä tiedostossa, millä rivillä ja minkätyyppinen virhe oli kyseessä. Tätä tapaa käytin Myposen verkkohallinnan testauksessa ja omatekoisen loki-kirjoittajan kanssa ohjelmoinnissa.

4.4 Qt-Sovelluksen kommunikaatio palvelimen kanssa

Kommunikaatio Qt-sovelluksen ja palvelimen välillä tapahtuu niin ikään HTTP-protokollan POST-pyyntöjä käyttämällä. Qt-sovelluksen C++ kielellä kirjoitetun pyynnön lähettäminen vaati hiukan enemmän koodia kuin esimerkiksi luvun 4.1 Ajax-esimerkissä. Palvelimella sijaitsevat pienoissovellukset palauttavat yleensä dataa Json-muodossa, joten senkin takia C++ kielellä kirjoitetut pyynnöt ovat hiukan monimutkaisempia. Seuraava esimerkki on tehty Qt:n versiolla 5.3, jossa käytetään kappaleen aikaisemmin esiteltyä PHP-rajapintaa. Koska kyseessä on Windows-sovellus, ei sitä voida suoraan testata web-selaimella. Esimerkki koostuu seuraavista tiedostoista:

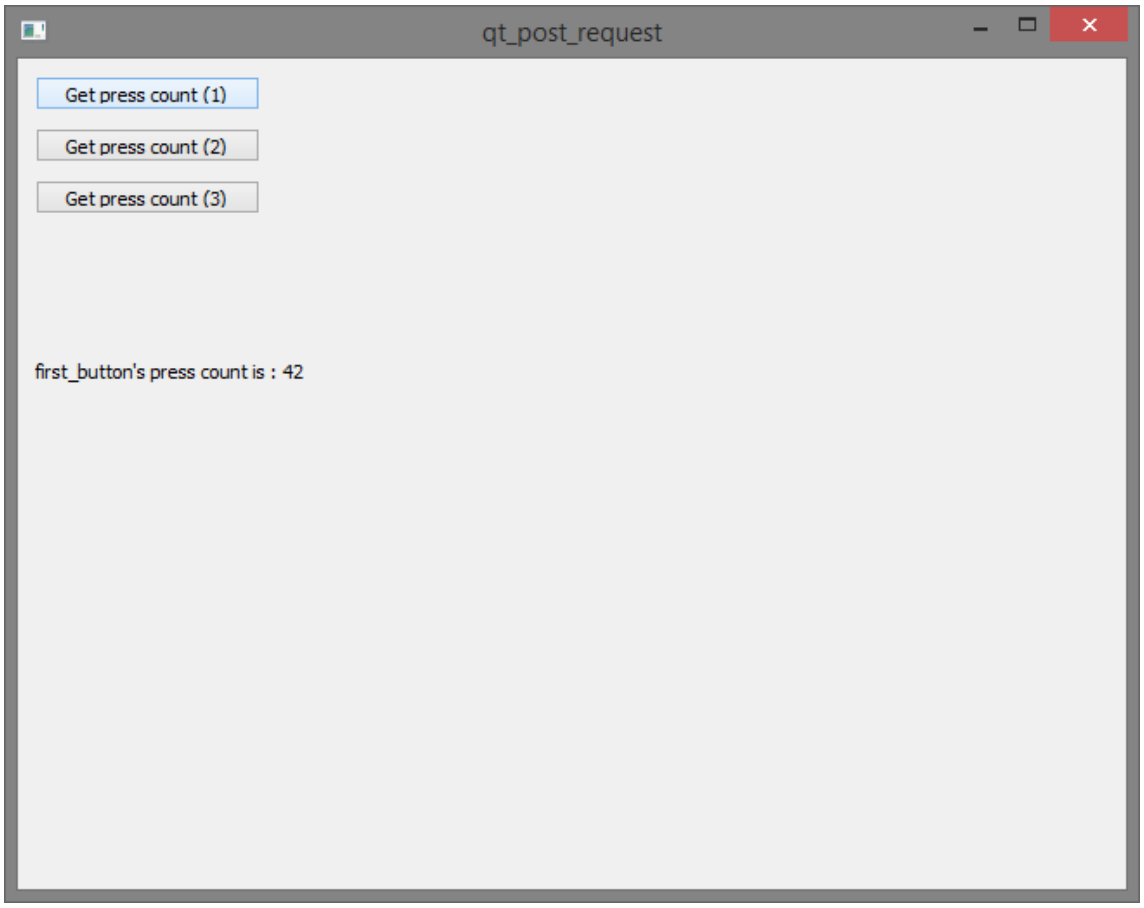
- *qt_post_request.pro*
- *mainwindow.h*
- *mainwindow.cpp*
- *main.cpp.*

Kuvassa on 23 esimerkkisovelluksen päänäkymä, jossa ohjelma on käynnistetty.



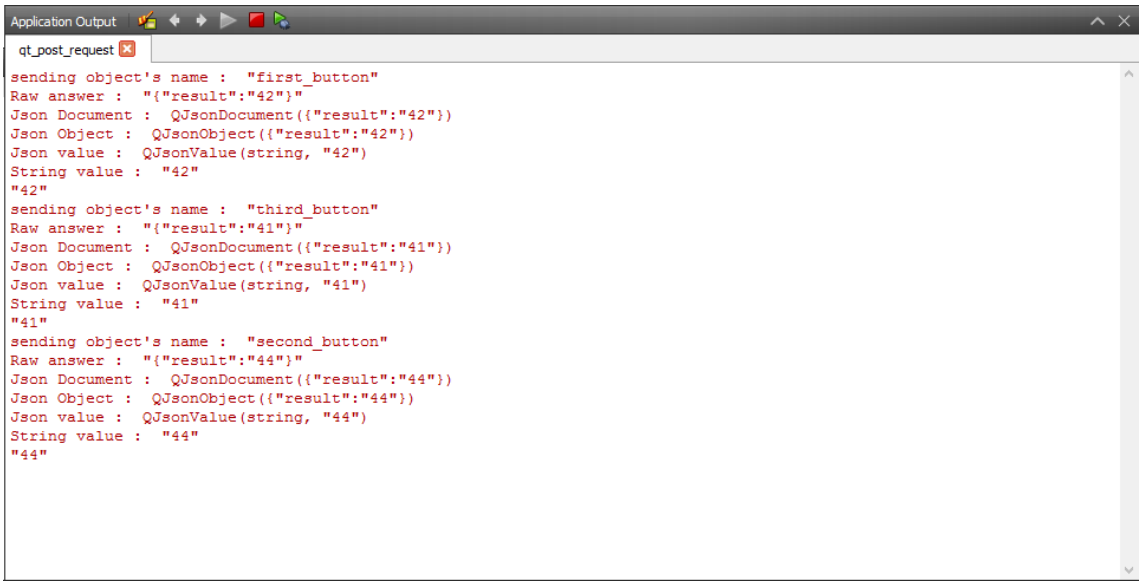
KUVA 23. Qt post-pyyntö esimerkki, ohjelman aloitus

Kuvassa 24 käyttäjä on painanut ensimmäistä nappia ja vastaus palvelimelta näytetään tekstikentässä.



KUVA 24. Qt post-pyyntö esimerkki, ensimmäistä nappia painettu

Kuvassa 25 on Qt-ohjelman loki-ikkuna, joka tulostaa *QDebug*-luokan *qDebug()*-komennolla *debug*-viestejä ikkunaan. Jokaisesta napin painalluksesta on kuvan 25 mukaisesti tulostunut muutama rivi tietoa, joitten avulla hoksaa helpommin miten Qt:n avulla parseroidaan json-data ymmärrettävämpään muotoon.



KUVA 25. Qt post-pyyntö esimerkki, debug-loki

Seuraavaksi käyn läpi Qt-ohjelman lähdekoodin. Esittelen vain *mainwindow.cpp*-tiedoston sisällön sekä pätkän *mainwindow.h*-tiedostosta, sillä ohjelman yksinkertaisen luonteen vuoksi niistä ymmärtää täysin ohjelman toiminnan. Kuvassa 26 on esitelty *mainwindow*-luokan konstruktori, jossa luodaan painonapit ja tekstikenttä sekä liitetään painonappien *click*-signaalit *sendRequest*-funktioon. Lopussa vielä liitetään funktiolta tuleva signaali, joka ilmoittaa vastauksenkäsittelyn olevan valmis, *printReply*-funktioon (eli slottiin), joka hoitaa vastauksen tulostamisen tekstikenttään.

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent),
      mSendRequest_button1(0),
      mSendRequest_button2(0),
      mSendRequest_button3(0),
      mReply_label(0),
      mButton_id(0)
{
    //set mainwindow geometry
    this->setGeometry(100,100,640,480);
    mSendRequest_button1 = new QPushButton(this);
    mSendRequest_button2 = new QPushButton(this);
    mSendRequest_button3 = new QPushButton(this);

    //initalize label to show reply from network request
    mReply_label = new QLabel("Waiting for reply...",this);
    mReply_label->setGeometry(10,170,500,20);

    //initialize push buttons
    //note that I also set names for each instance of push buttons
    mSendRequest_button1->setGeometry(10,10,130,20);
    mSendRequest_button1->setText("Get press count (1)");
    mSendRequest_button1->setObjectName("first_button");

    mSendRequest_button2->setGeometry(10,40,130,20);
    mSendRequest_button2->setText("Get press count (2)");
    mSendRequest_button2->setObjectName("second_button");

    mSendRequest_button3->setGeometry(10,70,130,20);
    mSendRequest_button3->setText("Get press count (3)");
    mSendRequest_button3->setObjectName("third_button");

    //connect click signal for each button to sendRequest function
    connect(mSendRequest_button1,SIGNAL(clicked()),this,SLOT(sendRequest()));
    connect(mSendRequest_button2,SIGNAL(clicked()),this,SLOT(sendRequest()));
    connect(mSendRequest_button3,SIGNAL(clicked()),this,SLOT(sendRequest()));
    //when parsing the json encoded reply is finished, print reply to label
    connect(this,SIGNAL(replyParsed(QString)),this,SLOT(printReply(QString)));
}
```

KUVA 26. Qt post-pyyntö esimerkki, mainwindow.cpp osa 1 / 3

Kuvassa 27 *sendRequest*-funktiossa alustetaan ja suoritetaan varsinainen kutsu palvelimelle. Ensimmäisenä tallennetaan funktiota kutsunut objekti osoitinmuuttujaan *senderObject*, jolloin konstruktorissa jokaiselle painonapille asetettu objektin nimi voidaan tarkastaa tässä vaiheessa ja käyttää sitä hyödyksi myöhemmin. POST-pyyntö aloitetaan määrittelemällä polku palvelulle ja rivillä 56 se tallennetaan *QUrl*-tyyppiseen muuttujaan. *QUrlQuery*:n avulla voidaan muodostaa yksinkertaista tekstipohjaista dataa sisältäviä parametreja POST-pyynnöllä lähetettäväksi. Parametrit asetetaan ko. luokan *addQueryItem*-funktiolla johon asetetaan kaksi parametria, avain-arvoparit eli *key-value pair*. Tässä esimerkissä käytetään samaa PHP-rajapintaa kuin aikaisemmissakin esimerkeissä, joten näin ollen käytetään kutsuissa myös samoja parametreja. Ensimmäisessä parametrissa määritellään tuttuun tapaan *action*-parametrin arvo, jonka php-rajapinta *main.php* käsittelee ja suorittaa halutun kyselyn. Riviltä 64 eteenpäin luodaan *QNetworkAccessManager*-luokasta olio, jonka avulla POST-pyyntö voidaan käytännössä suorittaa. Pyyntöä varten tarvitaan *QNetworkRequest* luokasta olio, joka muodostetaan käyttämällä halutun palvelun polkua eli *QUrl* tyyppisen muuttujan sisältöä. Luokan metodilla *setHeader* voidaan asettaa ns. HTTP-otsikko eli HTTP-

header, jolla viestitetään meta-datana pyynnön yhteydessä viestin rakenne. Lopuksi pyyntö suoritetaan käyttämällä *QNetworkAccessManager* luokan *post* funktiota.

```
42
43  MainWindow::~MainWindow()
44  {
45
46  }
47
48  void MainWindow::sendRequest()
49  {
50      //save the sending object
51      QObject *senderObject = sender();
52      qDebug()<< "sending object's name : " << senderObject->objectName();
53      QString senderObjectName = senderObject->objectName();
54
55      //initalize the request
56      QUrl serviceUrl = QUrl("http://timo.devfever.com/oppari/php_pdo_example_1/main.php");
57
58      QUrlQuery query;
59
60      query.addQueryItem("action","get_press_count");
61      query.addQueryItem("button",senderObjectName);
62
63      //initialize networkaccessmanager
64      QNetworkAccessManager *networkManager = new QNetworkAccessManager();
65
66      //connect networkManager's finished request signal to readReply function to begin parsing json data
67      connect(networkManager,SIGNAL(finished(QNetworkReply*)),this,SLOT(readReply(QNetworkReply*)));
68      QNetworkRequest networkRequest(serviceUrl);
69      //set the default header
70      networkRequest.setHeader(QNetworkRequest::ContentTypeHeader,"application/x-www-form-urlencoded");
71      //make the actual request
72      networkManager->post(networkRequest,query.toString(QUrl::FullyEncoded).toUtf8());
73  }
74
```

KUVA 27. Qt post-pyyntö esimerkki, mainwindow.cpp osa 2 / 3

Kuvan 28 *readReply*-funktio käsittelee palvelimelta tulevan vastauksen. Oletuksena vastaus on *QByteArray* muodossa, joten sen jatkokäsittely on helppoa. Koska vastaus palvelimelta tulee tuttuun tapaan Json-muodossa, täytyy vastaus käsitellä, jotta todellinen arvo saadaan kaivettua esiin. Ensimmäisen luodaan *QJsonDocument*-luokasta olio, joka muodostetaan suoraan *QByteArray* tyyppisestä *answer*-muuttujasta, joka siis pitää sisällään koko vastausviestin palvelimelta. Huomaa että aiemmin esitetyn kuvan 25 riveillä ”*raw answer :* ” on printattuna koko *answer*-muuttujan sisältö. *QJsonDocument*-luokan oliosta saadaan luotua *QJsonObject* eli Json-objekti. Huomaa että palautettu viesti on siis json-objekti muodossa. Jos palautettu viesti olisi json-taulukko muodossa, tulisi käyttää tässä kohtaa *QJsonArray* luokan oliota. *QJsonValue*-luokan *value* funktiolla voidaan *QJsonObject* luokan oliosta napata arvo välittämällä sen avain, eli *key,* parametrina (vrt. *key-value pair)*. Kun viesti on käsitelty ja muutettu sopivaan muotoon, voidaan se signaloida *printReply*-funktiolle, joka tulostaa saadun arvon tekstikenttään.Tämän esimerkin avulla viimeistään huomaa kuinka hyödyllinen hyvinkin yksinkertainen PHP-rajapinta voi olla. Samaista rajapintaa voisi kutsua vastaavanlaisilla metodeilla vaikkapa Android- tai iOS-sovelluksistakin. Kyseistä tapaa käytin verkkohallinnan ja Qt-sovelluksen kommunikointiin hyvin paljon, esimerkiksi ladattaessa kampanjan

tunnuksen tai ladattaessa jonkin konfigurointitiedoston latauspolun tai lähetettäessä statistiikkapäivityksiä palvelimen tietokantaan.

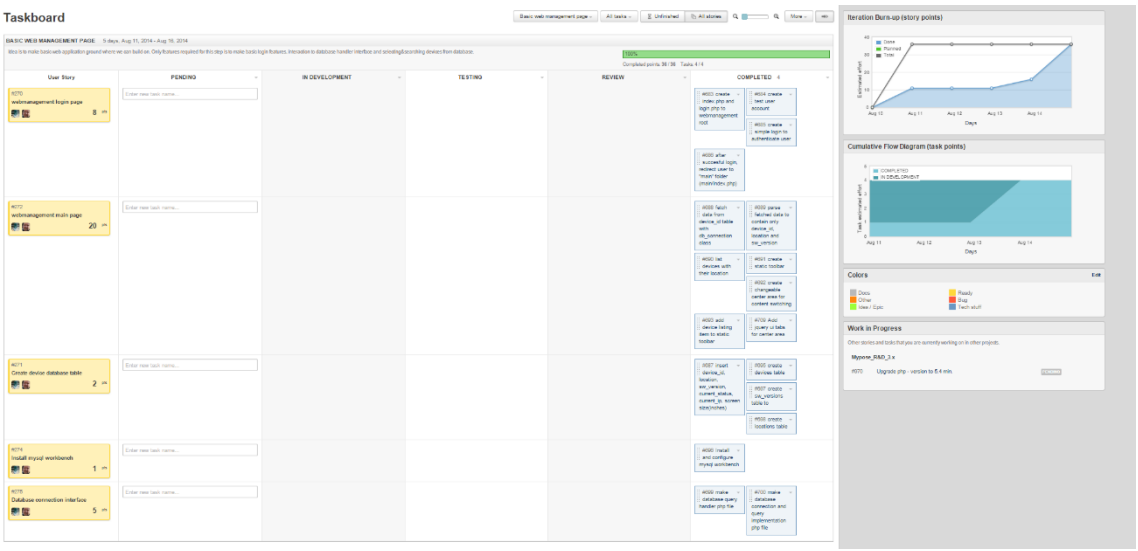
```
75  void MainWindow::readReply(QNetworkReply *reply)
76  {
77      QByteArray answer = reply->readAll();
78      qDebug() << "Raw answer : " << answer;
79      QJsonDocument document = QJsonDocument::fromJson(answer);
80      qDebug() << "Json Document : " << document;
81      QJsonObject jsonObject = document.object();
82      qDebug() << "Json Object : " <<jsonObject;
83      QJsonValue value = jsonObject.value(QStringLiteral("result"));
84      qDebug() << "Json value : " << value;
85      qDebug()<< "String value : " << value.toString();
86
87      emit replyParsed(value.toString());
88
89  }
90
91  void MainWindow::printReply(QString message)
92  {
93      qDebug()<< message;
94      mReply_label->setText("first_button's press count is : " + message);
95  }
96
```

KUVA 28. Qt post-pyyntö esimerkki, mainwindow.cpp osa 3 / 3

5 TESTAUS JA PROJEKTINHALLINTA

5.1 Projektinhallinta

Tilaajan kanssa päätimme käyttää projektinhallinnassa TinyPM-työkalua. Kyseinen työkalu oli jo Myposella käytössä entuudestaan ja sitä kerkesinkin jo käyttää ennen kuin aloitin tekemään opinnäytetyötä. Tinypm-työkalu on hyvin kevyt projektinhallintatyökalu, jolla voidaan hallinnoida tuotekehitystä. Opinnäytetyötä varten loin oman tuotteen verkkohallinnasta, johon tein oikeastaan itseä varten tehtävien jaon. Myposen omassa tuotekehityshallinnassa käytin geneerisiä taskeja, jotka koostuivat lähinnä isommista oman hallinnan tehtäväkokonaisuuksista. Näin pystyttiin sovittamaan muut työtehtävät ja opinnäytetyön tehtävät keskenään. Normaalista Scrum-menettelystä poiketen käytettiin normaalin user-storyn tilalla suurempaa tehtäväkokonaisuutta, koska se oli tässä projektissa luonnollisempaa. Kuvassa 29 on esimerkkinäkymä Tinypm-ohjelman näkymästä.



KUVA 29. Tinypm projektinhallinta, Taskboard

Sprinttejä kertyi yhteensä neljä kappaletta. Ensimmäinen sprintti koski suunnitteluvaihetta, jossa suunnittelin verkkohallinnan eri osa-alueiden arkkitehtuurit, alustavan tietokanta-arkkitehtuurin, sekä loin projektille kehitysjonon. Ensimmäisen sprintin pituus oli n. yksi viikko.

Siirryin toisessa sprintissä toteutukseen, jossa hoidin palvelimen alustuksen projektia varten. Aloitin silloin toteuttamaan PHP-rajapintaa tietokannan ja verkkohallinnan välillä. Suurin työmäärä kohdistui tässä sprintissä web-sivun tekemiseen ja eri teknologioihin tutustumiseen. Tietokannan ensimmäinen toteutus kuului myös tähän sprinttiin. Toisen sprintin kesto oli n. yksi viikko.

Kolmas sprintti oli pisin sprintti jossa toteutus eteni vauhdikkaasti. Tässä sprintissä valmistui konfiguraatiotiedostojen käsittely, tietokantojen hiominen, PHP-sovellusten lopullinen ohjelmointi, Qt-sovelluksen päivittäminen uuden verkkohallinnan mukaiseksi, sekä lukuisten muiden pienempien komponenttien ohjelmointi. Kolmannen sprintin kesto oli n. kolme viikkoa.

Viimeisessä sprintissä implementoitiin uusi graafinen ulkoasu sekä tein lopulliset viimeistelyt. Viimeisen sprintin kesto oli n. kaksi viikkoa. Tämän jälkeen projektinhallinta siirtyi osaksi Myposen pääprojektin hallintaa. Verkkohallinnan kanssa työskentely jatkuu siis edelleen, joten opinnäytetyön toteutuksen päätös oli siis neljännen sprintin loppu.

5.2 Testaus

Testaamisen suunnittelu tapahtui käytännössä edellisessä kappaleessa esitetyn projektinhallinnan kautta. Jos tehtävä koski kokonaista ohjelmoitavaa komponenttia, tein jokaiselle tällaiselle komponentille testausta varten oman tehtävän. Testattavia osia oli mm. PHP-rajapinta, laitteen valintatyökalu, uuden kampanjan luominen, Qt-sovelluksen kommunikointi palvelimen kanssa jne. Testaus koski siis opinnäytetyön toteutuksen ajan suurimmalta osin eri komponenttien yksittäisiä toimintoja. Suurimmillaan testaus koski esimerkiksi tapauksia, jossa luotiin uusi kampanja laitteelle, muokattiin laitteen asetuksia ja päivitettiin nämä Mypose-laitteelle. Testauksen avuksi tein *singleton*-tyyppisen PHP-lokisovelluksen. Sovellus kirjasi virheviestit tekstitiedostoon aikaleimalla. Viestiin sisältyi viestin lähettänyt funktio ja mahdollisesti Mypose-laitteen tunnus. Lisäksi viesteihin tallentui myös esimerkiksi tarkkailtavan muuttujan sisältö. Tämä osoittautui todella näppäräksi työkaluksi, joka nopeutti virhetilanteiden paikallistamista melkoisesti.

Lokikirjoittajan funktiokutsu voisi olla seuraavanlainen:
LogWriter::getInstance() -> writeGeneralLog("DB_Handler:: data value is $some_data_variable");

Esimerkissä oleva *LogWriter* luokan funktio *writeGeneralLog* tarkoittaa käytännössä sitä, että yleiseen lokitiedostoon kirjoitetaan funktion parametrina haluttu viesti. Funktio itsessään lisää viestiin aikaleiman ja mahdollisesti Mypose-laitteen tunnuksen, jos kysely tuli laitteen päästä. Näin lokiviestien lisääminen koodin on todella helppoa ja kutsumalla ko. luokan eri funktiota voidaan erotella erityyppiset viestit eri tiedostoihin. Esimerkiksi PDO-luokan virhesanomat voidaan tallenntaa omaan tiedostoon. Erityisesti PHP-sovellusten virheiden etsiminen voi olla työlästä, sillä jos esimerkiksi Qt-sovelluksesta lähetetty pyyntö palvelimelle ei palautakkaan mitään Qt-sovellukseen, silloin ei Qt:n oma *debug-loki* auta virheen paikallistamisessa. Virhelokin avulla pystytään paikallistamaan ongelma hyvin nopeasti. Toki esimerkiksi Firefoxin *firebug*-työkaluun on olemassa liitännäisiä, joilla PHP-koodin virheiden etsiminen onnistuu myös. Itse tehty lokikirjoittaja antaa mahdollisuuden laajentaa sitä uusilla ominaisuuksilla. Esimerkiksi kriittisen virheen tallentuessa on lokikirjoittajaan helppo lisätä vaikkapa automaattinen sähköpostinlähetys.

6 YHTEENVETO

Tämän opinnäytetyön aiheena oli suunnitella ja toteuttaa uusi verkkohallintasovellus Mypose Oy:lle. Työn tavoitteena oli saada verkkohallintasovellus uudelleenrakennettua siten, että jatkokehitys sovellukselle olisi helpompaa ja tulevaisuuden tarpeille olisi hyvä pohja, jonka päälle rakentaa lisää ominaisuuksia. Pintapuolisesti työn tavoitteena oli saada muutostyön jälkeen samat ominaisuudet, kuin oli entisessäkin sovelluksessa. Verkkohallintasovelluksen avulla mm. muokataan Mypose-laitteiden sisältöä ja luetaan statistiikka web-sivun kautta. Opinnäytetyö tuli valmistuttuaan välittömästi todelliseen käyttöön yrityksessä.

Projekti onnistui omasta mielestäni varsin hyvin, ottaen huomioon että ennen projektin aloittamista en ollut ohjelmoinut PHP-kielellä kovinkaan paljoa. Projektin aikana pääsin syventämään omia taitojani web-teknologioista, sekä tietysti C++ osaamistani Qt-ympäristössä. Opinnäytetyönä tällainen projekti oli loistava siinä mielessä, että projektin todellinen tarve yrityksessä oli hyvin selvä heti alusta asti, jolloin turhaa työtä ei tarvinnut tehdä. Projektin onnistuminen vaati erittäin monen osa-alueen yhtäaikaista hallintaa, joka kuuluu ehdottomasti tietotekniikan insinöörin taitoihin. Todellisessa työympäristössä työskenteleminen opettaa ja valmentaa insinöörimäiseen ajattelutapaan todella nopeasti, sillä paineet ja haasteet ovat todellisia. Lisäksi isojen projektien toteuttaminen on kokemuksena insinööriopiskelijalle erittäin arvokasta.

Tälle opinnäytetyölle asetetut vaatimukset täyttyivät ja jatkokehitys projektille edistyy parasta aikaa vauhdilla. Uusia vaatimuksia verkkohallintasovellukselle on tullut paljon ja monet tämänhetkiset ratkaisut ovat osoittautuneet todella hyödyllisiksi.

LÄHTEET

jQuery UI 1.11 API Documentation. 2014. jQuery. Saatavissa: http://api.jqueryui.com/.Hakupäivä 25.11.2014.

jQuery API. 2014. jQuery. Saatavissa: http://api.jquery.com/.Hakupäivä 25.11.2014.

Negus, Christopher 2009. Linux Bible 2010 Edition. Indianapolis: Wiley

Mitchell, Lorna Jane 2013. PHP Web Services. Sebastopol: O'Reilly

MySQL 5.7 Reference Manual. 2014. MySQL. Saatavissa: http://dev.mysql.com/doc/refman/5.7/en/.Hakupäivä 25.11.2014.

PHP Manual. 2014. PHP Documentation Group. Saatavissa: http://php.net/manual/en/.Hakupäivä 25.11.2014.

Rochkind, March 2013. Expert PHP and MySQL: Application Design and Development. Colorado: Apress

Stackoverflow. 2014. Stackoverlow. Saatavissa: http://stackoverflow.com/.Hakupäivä 25.11.2014.

ThemeRoller. 2014: jQuery. Saatavissa: http://jquery.com/themeroller/.Hakupäivä 25.11.2014.

LIITE 1

LÄHTÖTIETOMUISTIO

Työn tiedot

Tekijä[1]	Tilaaja[2]
Timo Asumaniemi	Mypose Oy

Tilaajan yhdyshenkilö ja yhteystiedot[3]

Lassi Anttonen, 040 516 6252

Työn nimi[4]

Verkkohallintapalvelun uudelleensuunnittelu ja toteutus

Työn kuvaus[5]

- Suunnitella Mypose Oy:n nykyinen verkkohallintapalvelu uudelleen.
- Olemassa olevan palvelun siirto tietokantoineen uudelle palvelimelle
- Uuden palvelimen konfigurointi
- Uuden palvelun vaatimusten määrittäminen sekä arkkitehtuurisuunnittelu
- Ohjelmointi palvelimelle PHP – sovelluksena.
- Implementointi Qt – sovellukseen laitteille.

Työn tavoitteet[6]

- Uuden palvelimen käyttöönotto
- Parantaa verkkohallinnan toteutusta joustavammaksi ja helpommin muokattavissa olevaksi
- Luoda riisutumpi versio admin – tasosta asiakkaan käyttöön jolloin asiakas voi tehdä pintapuolisia muutoksia laitteille itse.
- Uuden kielituen implementointi palveluun.

Tavoiteaikataulut[7]

Kesä 2014.

Päiväys ja allekirjoitukset[8]

02/06/2014 Timo Asumaniemi Tekijän allekirjoitus	02/06/2014 Lassi Anttonen Tilaajan allekirjoitus

1. Tekijän nimi, puhelinnumero ja sähköpostiosoite.
2. Työn teettävän yrityksen virallinen nimi.
3. Sen henkilön nimi ja yhteystiedot, joka yrityksessä valvoo työn suoritusta.
4. Työn nimi voi olla tässä vaiheessa työnimi, jota myöhemmin tarkennetaan.
5. Työ kuvataan lyhyesti. Siinä esitetään muun muassa työn tausta, lähtötilanne ja työssä ratkaistavat ongelmat.
6. Esitetään lyhyesti ja selvästi työn tavoitteet.
7. Esitetään projektin tavoiteaikataulu. Silloin, kun työllä on välitavoitteita, myös ne merkitään aikatauluun. Tavoiteaikataulun ja oppilaitoksen yleisaikataulun perusteella tekijä laatii oman aikataulunsa.
8. Lähtötietomuistio päivätään ja sen allekirjoittavat tekijä ja tilaajan yhdyshenkilö.