OULUN SEUDUN
AMMATTIKORKEAKOULU

Timo Haas

SIJAISJÄRJESTELMÄN JATKOKEHITYS

SIJAISJÄRJESTELMÄN JATKOKEHITYS

Timo Haas
Opinnäytetyö
Kevät 2012
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistojen kehitys

Tekijä(t): Timo Haas
Opinnäytetyön nimi: Sijaisjärjestelmän jatkokehitys
Työn ohjaaja(t): Pekka Alaluukas
Työn valmistumislukukausi ja -vuosi: Kevät 2012 Sivumäärä: 23

Tämä opinnäytetyö käsittelee Caritas-Säätiön Sijaisjärjestelmän jatkokehitysprojektia, joka tehtiin kesällä 2011. Työn tavoitteina oli lisätä järjestelmään uusina ominaisuuksina dokumenttien tulostaminen ja sähköinen tunti-ilmoitus. Dokumenteissa piti vertailla eri tekniikoita, joilla valmiita pohjia voidaan esitäyttää ja tulostaa web-pohjaisesta järjestelmästä. Tunti-ilmoituksessa tavoitteena oli nykyisen paperisen prosessin siirtäminen sähköiseen muotoon ja liittäminen osaksi sijaisjärjestelmää. Järjestelmän kautta sijaisten ja esimiesten tuli päästä tarkistamaan omia työvuorojaan sekä tekemään niihin korjauksia. Palkkakausien päätteeksi kaikki korjatut vuorot tuli saada haettua Excel-taulukossa ulos järjestelmästä.

Dokumenttien täyttäminen ja tulostaminen toteutettiin JasperReport-kirjastolla. Valmiita dokumentti-pohjia hyödynnettiin muuttamalla ne ensin kuviksi xps2img-nimisellä ohjelmalla ja upottamalla kuvat dokumenttien taustalle. Tunti-ilmoitus rakennettiin muun järjestelmän mukaisesti Grails-sovelluskehyksen päälle. Kaikki käyttöliittymät tehtiin JavaScriptillä käyttäen ExtJS-kirjastoa.

Työn tuloksena kumpikin uusi ominaisuus saatiin toteutettua ja lisättyä sijaisjärjestelmään. Sijaiset ja esimiehet pystyvät järjestelmän kautta luomaan valmiita työsopimuksia ja työtodistuksia PDF-muodossa. Tunti-ilmoitus toimii nyt kokonaan verkossa ja kaikki vuorot tarkistetaan sen kautta heti toteutumisen jälkeen. Kaikki tarkistetut vuorot voidaan hakea palkkakauden päätteeksi hallinnan kautta ulos.

Asiasanat: Sijaisrekisteri, JasperReport, Grails, ExtJS

SISÄLLYS

TIIVISTELMÄ 3
SISÄLLYS 4
1 JOHDANTO 5
2 KÄYTETYT TEKNIIKAT 6
3 ONGELMAN MÄÄRITYS 7
3.1 Paperinen tunti-ilmoitus 7
3.2 Todistukset 8
4 TUNTI-ILMOITUS 9
4.1 Tietokannat 10
4.2 Käyttöliittymät 11
4.3 ButtonColumn-plugin 14
5 DOKUMENTIT 16
5.1 Microsoft Officen xml-formaatti 16
5.2 Jasper Report 18
5.3 Ratkaisu 19
6 TESTAUS 20
7 POHDINTA 21
LÄHTEET 22
LIITE 1. ButtonColumn-luokka 23

1 JOHDANTO

Sijaisjärjestelmä on Caritas-Säätiön rekrytointiohjelma, jolla haetaan työtekijöitä lyhyisiin sijaisuuksiin. Esimiehet lisäävät järjestelmään kaikki tulevat sijaisvuorot, joita rekisteröityneet käyttäjät voivat katsella ja varata itselleen. Järjestelmä on alkujaan kehitetty harjoitteluprojektien aikana, kesällä 2010.

Työn tavoitteena oli jatkaa sijaisjärjestelmän kehitystä kahdella suuremmalla lisäyksellä, jotka tulevat helpottamaan todistusten tekemisessä ja työvuorojen tunti-ilmoituksessa. Työn ensimmäinen vaihe oli löytää sopiva tekniikka, jolla dokumentteja voidaan täyttää web-pohjaisessa järjestelmässä. Tähän oli olemassa useita eri tekniikoita, joista tämän työn osalta keskitytään JasperReport-kirjastoon ja Microsoft Officen xml-muotoon. Toinen vaihe oli toteuttaa se sijaisjärjestelmään siten, että kaikki tiedot voidaan hakea sen tietokannoista.

Työn toinen tavoite oli paperisen tunti-ilmoituksen siirtäminen sähköiseen muotoon ja liittäminen osaksi sijaisjärjestelmää. Järjestelmän kautta sijaisten ja esimiesten tuli päästä tarkistamaan omia vuorojaan verkossa ja tekemään niihin korjauksia. Palkkakauden päätteeksi kaikki korjatut vuorot toimitetaan palkanlaskentaan yhdessä listassa, josta ne voidaan helposti jatkokäsitellä. Lisäosa toteutettiin muun järjestelmän mukaisesti Grails-sovelluskehyksen päälle käyttäen ExtJS-kirjastoa käyttöliittymien toteutuksessa.

2 KÄYTETYT TEKNIIKAT

Grails on avoimeen lähdekoodiin perustuva sovelluskehys nykyaikaisten web-sovellusten rakentamiseen. Se on tehty Groovy-ohjelmointikielellä, joka pohjautuu Javaan ja on näin ollen myös yhteensopiva muiden Java-kirjastojen kanssa. Grails toteuttaa MVC-mallia jakamalla sovelluksen tietokantaan, kontrollereihin ja näkymiin. Kontrollerit käsittelevät kaikki palvelimelle tulevat pyynnöt ja palauttavat tuloksena näkymän. Tietokantojen pohjalla toimii Hibernate, jonka kautta Grailsissa voidaan käyttää melkein mitä tahansa SQL-tietokantaa. Valmiit sovellukset pakataan Java Servleteiksi, jotka voidaan sijoittaa mille tahansa niitä tukevalle palvelimelle. (Grails.)

ExtJS on JavaScript-kirjasto vuorovaikutteisten web-sivujen tekemiseen. Siinä sivut rakennetaan HTML-koodin sijaan valmiista komponenteista, joilla saadaan tehtyä monimutkaisiakin käyttöliittymiä helposti ja vaivattomasti. Kirjaston mukana tulee paljon valmiita komponentteja erilaisten dialogien, taulukoiden ja muiden rakenteiden luomiseen. Se on myös täysin yhteensopiva kaikkien käytetyimpien selainten kanssa. ExtJS:ssä on myös pyritty käyttämään mahdollisimman paljon Ajax-tekniikoita, joilla kommunikointi palvelimen kanssa voidaan tehdä taustalla ilman, että sivua täytyy ladata uudestaan. (Sencha Ext JS. 2012.)

JasperReport on avoimeen lähdekoodiin perustuva raportointityökalu, jolla voidaan luoda valmiita raportteja useissa eri formaateissa. Se on rakennettu Javan päälle eikä vaadi toimiakseen mitään ulkopuolisia kirjastoja. Jasper on alkujaan kehitetty toimimaan itsenäisenä ohjelmana, joka hakee tarvitsemansa tiedot suoraan tietokannasta. Ohjelma on jälkeenpäin muutettu kirjastoksi, jolla se voidaan sisällyttää muihin ohjelmiin ja käyttää muitakin tietolähteitä. Jasperissa dokumenteille luodaan malli XML-tyyppisellä kielellä, jossa määritellään eri kentät ja muut graafiset elementit. Pohjien perusteella Jasper osaa luoda valmiin dokumentin useissa eri formaateissa, kuten PDF, Excel, Word ja OpenOffice. (JasperReport. 2012.) Jasperiin on olemassa myös valmis editori iReport, jolla pohjia voidaan luoda ja muokata (iReport. 2012).

3 ONGELMAN MÄÄRITYS

Alustavasti Caritas-Säätiöllä oli kahdenlaisia ongelmia, joihin lähdettiin hakemaan ratkaisua tämän projektin myötä. Ensimmäinen oli tunti-ilmoitus, joka toimi aiemmin paperilla ja joka haluttiin siirtää sähköiseen muotoon. Toinen oli erilaisten dokumenttien esitäyttäminen ja tulostaminen sijaisjärjestelmän kautta. Näistä kummastakin kerrotaan seuraavassa tarkemmin.

3.1 Paperinen tunti-ilmoitus

Tunti-ilmoituksella tarkoitetaan työn toteutumisen jälkeistä prosessia, jossa tarkistetaan työvuorojen tiedot, ennen kuin ne siirtyvät maksuun. Siinä varmistetaan, että vuorot ovat toteutuneet oikein ja että työajat täsmäävät. Erilaiset erikoistapaukset kuten ylityötilanteet selvitetään samalla.

Nykyinen prosessi lähtee sillä, että sijaiset täyttävät erillisen tunti-ilmoitus lapun jokaisen maksukauden lopussa. Ilmoitukseen merkitään omat henkilötiedot sekä tiedot kaikista tehdyistä vuoroista. Jokaiseen yksikköön, missä sijainen on ollut töissä, tehdään oma lappu. Laput toimitetaan kyseisen yksikön esimiehelle, joka tarkistaa vuorojen tiedot. Tarkistamisen jälkeen laput toimitetaan palkanlaskentaan, jossa ne syötetään omiin ohjelmiinsa jatkokäsittelyä varten. Vasta kaiken tämän jälkeen voidaan palkat maksaa työntekijöille.

Nykyinen prosessi on tarkoitus automatisoida kokonaan siten, että vuorot saadaan vietyä suoraan sijaisjärjestelmästä palkanlaskentaan. Tässä työssä on kuitenkin tavoitteena päästä eroon ensimmäisestä osasta eli paperilistoista. Prosessi on tarkoitus pitää aikalailla samanlaisena, mutta siirtää sähköiseen muotoon ja liittää osaksi sijaisjärjestelmää. Uuden järjestelmän kautta sijaisten tulisi pystyä korjaamaan vuorojen tietoja niiden toteutumisen jälkeen. Samalla tavalla esimiesten tulisi päästä hyväksymään omassa yksikössä tehtyjä vuoroja. Vuoro liitetään maksukauteen vasta, kun esimies on

sen hyväksynyt. Maksukauden päätteeksi kaikki sen aikana korjatut vuorot tulisi pystyä hakemaan Excel-taulukossa ulos järjestelmästä.

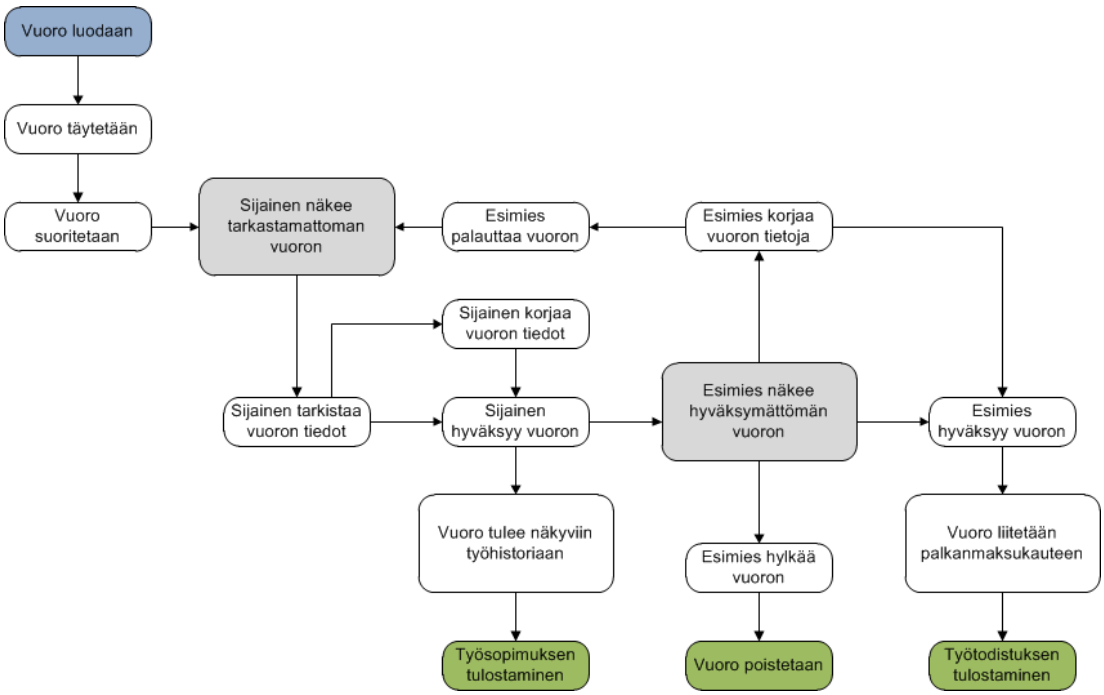
3.2 Todistukset

Jokaisesta työvuorosta joka Caritas-Säätiössä suoritetaan, joudutaan tekemään erillinen työsopimus ja työtodistus. Sijaisuudet eivät ole tästä poikkeus ja vievät tällä hetkellä paljon aikaa ja resursseja. Eri sijaisia voi olla yhtenä ajanjaksona hyvinkin paljon ja todistusten tekeminen kaikille on työlästä. Todistuksiin on olemassa valmiit pohjat, mutta ne joudutaan silti täydentämään käsin.

Todistusten tekeminen haluttaisiinkin automatisoida ainakin sijaisuuksien osalta. Sijaisjärjestelmään on tarkoitus lisätä ominaisuus, jonka kautta kaikki todistukset voitaisiin luoda automaattisesti. Järjestelmän tulisi pystyä hakemaan tarvittavat tiedot sijaisjärjestelmän tietokannoista ja täyttämään samoja dokumentti-pohjia, jotka nytkin joudutaan käsin tekemään. Valmiit todistukset voi sen jälkeen joko suoraan tulostaa tai kopioida itselle talteen. Ominaisuus tulisi myös sijaisille käyttöön, jolloin he voivat luoda omat todistukset aina tarvittaessa.

4 TUNTI-ILMOITUS

Tunti-ilmoitusta suunniteltaessa otettiin lähtökohdaksi nykyinen paperinen versio. Suunnittelu aloitettiin miettimällä kaikki eri vaiheet työvuoron näkökulmasta, miten vuoron käsittely jatkuu sen toteutumisen jälkeen aina palkanmaksuun asti. Kuvassa 1 on alustava suunnitelma tästä, jonka perusteella järjestelmää lähdettiin toteuttamaan.



KUVA 1. Alustava suunnitelma työvuorojen hyväksyntäprosessista

Työvuoron ajatellaan olevan suoritettu, kun siihen merkitty työaika on mennyt umpeen. Tämän jälkeen vuoro tulee näkyviin sijaiselle, jonka tehtävänä on tarkistaa vuoron tiedot ja korjata ne tarvittaessa. Vuoroon täytyy myös kirjata perustelut, jos työaikoihin on tullut muutoksia esimerkiksi ylityötapauksissa. Sijaisen tarkastuksen jälkeen vuoro siirtyy sen yksikön esimiehelle, jossa työ on suoritettu. Esimies tarkistaa vuoron tiedot ja joko hyväksyy tai palauttaa sen. Palautettaessa voidaan pyytää sijaiselta lisää perusteluita tai tarkennuksia vuoroon liittyen. Palautettu vuoro tulee uudelleen näkyviin sijaiselle. Vuoro siirtyy maksuun vasta, kun esimies on sen hyväksynyt. Tätä pallotte-

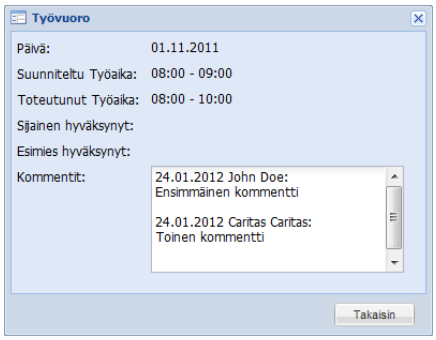
lua voidaan jatkaa tarvittaessa niin kauan, kuin vuorossa on jotain epäselvää.

4.1 Tietokannat

Seuraavaksi mietittiin millä tietokantamuutoksilla prosessi saadaan toimimaan. Kuten kuvasta 1 voikin jo päätellä, vuoro voi olla kaikkiaan neljässä eri tilassa: alkutila, sijaisen tarkistama, palautettu tai hyväksytty. Näitä tiloja kuvaamaan luotiin kaksi boolean-tyyppistä tietokenttää is_revised ja is_approved. Kentät kuvaavat sitä, onko sijainen tarkistanut vuoron ja onko esimies hyväksynyt sen. Uusissa vuoroissa kumpikin kenttä saa oletuksena arvon false tarkoittaen, että se on alkutilassa. Kenttiä muutetaan sen mukaan, kun sijainen ja esimies tarkistavat vuoron tietoja. Vuoro siirtyy maksuun vasta, kun kumpikin kenttä on saanut arvon true.

Korjatuille aloitus ja lopetusajoille lisättiin myös omat kentät nimeltä real_start_time ja real_end_time. Nämä tarvittiin sen takia, että alkuperäiset ajat säilyisivät jossakin, jos niitä tarvitaan jatkossa vuoron tietojen selvittämiseen. Kentistä näkee vuoron tämänhetkisen toteutuneen ajan, jota muutetaan aina, kun sijainen tai esimies korjaa vuoron tietoja.

Sijaisten ja esimiesten piti myös pystyä jotenkin viestimään keskenään, jos vuoron tiedoissa oli jotain epäselvyyksiä. Nämä tapaukset ovat kuitenkin aika harvinaisia, joten niihin ei lähdetty rakentamaan mitään monimutkaista ratkaisua. Keskustelua varten luotiin uusi longtext-tyyppinen tekstikenttä, johon kummankin kommentit tallennettiin. Jokaisen kommentin väliin lisättiin erottava rivinvaihto sekä henkilön nimi ja aika, jolloin kommentti on lisätty (kuva 2).



KUVA 2. Työvuoron dialogi, johon on lisätty kaksi kommenttia

Vuorot piti lopuksi vielä yhdistää meneillään olevaan maksukauteen. Tätä varten luotiin approval_date-niminen aikaleima kenttä. Siihen tallennetaan aika, jolloin vuoron tiedot on tarkistettu ja vuoro siirtynyt maksuun. Oikea maksukausi voitiin selvittää vertaamalla tätä kenttää maksukauden ensimmäiseen ja viimeiseen päivään. Jos vuoro oli hyväksytty näiden välissä, kuului se kyseiseen maksukauteen (kuva 3).

	Field Name	Datatype	Len	Default	PK?	Not Null?	Unsigned?	Auto Incr?	Zerofill?
	approval_date	datetime			☐	☐	☐	☐	☐
	comment	longtext			☐	☐	☐	☐	☐
	is_approved	bit	1		☐	☐	☐	☐	☐
	is_revised	bit	1		☐	☐	☐	☐	☐
	real_end_time	time			☐	☐	☐	☐	☐
	real_start_time	time			☐	☐	☐	☐	☐

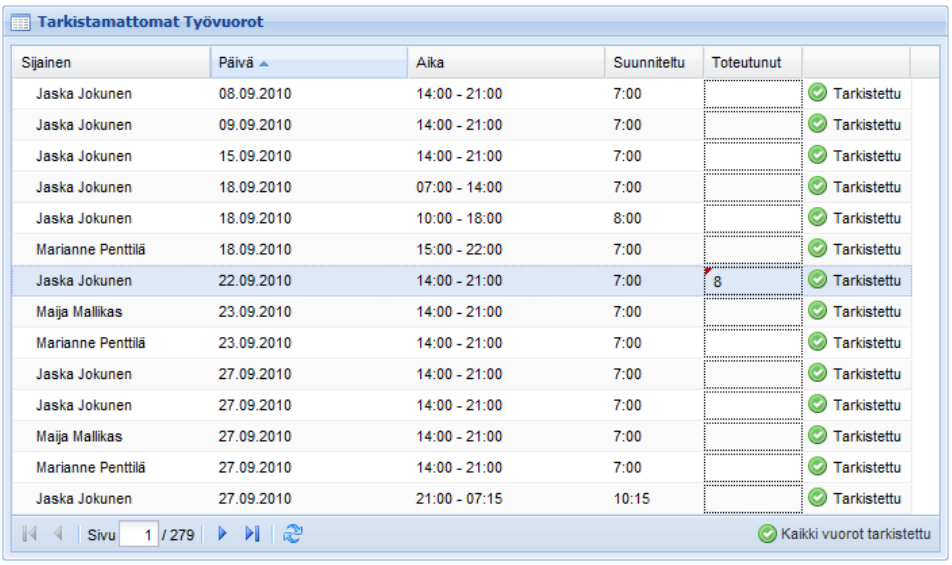
KUVA 3. Uudet kentät tietokannassa

4.2 Käyttöliittymät

Käyttöliittymien lähtökohtana oli tehdä selkeä, helppokäyttöinen sekä tehokas sivu päivittäiseen käyttöön. Sivun piti olla sen verran yksinkertainen, että ensikertalainenkin ymmärtää sen toiminnan. Samalla myös tehokas siinä mielessä, että käsiteltäviä vuoroja voi hyvinkin tulla paljon ja niiden läpikäyminen täytyy olla nopeaa ja vaivatonta.

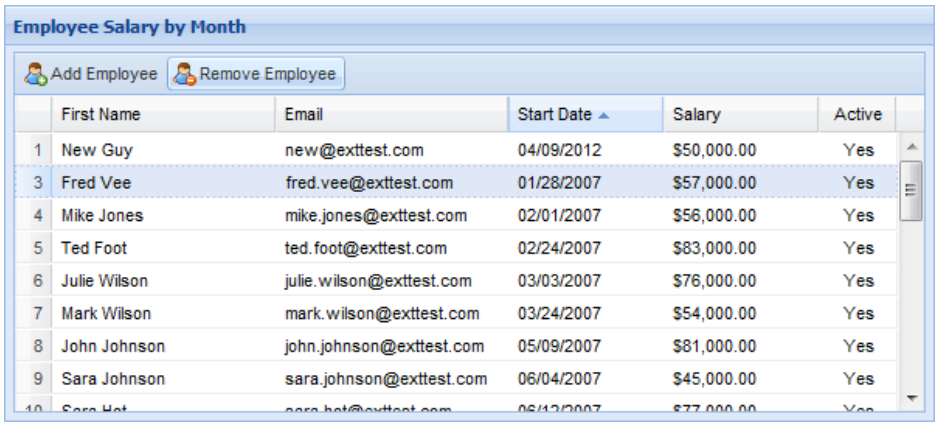
Suunnitelmissa päädyttiin tekemään kokonaan uudet sivut tunti-ilmoitusta varten. Ne olisi voitu lisätä olemassa oleville sivuillekin, mutta erillinen sivu oli selkeämpi ratkaisu, varsinkin jos ominaisuus joudutaan joskus tulevaisuudessa poistamaan järjestelmästä. Sekä sijaisille että esimiehille rakennettiin

omat sivut, joissa he näkivät listalla kaikki heitä koskevat vuorot. Listan oikeaan laitaan lisättiin toiminnot, mitä käyttäjä voi vuoroon liittyen tehdä. Näitä olivat esimerkiksi tarkistaminen, korjaaminen ja hyväksyminen. Vuorot järjesteltiin maksukausittain, jolloin samasta näkymästä pystyi selailemaan vanhojakin vuoroja. Kuvassa 4 on ensimmäinen versio tunti-ilmoitus-näkymästä.



KUVA 4. Ensimmäinen versio esimiesten tunti-ilmoitus-näkymästä, jossa toiminnot on tehty RowActions-lisäosalla

Suurimmaksi ongelmaksi sivuja tehdessä tuli erilaisten toimintojen lisääminen taulukkoon. Jokaiseen riviin eli työvuoroon piti jotenkin liittää erilaisia toimintoja, joita käyttäjä voi siinä tehdä. ExtJS ei kuitenkaan tue tätä ollenkaan vaan sen tapa toteuttaa vastaavanalainen ominaisuus on lisätä toiminnot taulukon ylätunnisteeseen (kuva 5). Vuoroja hyväksyttäessä olisi siis ensin pitänyt valita vuoro ja sen jälkeen toiminto ylätunnisteesta. Tämä olisi vaatinut ylimääräisiä napsautuksia ja ollut aivan liian vaivalloista varsinkin, jos vuoroja on paljon.

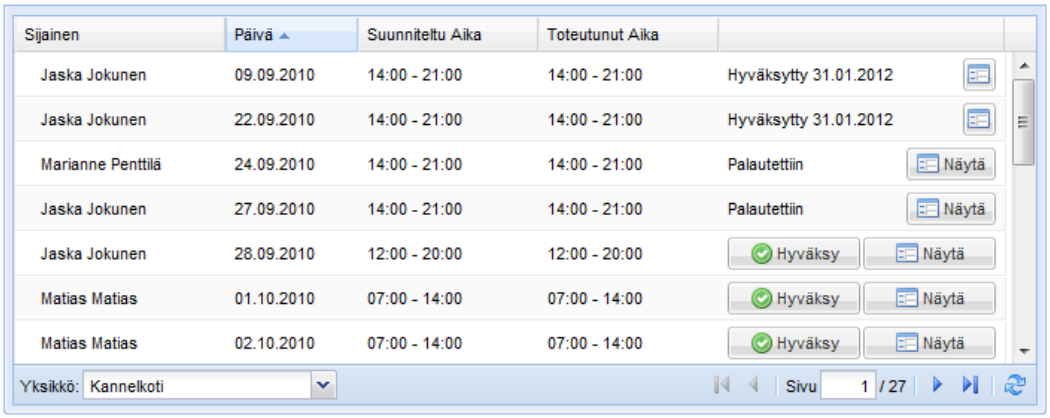


KUVA 5. ExtJS-esimerkki taulukosta, jossa toiminnot ovat ylätunnisteessa

ExtJS:n on olemassa RowActions-niminen lisäosa joka on kehitetty juuri tätä ongelmaa varten. Siinä taulukon yhteen sarakkeeseen saadaan lisätty niin sanottuja actioneita jotka toimivat painonappien tapaan. Jokainen action koostuu kuvakkeesta ja tekstistä jotka piirretään taulukon yhden solun sisälle. Niihin voidaan liittää erilaisia toimintoja jotka suoritetaan sitä napsautettaessa. Actioneita voi tarvittaessa olla useampiakin joista jokaiseen saadaan liitettyä eri toiminto. (Ext.ux.grid.RowActions Plugin by Saki - ver.: 1.0. 2008.)

Sivun ensimmäisessä versiossa toiminnot lisättiin taulukkoon RowActions-lisäosalla (kuva 4). Jälkeenpäin todettiin kuitenkin että ne olivat huono vaihtoehto toimintojen lisäämiseen. Kuten kuvasta 4 näkeekin, ne eivät anna käyttäjälle oikeaa kuvaa painonapista vaan näyttävät ennemmin taulukon tietokentiltä. Actioneiden ulkoasua ei pystynyt muokkaamaan jälkeenpäin joten niitä ei voitu korjata silläkään. Käyttöliittymän saaminen selkeäksi oli kuitenkin sen verran tärkeä asia että tähän lähdettiin hakemaan jotain muuta ratkaisua.

Ratkaisuna toimintojen lisäämiseen kehitettiin oma plugin josta on kerrottu tarkemmin seuraavassa luvussa. Pluginin avulla taulukkoon saatiin upotettua normaaleja button-komponentteja jotka näyttivät paljon selkeämmiltä ja antavat käyttäjälle heti oikean kuvan toiminnosta. Samaan sarakkeeseen saatiin näkymään myös tekstiä jolla annetaan lisätietoa vuoron tilaan liittyen (kuva 6).



KUVA 6. Tunti-ilmoitus-näkymä, jossa on käytetty ButtonColumn-ratkaisua

4.3 ButtonColumn-plugin

ExtJS:n taulukoihin ei saada suoriltaan upotettua mitään muita komponentteja, koska ne on suunniteltu näyttämään vain tekstiä. Tunti-ilmoitus-näkymässä ja muutakin sivustoa tehdessä on kuitenkin tullut vastaan tilanteita, jossa taulukkoon pitäisi pystyä liittämään painonappeja tai muita komponentteja. Yleensä nämä on jouduttu ratkaisemaan jollain muulla tapaa, suunnitellen käyttöliittymät eri lailla tai käyttäen jotain lisäosaa. Tavoitteeksi ButtonColumn-luokkaa tehdessä otettiinkin saada kaikki ExtJS:n komponentit toimimaan taulukossa.

ExtJS:ssä on olemassa erityinen Container-komponentti joka toimii muiden komponenttien säiliönä. Sen sisälle voidaan lisätä kaikkia muita komponentteja joiden toiminnasta Container huolehtii täysin. Jos Container-komponentti saataisiin toimimaan taulukossa, toimisivat siinä sen jälkeen kaikki muutkin komponentit.

Container saatiin näkymään taulukossa suoraan lisäämällä siitä saatu html-koodi taulukon soluun. Erilaiset tapahtumat eivät kuitenkaan toimineet suoraan, vaan ne jouduttiin itse välittämään taulukolta komponenteille. Nämä saatiin kuitenkin toimimaan helposti liittämällä taulukon tapahtumiin käsittelijät ja välittämällä ne itse. Kaikki muutokset pakattiin lopuksi vielä omaan pluginiin, josta sitä on helppo käyttää muissakin taulukoissa. Pluginin koodit ovat saatavilla liitteessä 1.

ButtonColumn-pluginia käytetään luomalla siitä uusi instanssi ja määrittämällä sille handler-funktio. Handler-funktiota kutsutaan jokaisen taulukon rivin kohdalla ja sen tehtävänä on palauttaa siinä näkyvät komponentit (kuva 7). Instanssi lisätään taulukon sarakkeisiin samalleen kuin kaikki muutkin. Lisäksi se pitää antaa myös pluginina taulukolle jotta kaikki tapahtumat saadaan välitettyä taulukon ja komponenttien välillä (kuva 8).

```
1.  var buttonColumn = new Ext.ux.ButtonColumn(
2.  {
3.    // Handleria kutsutaan kerran jokaista riviä kohden. Sen tehtävänä on
4.    // luoda komponentit kyseiselle riville ja palauttaa ne taulukossa.
5.    handler: function(record, rowIndex, colIndex, store)
6.    {
7.      // Tässä vaiheessa voidaan lukea taulukon muita tietoja
8.      // sekä muuttaa komponentteja sen mukaan.
9.      return [{
10.       xtype   : 'button',
11.       text    : 'Hyväksyn',
12.       iconCls : 'silk-accept',
13.       handler : function(it, event) { }
14.     }, {
15.       xtype   : 'button',
16.       text    : 'Korjaa',
17.       iconCls : 'silk-application-form',
18.       handler : function(it, event) { }
19.     }]
20.   }
21. });
```

KUVA 7. Esimerkki ButtonColumnin luomisesta ja asetuksista

```
1.  var grid = new Ext.grid.GridPanel(
2.  {
3.    colModel: new Ext.grid.ColumnModel({
4.      columns: [
5.        { header: 'Päivä', dataIndex: 'date' },
6.        { header: 'Aika',    dataIndex: 'time' },
7.        { header: 'Yksikkö', dataIndex: 'unit' },
8.        buttonColumn // Sarake lisätään muiden tavoin taulukkoon
9.      ]
10.   }),
11.
12.   plugins: [
13.     buttonColumn   // Sarake pitää lisätä myös pluginina taulukkoon
14.                    // jotta piirtäminen ja päivitys toimivat.
15.   ]
16. });
```

KUVA 8. Esimerkki ButtonColum-pluginin liittämisestä taulukkoon

5 DOKUMENTIT

Järjestelmästä tuli saada tulostettua työsopimuksia ja työtodistuksia. Kumpaankin oli olemassa valmiit pohjat Caritas-Säätiöltä, jotka piti esitäyttää tulostamisen yhteydessä. Dokumentteihin tuli saada täytettyä kyseisen sijaisen henkilötiedot sekä lista suoritetuista tai varatuista työvuoroista. Ratkaisua tähän haettiin monesta eri suunnasta, joista kerrotaan seuraavassa tarkemmin.

5.1 Microsoft Officen xml-formaatti

Koska dokumenttien pohjat olivat Microsoft Officen doc-tyyppisiä, lähettiin ratkaisua etsimään aluksi sen kautta. Officen kaikkiin ohjelmiin on olemassa kaksi tallennusmuotoa. On olemassa oletuksena käytetty binäärinen sekä xml-formaatti. Oletuksena kaikki tallennetaan binäärisessä muodossa, joka on paljon pienempi kooltaan ja nopeampi käsitellä. Kaikki dokumentit saadaan myös tallennettua xml-muodossa, jota on helpompi lukea ja käsitellä ohjelman ulkopuolella. Formaatit eroavat siinä, että xml tallennetaan selkeänä tekstinä, jota voidaan helposti lukea ja muokata tekstinkäsittelyohjelmalla. Binääristä ei taas voida lukea ilman, että tiedetään tarkalleen, miten se on tallennettu. Kuvassa 9 on esimerkki xml-formaatista, joka on otettu työsopimus-pohjasta.

```
<w:p w:rsidR="00405074" w:rsidRDefault="002F4D20">
    <w:pPr>
        <w:rPr>
            <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>
            <w:i/>
            <w:spacing w:val="10"/>
            <w:sz w:val="28"/>
        </w:rPr>
    </w:pPr>
    <w:r>
        <w:rPr>
            <w:rFonts w:ascii="Arial" w:hAnsi="Arial"/>
            <w:i/>
            <w:spacing w:val="10"/>
            <w:sz w:val="28"/>
        </w:rPr>
        <w:t>Caritas-Säätiö</w:t>
    </w:r>
</w:p>
```

KUVA 9. Esimerkki xml-formaatista, joka on otettu työsopimus-pohjasta

Käytetty Grails-sovelluskehys osaa suoraan lukea ja käsitellä xml-muotoista dataa, joten dokumentit saatiin helposti avattua ja muokattua siinä. Suurin ongelma tässä ratkaisussa oli kuitenkin oikeiden kenttien löytäminen tiedostosta. Osaan kentistä oli lisätty ympärille bookmark-tagit, jolloin ne voitiin hakea sen avulla (kuva 10). Suurimmasta osasta kentistä nämä kuitenkin puuttuivat kokonaan, joten niiden löytäminen olisi ollut hankalaa tai miltei mahdotonta.

```
<w:bookmarkStart w:id="0" w:name="kbCompName_1"/>
<w:r>
    <w:instrText xml:space="preserve"> FORMTEXT </w:instrText>
</w:r>
<w:r>
    <w:fldChar w:fldCharType="separate"/>
</w:r>
<w:r>
    <w:rPr>
        <w:rFonts w:ascii="Lucida Console" w:hAnsi="Lucida Console"/>
    </w:rPr>
    <w:t>Caritas-Säätiö</w:t>
</w:r>
<w:r>
    <w:fldChar w:fldCharType="end"/>
</w:r>
<w:bookmarkEnd w:id="0"/>
```

KUVA 10. Bookmark-tagit xml-tiedostossa. Kuvassa on työnantaja-kenttä työsopimus-pohjasta

Vaikka tarvittavat kentät olisikin löydetty ja halutut tiedot saatu täytettyä, olisi tiedosto silti pitänyt vielä muuttaa takaisin binääriseen muotoon ennen kuin se annetaan käyttäjälle. Ne olisi voitu antaa myös xml-muodossa, mutta tämä olisi saattanut aiheuttaa ongelmia vanhemmissa ohjelmissa. Muuntamiseen ei kuitenkaan löydetty mitään valmista ohjelmaa tai kirjastoa, joten se olisi ollut miltei mahdotonta toteuttaa.

Ratkaisun hyviä puolia:

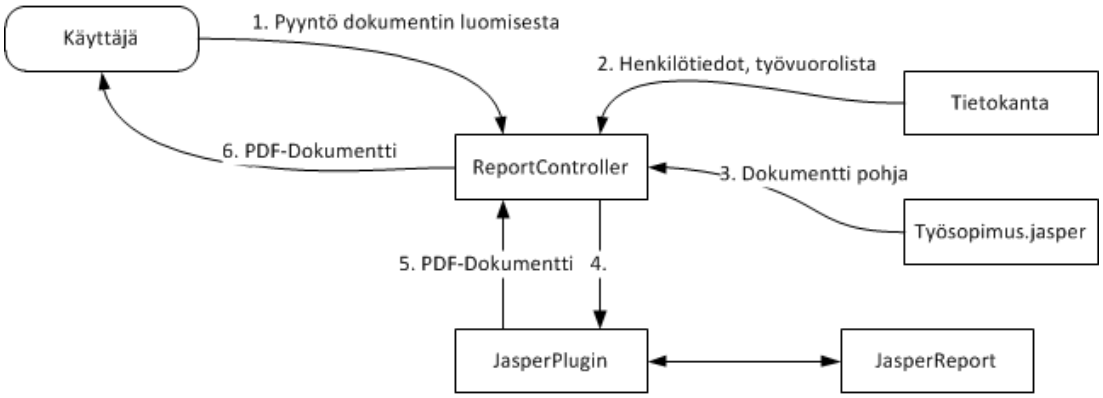
- Pohjaa olisi voinut muuttaa jälkeenpäin Wordissa.

Huonoja puolia:

- Tiedosto olisi pitänyt antaa käyttäjille xml-formaatissa.
- Erilaisten listojen ja muiden rakenteiden lisääminen on hankalaa.

5.2 Jasper Report

Jasperissa dokumenteille luodaan pohja xml-tyyppisellä kielellä, jossa määritellään kaikki kentät ja muut graafiset elementit. Pohjien luomiseen on olemassa valmis editori iReport, joka helpottaa niiden tekemisessä (iReport. 2012). Pohjat käännetään ennen käyttöä Jasperin omaan muotoon, jossa varmistetaan, että kaikki tietolähteet löytyvät eikä kenttiin ole tullut mitään kirjoitusvirheitä. Kääntämisen jälkeen voidaan pohjalla ja annetulla datalla luoda valmiita dokumentteja, jotka voidaan suoraan antaa käyttäjälle (kuva 11).



KUVA 11. Dokumentin luomisprosessi JasperReport-kirjaston avulla

Grailsiin on olemassa myös valmis lisäosa jolla Jasperin saa liitettyä siihen suoraan. Lisäosassa tulee valmiina kontrolleri jonka kautta dokumentteja pystyy luomaan ja lataamaan. Se auttaa myös tietojen välittämisessä Grailsin ja Jasperin välillä. (Hohns, Sebastian 2012.)

Ainut heikkous Jasperissa oli, ettei siinä ei ollut minkäänlaista työkalua olemassa olevien pohjien hyödyntämiseen. Kumpikin dokumentti olisi näin ollen pitänyt rakentaa kokonaan uudestaan Jasperin omassa muodossa. Muistakaan vastaavanlaisista ohjelmista tätä ominaisuutta ei löytynyt. Pohjien rakentaminen kokonaan uudestaan olisi ollut aivan liian suuri työ tämän opinnäytetyöntyön puitteissa, joten sitä ei lähdetty edes yrittämään.

Jasperin hyviä puolia:

- Dokumentit saatiin useissa eri formaateissa.
- Dokumenttien rakennetta pystyy jälkeenpäin muokkaamaan ilman että järjestelmään täytyy tehdä muutoksia.
- Valmis lisäosa Grailsiin joka helpottaa kirjaston käyttämistä.

Huonoja puolia:

- Valmiita pohjia ei pystynyt hyödyntämään mitenkään.

5.3 Ratkaisu

Lopulta päädyttiin kuitenkin käyttämään JasperReportia dokumenttien luomisessa, koska siinä oli kaikki ominaisuudet mitä haettiinkin ja hyvät mahdollisuudet jatkokehitystä varten. Pohjia pystyi helposti muuttamaan mukana tulevalla editorilla ja päivittämään sivulle jälkeenpäin. Dokumentteihin pystyi myös lisäämään ja poistamaan jälkikäteen kaikkia tietokannan tietoja ilman mitään muutoksia järjestelmään.

Valmiita pohjia hyödynnettiin poistamalla niistä ensin kaikki kentät jotka täytetään Jasperissa. Pohjat muutettiin sen jälkeen kuviksi, jotka voitiin upottaa dokumenttien taustalle. Kaikki täytettävät kentät ja muut elementit lisättiin kuvien päälle. Näin dokumenteista saatiin valmiiden pohjien näköisiä ilman että niitä olisi jouduttu rakentamaan kokonaan uudestaan. Jatkossa pohjat voidaan muuttaa kokonaan Jasperin muotoon jos tarvitaan mutta tällä hetkellä nykyinen ratkaisu näyttää riittävän.

Word-dokumenttien muuttamisessa kuviksi käytettiin xps2img-nimistä ohjelmaa (xps2img. 2012). Muuntamista varten pohjat täytyi ensin tallentaa xps-muotoon joka on eräänlainen tulostamiseen käytetty formaatti. Tallentaminen onnistuu suoraan Microsoft Wordista tulostamalla dokumentti johonkin tiedostoon. Tämän jälkeen saatu xps-tiedosto muutettiin kuvaksi xps2img-ohjelmalla, josta tuloksena saatiin valmis jpeg-kuva.

6 TESTAUS

Kaikki uudet ominaisuudet täytyi myös testata perusteellisesti ennen käyttöönottamista. Testaus haluttiin kuitenkin suorittaa tulevilla käyttäjillä eli sijaisilla ja esimiehillä. Oikeiden työvuorojen saaminen testiin oli myös tärkeää jotta mahdolliset käytännön ongelmat huomattaisiin ja saataisiin korjattua.

Testaaminen olisi voitu suorittaa suoraan tuotantoversiossa, lisäämällä sinne uudet ominaisuudet ja rajaamalla pääsy vain testiryhmälle. Tässä ratkaisussa olisi kuitenkin kaikki muutokset pitänyt lisätä tuotantoympäristöön, jotka olisivat voineet pahimmassa tapauksessa kaataa koko järjestelmän. Tuotantoversiota ei haluttu kuitenkaan vaarantaa mitenkään joten tästä ratkaisusta luovuttiin. Ominaisuuksien lisääminen ja rajaaminen olisi aiheuttanut myös ylimääräistä työtä. Testikäyttäjiä olisi tässä ratkaisussa voitu ottaa paljon enemmän ja testaaminen suorittaa monilla eri päätelaitteilla.

Koska testaaminen piti suorittaa joka tapauksessa verkossa, rakennettiin tuotantoversion rinnalle erillinen testiversio järjestelmästä. Testaaminen suoritettiin maksukausittain kopioimalla jokaisen kauden välissä uudet tiedot tuotantoversiosta testiversioon. Näin päästiin testaamaan ajantasaisilla tiedoilla mutta kuitenkin erillään tuotantoversiosta. Koska kaikki vuorot olivat oikeita, saatiin hyvin todellinen kuva siitä, miten järjestelmä tulee toimimaan tuotannossa.

Itse testaamisen suoritti Caritas-Säätiö omissa tiloissaan kutsumalla paikalle joitakin sijaisia ja esimiehiä joilla oli paljon vuoroja testattavina maksukausina. Testaajat pääsivät kirjautumaan testiversioon omilla tunnuksillaan ja testaamaan järjestelmää tutuilla vuoroilla. Koska testaaminen tapahtui valvottuna, saatiin testeistä myös hyvää palautetta käytettävyyden kannalta.

7 POHDINTA

Työssä oli tavoitteena lisätä sijaisjärjestelmään dokumenttien tulostaminen ja sähköinen tunti-ilmoitus. Tuloksena kumpikin uusi ominaisuus saatiin toteutettua ja liitettyä osaksi muuta järjestelmää. Ne ovat tällä hetkellä tuotantokäytössä ja toimineet hyvin.

Sijaisjärjestelmän kautta pystytään nyt luomaan sijaisille valmiita työsopimuksia ja työtodistuksia napin painalluksella. Ominaisuus on vähentänyt henkilökunnan työmäärää huomattavasti ja innostanut myös sijaisia huolehtimaan omista todistuksistaan. Haastavinta tässä työssä oli löytää ratkaisu, joka toimisi nykyisessä järjestelmässä ja jossa valmiita dokumentti-pohjia voitaisiin hyödyntää parhaiten. Nykyinen ratkaisu ei ole mitenkään täydellinen mutta paras mitä projektin aikarajoissa ehdittiin tehdä. Jatkossa dokumentti-pohjia voisi vielä parannella siirtämällä ne kokonaan Jasperin muotoon. Järjestelmään saatiin kuitenkin hyvä pohja valmiiksi, jota on helppo laajentaa tulevaisuudessa.

Tunti-ilmoitus on nyt siirretty kokonaan sähköiseen muotoon ja liitetty sijaisjärjestelmään. Vuorot tarkistetaan sen kautta heti niiden toteutumisen jälkeen ja palkanlaskentaan saadaan aina ajantasaiset listat tehdyistä vuoroista. Käytetyt tekniikat olivat jo ennalta tuttuja, joten niiden osalta ei ollut sen kummempia ongelmia. Eniten aikaa työssä meni käyttöliittymien suunnitteluun ja toteutukseen. Työn ohessa luotiin myös hyvä lisäosa ExtJS-kirjastoon, joka laajentaa sen taulukoiden käytettävyyttä erilaisissa projekteissa.

LÄHTEET

Ext.ux.grid.RowActions Plugin by Saki - ver.: 1.0. 2008. Saatavissa: http://rowactions.extjs.eu. Hakupäivä 1.2.2011.

Grails. Saatavissa: http://grails.org/. Hakupäivä 24.4.2012.

Hohns, Sebastian 2012. jasper plugin. Saatavissa: http://grails.org/plugin/jasper/. Hakupäivä 1.2.2011.

iReport. 2012. Saatavissa: http://sourceforge.net/projects/ireport/. Hakupäivä 25.4.2012.

JasperReport. 2012. Saatavissa: http://jasperforge.org/projects/jasperreports/. Hakupäivä 5.3.2011.

Sencha Ext JS. 2012. Saatavissa: http://www.sencha.com/products/extjs/. Hakupäivä 24.4.2012.

xps2img. 2012. Saatavissa: http://sourceforge.net/projects/xps2img/. Hakupäivä 5.3.2011.

```
1.  Ext.namespace('Ext.ux');
2.
3.  Ext.ux.ButtonColumn = Ext.extend(Ext.grid.Column,
4.  {
5.    colIndex: 0,
6.
7.    cols: {},
8.
9.    items: [],
10.
11.   init: function(grid) {
12.     grid.on('afterlayout', function(it, layout) {
13.       for(var i in this.cols) {
14.         this.cols[i].doLayout();
15.       }
16.     }, this);
17.
18.   grid.getView().on('beforerefresh', function(it, layout) {
19.     for(var i in this.cols) {
20.       this.cols[i].destroy();
21.     }
22.     this.cols = {};
23.   }, this);
24.
25.   grid.getView().on('refresh', function(it, layout) {
26.     for(var i in this.cols) {
27.       this.cols[i].render(
28.         grid.getView().getCell(i, this.colIndex).firstChild);
29.       }
30.     }, this);
31.   },
32.
33.   initComponent: function() {
34.     Ext.ux.ButtonColumn.superclass.initComponent.apply(this, arguments);
35.   },
36.
37.   renderer: function(value, metaData, record, rowIndex, colIndex, store) {
38.     this.colIndex = colIndex;
39.
40.     this.cols[rowIndex] = new Ext.Container({
41.       layout: 'hbox',
42.       layoutConfig: {
43.         align: 'middle'
44.       },
45.       defaults: {
46.         margins: '0 2 0 0'
47.       },
48.       items: this.handler(record, rowIndex, colIndex, store)
49.     });
50.
51.     return(' ');
52.   },
53.
54.   handler: function(record, rowIndex, colIndex, store) {
55.     return this.items;
56.   }
57. });
58.
59. Ext.reg('buttoncolumn', Ext.ux.ButtonColumn);
```