Joni Lappalainen

Työvuorojen suunnittelu -verkkosovellus

Metropolia Ammattikorkeakoulu
insinööri (AMK)
tietotekniikka
Insinöörityö
15.11.2010

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Joni Lappalainen Työvuorojen suunnittelu -verkkosovellus 30 sivua + 2 liitettä 11.11.2010
Tutkinto	insinööri (AMK)
Koulutusohjelma	tietotekniikka
Suuntautumisvaihtoehto	ohjelmointi
Ohjaaja	Yliopettaja Kirsti Äystö

Insinöörityön tavoitteena oli suunnitella ja toteuttaa yritykselle web-tietokantasovellus. Sovelluksen oli tarkoitus helpottaa työvuorojen suunnittelua ja siihen liittyvien tietojen hallintaa. Yrityksen kasvun ja siitä aiheutuvien lisätöiden takia yrityksen täytyi löytää edullisempi vaihtoehto hallita ja suunnitella työvuoroja. Työvuorojen suunnittelu oli aikaisemmin toteutettu kynällä, paperilla ja tekstinkäsittelyohjelmalla.

Työvuorojen suunnittelun lisäksi sovelluksessa otettiin huomioon siihen liittyvät asiat, kuten työntekijät, työpisteet, palkat ja viestintä. Työssä perehdyttiin tarkemmin myös Ruby on Rails -web-sovelluskehykseen ja muihin työssä käytettyihin menetelmiin.

Sovelluksen toteuttamiselle annettiin aikaa noin neljä kuukautta. Sovellusta tultaisiin käyttämään kesäisin yrityksen ollessa aktiivinen ja tarkoitus oli saada sovellus käyttöön jo tulevana kesänä.

Sovellus helpotti työvuorojen suunnittelutyötä ja vaadittava työmäärä väheni. Ruby on Rails osoittautui hyväksi menetelmäksi kehittää tietokanta -pohjaisia web -sovelluksia.

Avainsanat	työvuorot, verkkosovellus, verkkopalvelu, Ruby on Rails

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	Joni Lappalainen
Title	A web application for planning employees' shifts
Number of Pages	30
Date	11 November 2010
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Kirsti Äystö

The goal of this work was to develop a web database application for a company. The purpose of the developed application was to ease the amount of work of the work shift planning and the management of related concerns. It was the growth of the company and the extra work caused by the growth that set the need for more efficient management of work shifts. Planning work was previously done with a pencil, paper and word processing software.

In addition to the issue of planning work shifts there were related issues like employees, stores, salaries and communication. In this work a more detailed look was taken into the Ruby on Rails framework and other techniques that were used.

A time allocation of approximately four months was given to design and implement the application. The application was going to be used during the summers when the company is active and it should be already in use in upcoming summer.

The application succeeded in easing the amount of work. Ruby on Rails proved to be a great framework to develop this kind of database based web applications.

Keywords	work shifts, web application, internet, Ruby on Rails

Lyhenteet, käsitteet ja määritelmät

Ajax	Ryhmä tekniikoita, jotka mahdollistavat asynkronisen tietojen käsittelyn palvelimen ja asiakaspään välillä ilman ylimääräisiä sivulatauksia.
CSS	Cascading style sheets. Tyylien kuvauskieli, jolla kuvataan tyylejä pääasiassa HTML ja XML -dokumenteille.
DOM	Document Object Model. Html-dokumentin puumalli.
Git	Versionhallintajärjestelmä, jolla ylläpidetään versiopuuta projektista ja sen tiedostoista.
HTML	HyperText Markup Language on merkkikieli, jolla kuvataan web-sivuja selaimille.
HTTP	Hypertext Transfer Protocol on tietoliikenne-protokolla, jota käytetään asiakkaan ja serveripään kommunikoinnissa.
HTTPS	Hypertext Transfer Protocol Secure. HTTP-protokolla, jossa käytetään SSL/TLS-protokollaa yhteyden suojaamiseksi.
Javascript	Skriptauskieli, jota käytetään asiakaspuolella käyttöliittymien skriptauksessa.
RIA	Rich Internet Application on web-sovellus, jossa suuri osa logiikasta on toteutettu asiakkaan puolella selaimessa.
RJS	Ruby on Railsin sapluunakieli, jolla kirjoitetaan Javascriptiä Rubylla.
SOAP	Simple Object Access Protocol on protokolla, jolla voidaan lähettää ja pyytää rakenteellista tietoa web-palveluissa.
SSL/TLS	Transport Layer Security on salausprotokolla, jolla voidaan suojata TCP -protokollan yhteyksiä.

Sisällys

1 Johdanto

Asiakas ja tilaaja on yritys, joka toimii kesäisin marjojen ja vihannesten vähittäiskauppiaana Pohjois-Suomessa. Yrityksen työntekijät koostuvat pääosin paikallisista nuorista kesätyöläisistä. Työntekijöiden määrä on noin 30 - 40. Kaikki työntekijät palkataan tuntiperusteisesti. Yritys joutuu hakemaan joka kesä suurimman osan työntekijöistä uudestaan ja suunnittelemaan aktiivisesti työvuorot viikoittain. Työvuorot saattavat muuttua säätilan tai tapahtumien takia yllättäen. Työvuorolistojen ja työntekijöiden täytyy pysyä ajan tasalla muutoksista.

Selkeä tarve yritykselle on sujuvampi ja helpompi tapa hallita työvuoroja ja viestiä työntekijöiden ja työvuorojen suunnittelijan välillä. Myös työntekijöiden tietojen hallintaan tarvitaan automatisointia. Yksi toistuvimmista työvaiheista on työaikojen suunnittelu ja siihen liittyvät asiat, kuten toteutuneet työajat ja palkkojenlaskeminen. Työajat täytyy suunnitella, työntekijöille tulostaa vuorolistat ja merkitä toteutuneet ajat, joiden perusteella lasketaan palkka. Tuntilistat ja tarvittavat työntekijöiden tiedot annetaan palkanlaskijalle, joka kirjoittaa tietojen perusteella palkkakuitin, jonka sitten lopulta työntekijäkin saa.

Aikaisemmin yritys oli tehnyt työaikojen suunnittelun paperilla ja kynällä ja tekstinkäsittely-ohjelmilla. Ylimääräistä työtä olivat tietojen kopioiminen, ylläpito, työaikojen ristiinvertailu ja laskeminen. Palkkojen laskemisessa inhimilliset virheet olivat lisäksi mahdollisia. Työvuorojen tulostamista varten kaikille työntekijöille täytyi kirjoittaa omat vuorolistansa.

Tavoitteena on tehdä web-sovellus, jolla yritys voi hallita työntekijöiden tietoja sekä suunnitella työvuorot, tulostaa työvuorolistat, lähettää viestejä työntekijöille ja laskea ja tulostaa palkkakuitit. Sovellus toisi tehokkuutta ja tarkkuutta aikaisempaan menetelmään verrattuna. Sovelluksella pyritään nopeuttamaan koko prosessia vähentämällä työmäärää ja virheriskejä.

Työ rajataan sovelluksen määrittelyyn, suunnitteluun, toteutukseen ja asennukseen. Sovelluksen ylläpidosta tulee vastaamaan tilaajayritys itse. Tilaajayrityksen on tarkoitus

ottaa uusi sovellus käyttöön tulevana kesänä 2011. Yritys toimii vain kesäisin mutta valmistelut alkavat jo hyvissä ajoin ennen kesää. Tulevaa sovellusta pitää myös tulevien käyttäjien päästä kokeilemaan.

2 Määrittely

2.1 Johdanto

Määritellyllä sovelluksella on tarkoitus suunnitella yrityksen työntekijöiden työvuorot. Lisäksi sovelluksella on tarkoitus laskea työntekijöiden palkat, lähettää viestejä työntekijöille ja hallita työntekijöiden ja työpisteiden tietoja.

Sovelluksen tilaaja on ulkomaisten ja kotimaisten tuoremarjojen ja vihannesten torikauppias Pohjois-Suomessa. Yritys on aktiivinen kesäisin ja toimii usealla paikkakunnalla monissa eri työpisteissä ja tapahtumissa. Työntekijät ovat nuoria ja paikallisia kesätyöntekijöitä. Yritys on kasvanut, työpisteitä ja työntekijöitä on tullut lisää. Nykyisellä menetelmällä tietojen hallinta alkaa käydä hankalaksi ja kalliiksi.

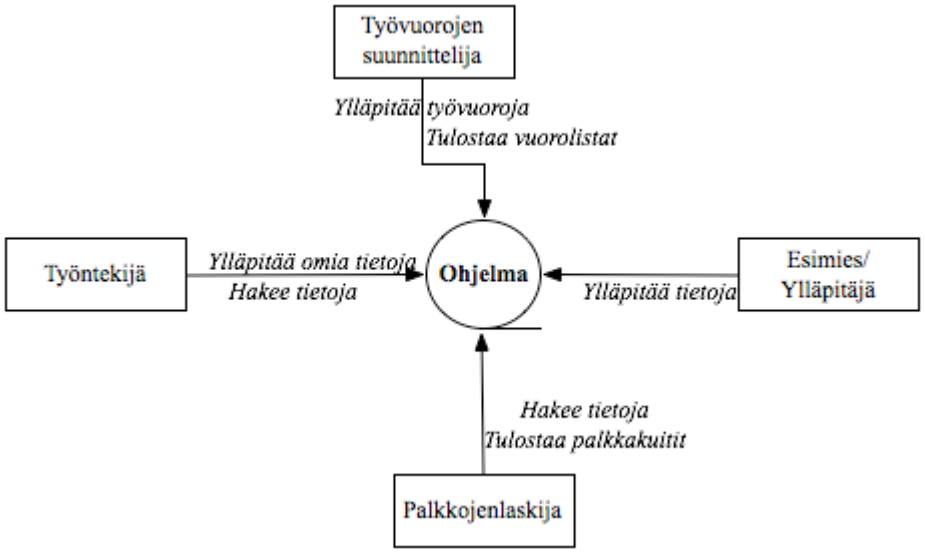
Sovelluksen tavoitteena on päästä eroon nykyisen menetelmän ylimääräisistä työvaiheista ja virheistä. Työvuorojen suunnittelusta koituu nykyisellä menetelmällä paljon töitä, koska yrityksellä on paljon työntekijöitä. Työntekijöistä suurin osa vaihtuu joka vuosi. Lisäksi kaikki työntekijät ovat töissä tuntipalkalla ja työajat saattavat muuttua nopeasti.

Sovelluksella on eri käyttäjiä ja oikeuksia. Käyttäjiä ovat työnantaja, työvuorojen suunnittelija, palkanlaskija ja mahdollisesti työntekijä itse. Kaikki käyttäjät voivat olla eri puolilla Suomea, kuitenkin kaikilla täytyisi olla mahdollisuus nähdä ajantasaiset tiedot samalla hetkellä.

2.2 Ulkoiset liittymät

Sovellus on tarkoitettu käytettäväksi selaimella. Käyttäjä voi olla työvuorojen suunnittelija, työntekijä, ylläpitäjä tai palkanlaskija. Käytännössä riittää kuitenkin kaksi käyttäjätasoa: ylläpitäjä ja peruskäyttäjä, joka voi olla myös työntekijä. Ylläpitäjän ja peruskäyttäjän

oikeudet ja näkymät poikkeavat toisistaan. Ylläpitäjä näkee kaikki toiminnot ja omistaa oikeudet muuttaa ja ylläpitää kaikkia tietoja. Peruskäyttäjällä on oikeudet muuttaa omia käyttäjä- ja yhteystietojaan ja nähdä sille lähetetyt viestit. Peruskäyttäjä voi myös viitata työntekijään, jolloin käyttäjä näkee omat työvuorot ja lähetetyt palkkakuitit. Ylläpitäjä voi halutessaan antaa työntekijälle myös oikeuden muuttaa toteutuneita työvuorojaan. Kuvassa 1 esitetään, kuinka eri käyttäjien on tarkoitus sovellusta käyttää.



Kuva 1. Liittymäkaavio

2.3 Tietokanta

Tietokannan käsittellinen määrittely perustui asiakkaan kanssa käytyihin haastatteluihin, aihealueeseen tutustumisesta kertyneisiin tietoihin ja muiden samaa aihealuetta koskevien sovelluksien tutustumisesta kertyneisiin tietoihin.

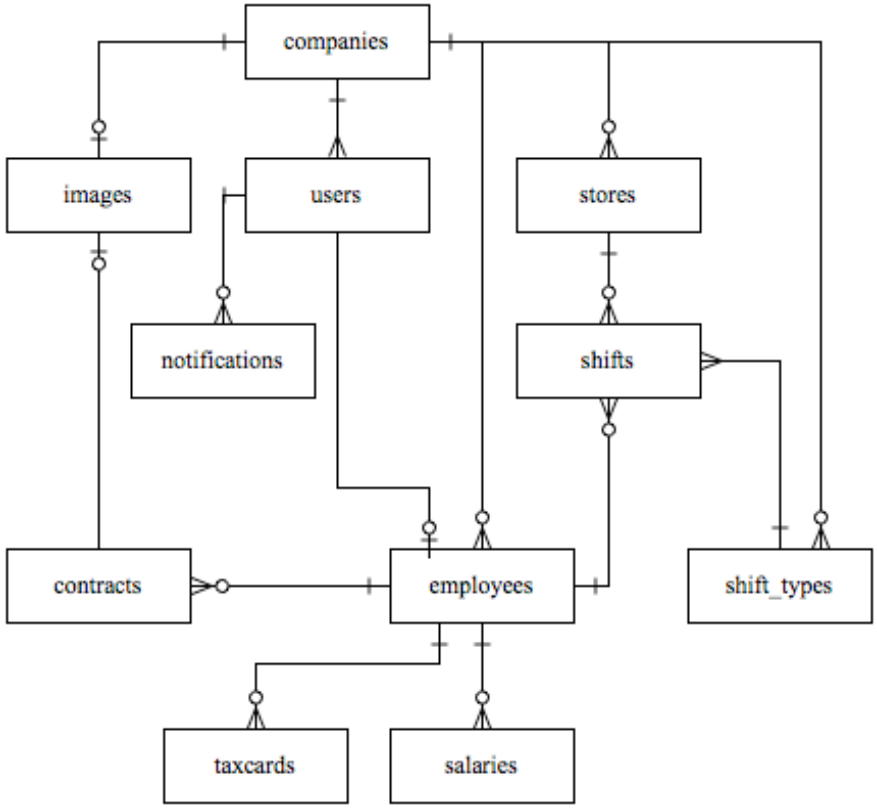
Yritys-käsite on tietokannan keskipiste. Se on muita tietoja yhdistävä tieto. Tämä mahdollistaa useiden yritysten palvelun samalla alustalla. Yrityksellä on useita työntekijöitä, työpisteitä ja käyttäjiä. Työntekijöillä on useita työvuoroja, työsopimuksia ja verokortteja. Työntekijä voi olla myös sovelluksen peruskäyttäjä, jolla on rajoitettuja oikeuksia.

Työvuoro voi olla suunniteltu tai toteutunut, mutta tiedot molemmista täytyy jäädä talteen. Työntekijällä voi olla useita työsopimuksia, koska sama työntekijä voi olla töissä eri

jaksoina eri palkalla. Työntekijällä on useita verokortteja, koska verokortti voi muuttua.

Käyttäjällä voi olla useita viestejä. Käyttäjä voi olla viestin lähettäjä tai vastaanottaja. Viestillä voi olla useita vastaanottajia. Työntekijällä on useita palkkakuitteja. Palkkakuitti pitää sisällään tiedot työntekijälle lasketusta palkasta palkkajakson aikana.

Näiden tietojen pohjalta saadaan rakennettua käsitteet ja niiden väliset relaatiot. Kuvassa 2 on esitetty luokkakaavio tietokannasta.



Kuva 2. Luokkakaavio

2.4 Toiminnot

Sovellus täytyy olla suojattu ulkopuolisilta tahoilta. Sovellukseen pääsee sisään vain kirjautumalla sisään käyttäjänimellä ja salasanalla. Jos käyttäjänimeä tai salasanaa ei muista, niin sovellukselta voi pyytää tunnukset sähköpostiin, joka on asetettu tili luonnin yhteydessä.

Sovelluksella lähetetään myös viestejä ja ilmoituksia. Ylläpitäjä voi lähettää viestejä valituille työntekijöille, käyttäjille tai kaikille työntekijöille. Viestin voi lähettää myös sähköpostiin. Viesti menee joka tapauksessa aina myös sovelluksen sisäiseen viestijonoon, joka näkyy vastaanottajan profiilissa. Viestitoimintoa on tarkoitus käyttää myös päiväkirjana. Päiväkirjaviestit näkyvät vain ylläpitäjälle. Ylläpitäjän on tarkoitus käyttää päiväkirjaviestejä hyväksi työvuorojen suunnittelussa.

Sovelluksen pääpaino on työvuorojen suunnittelussa. Työntekijöille asetetaan työaikoja ja sijoitetaan työpisteisiin. Myöhemmin työntekijöille merkitään toteutuneet työajat. Toteutuneiden työaikojen perusteella voidaan laskea palkat.

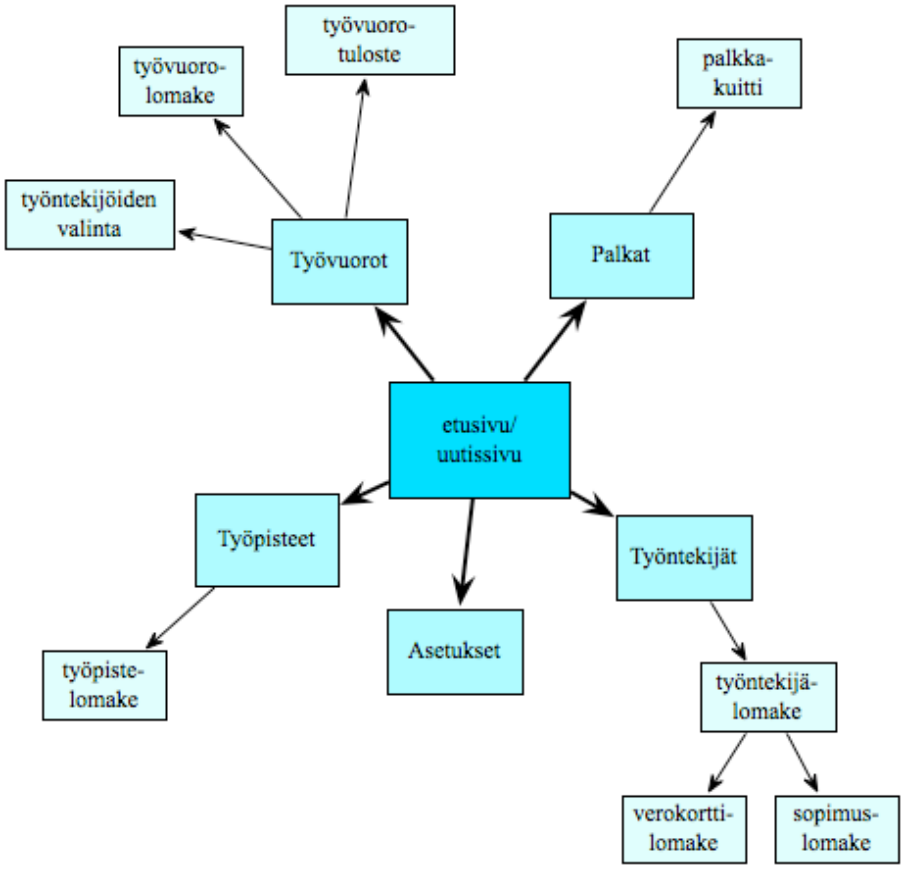
Ylläpitäjän täytyy voida lisätä, poistaa ja muokata työntekijöitä ja heidän tietoja. Tässä tapauksessa on myös tärkeää, että työntekijät ryhmitellään oikein. Suurin osa työntekijöistä vaihtuu joka kesä. Työntekijät voivat olla samannimisiäkin. On tärkeää, että työvuoroja suunniteltaessa ei näytetä vanhoja työntekijöitä ollenkaan.

Työpisteitä täytyy voida lisätä, poistaa ja muokata. Työpisteitä on paljon ja ne saatetaan poistaa käytöstä lyhyenkin käytön jälkeen. Työvuoroja suunniteltaessa on tärkeää, että käytöstä poistuneita työpisteitä ei näytetä turhaan.

Palkanlaskenta on yksi keskeisimmistä toiminnallisuuksista. Sovellus laskee toteutuneiden työvuorojen ja palkkatietojen perusteella työntekijöille palkkatiedot. Sovellus ottaa huomioon lisät, työntekijän verotiedot, sosiaaliturvamaksut ja työeläkemaksut. Laskettua palkkakuittia on tarkoitus käyttää ainakin toistaiseksi palkkatietoina ulkoistetulle palkanlaskijalle.

3 Suunnittelu

Oleellinen osa web-sovelluksen suunnittelua on sivukartta. Sivukartta esittää sovelluksen eri tilojen kytkennät toisiinsa, mistä tilasta pääsee mihinkin tilaan. Kuvassa 3 on sivukartta sovelluksesta. Ulommaiset tilat eivät varsinaisesti ole uusia sivuja vaan ikkunoissa esitettäviä Ajax-lomakkeita, joiden näyttäminen ei vaadi uutta sivulatausta.



Kuva 3. Sivukartta

3.1 Tietokanta

Tietokanta on koko sovelluksen sydän, se pyritään pitämään mahdollisimman yksinkertaisena ja tehokkaana. Tietokanta on tarkoitus pitää kolmannessa normaalimuodossa (3NF), jolloin kanta pysyy ehjänä ja järkevänä. Tietokannan taulut ja niiden relaatiot vastaavat todellisia käsitteitä ja malleja. Tarkoitus on käyttää Ruby on Rails ActiveRecord -mallia ja tehdä luokat, jotka vastaavat tietokannan tauluja ja niiden relaatioita. Näihin luokkiin kirjoitetaan myös käsitteelle ominaiset toiminnot. Liitteessä 2 on listattu tietokanta tarkemmin ActiveRecord::Schema -objektina, jonka avulla voidaan luoda esimerkiksi MySQL- tai SQLite -tietokanta.

Relaatiot saadaan määrittelystä mutta joitain käsitteitä voi ja kannattaa jakaa ylläpidettävyyden kannalta eri tauluihin. Esimerkiksi työvuoron tyyppi toistuu kaikissa

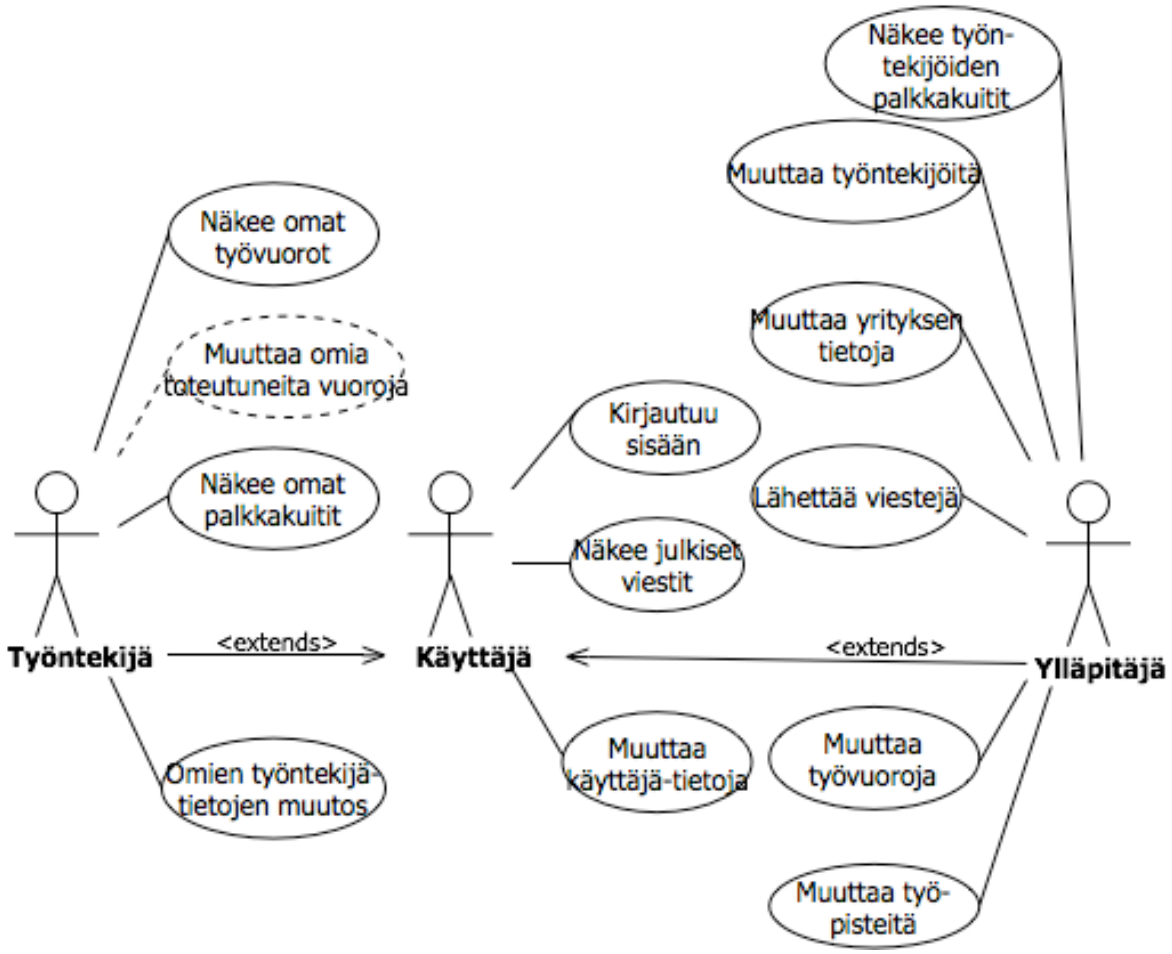
työvuoroissa. Työvuoron tyyppi voisi olla vain oma kenttä työvuorotaulussa, mutta jos se on oma taulunsa, johon viitataan, voidaan ylläpitää järkevästi erilaisia työvuorotyyppejä. Toisaalta liiallinen tietojen hajoittaminen eri tauluihin voi myös hidastaa tietokantahakuja.

Tietokannan relevanttina pysymiseksi tietojen pirstaloituminen täytyy estää. Relaatiotietokannassa tauluilla on viittauksia toisiin tauluihin. Jos jokin rivi poistetaan taulusta, siihen viittaavat muiden taulujen rivit ovat rikki. Tätä voidaan hallita ActiveRecord-mallien avulla. Käyttämällä malleja voidaan lisätä toiminnallisuutta, joka muuten olisi hankala toteuttaa suoraan SQL-kielellä. Esimerkiksi ennen työntekijän poistoa voidaan tarkistaa, onko työntekijällä toteutuneita työvuoroja. Työntekijä, jolla on työvuoroja voidaan poistaa vain näkyvistä. Näin säilytetään tiedot tilastoja varten ja työntekijä voidaan palauttaa myöhemmin takaisin. Käyttäjän halutessa poistetaan myös työntekijän työvuorot ja muut tiedot, joissa on viittauksia työntekijään.

Työvuoro sisältää tiedon sekä suunnitellusta työvuorosta että toteutuneesta työvuorosta. Näin ei tarvitse tehdä ylimääräisiä hakuja vaan kaikki tarvittavat tiedot saadaan molemmissa tapauksissa yhdestä objektista. Sovelluksessa esimerkiksi työvuoroja listatessa tullaan näyttämään suunnitellut ja toteutuneet vuorot yhdellä kertaa.

3.2 Käyttötapaukset

Käyttötapaukset kuvaavat toimintoja ja vuorovaikutuksia, joita järjestelmän ja sen käyttäjäroolien välillä on. Käyttäjäroolit on jaettu kahteen eri rooliin, jotka periytyvät samasta roolista eli peruskäyttäjästä. Se tarkoittaa sitä, että sekä työntekijällä että ylläpitäjällä on peruskäyttäjän tiedot ja toiminnot. Työntekijällä ja ylläpitäjällä on kuitenkin myös omia käyttötapauksia ja oikeuksia. Kuvassa 4 on esitetty käyttötapauskaavio, josta näkee, mitä käyttötapauksia eri rooleilla on.



Kuva 4. Käyttötapauskaavio

Kuvassa 5 on kerätty käyttötapaukset taulukkoon ja merkitty ne tunnisteella. Käyttötapaukset selitetään myöhemmin tarkemmin ja tapauskohtaisesti.

Käyttötapaus ID	Ensisijainen rooli	Käyttötapauksen nimi
1	Käyttäjä	Kirjautuu sisään
2	Käyttäjä	Näkee julkiset viestit
3	Käyttäjä	Muuttaa käyttäjä-tietoja
4	Ylläpitäjä	Muuttaa työntekijöitä
5	Ylläpitäjä	Muuttaa yrityksen tietoja
6	Ylläpitäjä	Lähettää viestejä
7	Ylläpitäjä	Muuttaa työvuoroja
8	Ylläpitäjä	Muuttaa työpisteitä
9	Työntekijä	Näkee omat työvuorot
10	Työntekijä	Muuttaa omia toteutuneita vuoroja
11	Työntekijä	Näkee omat palkkakuitit
12	Työntekijä	Omien työntekijä-tietojen muutos
13	Ylläpitäjä	Näkee työntekijöiden palkkakuitit

Kuva 5. Käyttötapaustaulu

Käyttötapaus 1: "Käyttäjä kirjautuu sisään."

Käyttäjä avaa selaimella sovelluksen ja kirjautumisikkuna tulee esiin. Käyttäjä kirjoittaa saamansa käyttäjänimen ja salasanan ja lähettää tiedot painamalla ikkunassa olevaa nappia. Selain ohjautuu sovelluksen etusivulle kirjautumisen onnistuessa. Jos sisäänkirjautuminen epäonnistuu, tullaan takaisin samalle sivulle ja ikkunassa näytetään virheilmoitus. Liitteessä 1 on malli kirjautumisikkunasta.

Käyttötapaus 2: "Käyttäjä näkee julkiset viestit."

Sovelluksen etusivulla käyttäjän ollessa sisäänkirjautuneena näytetään viestit. Näytettävät viestit ovat kaikille julkiset tai viestit, joissa kirjautunut käyttäjä on ollut vastaanottajana. Viestistä näkyy, kuka sen on lähettänyt ja milloin se on lähetetty sekä mikä on viestin sisältö. Viesteihin ei voi vastata.

Käyttötapaus 3: "Käyttäjä muuttaa käyttäjätietoja."

Käyttäjän ollessa sisäänkirjautuneena pääsee käyttäjä muuttamaan asetuksia "Asetukset" linkin avulla. Asetukset -sivulla voidaan muuttaa salasana ja yhteystiedot.

Käyttötapaus 4: "Ylläpitäjä muuttaa työntekijöitä."

Ylläpitäjä voi muuttaa työntekijöiden tietoja "Työntekijät"-sivun kautta. Työntekijät näkyvät listassa sukunimen perusteella järjestettynä. Listassa olevat työntekijät voivat olla aktiivisia, ei-aktiivisia ja hakijoita. Aktiivinen työntekijä on työntekijä, jolla on kyseisellä ajanhetkellä voimassa oleva tai myöhemmin alkava työsopimus.

Ei-aktiivisella työntekijällä on työsopimus, joka on mennyt jo vanhaksi. Hakija on työntekijä, jolla ei ole vielä ollenkaan työsopimusta. Työntekijöitä voi hakea, lisätä, poistaa ja muokata. Työntekijän kaikki yhteystiedot ja käyttäjätiedot ovat näkyvillä ja muutettavissa. Ylläpitäjä voi myös myöntää tai evätä työntekijältä oikeuden kirjautua sovellukseen ja oikeuden muuttaa toteutuneita työvuorojaan.

Uuden työntekijän lisäys tapahtuu "Hakijat"-sivun kautta, jossa uusi hakija lisätään napsauttamalla taulukon viimeistä ja tyhjää riviä. Kun uusi hakija on luotu, hänelle voidaan tehdä työsopimus, mikä tekee hakijasta työntekijän. Liitteessä 1 on malli työntekijä- ja työsopimuslomakkeesta.

Käyttötapaus 5: "Ylläpitäjä muuttaa yrityksen tietoja."

"Asetukset"-linkin kautta ylläpitäjä pääsee yleisten asetusten sivulle, josta yrityksen perustiedot ovat muutettavissa. Ylläpitäjä voi muuttaa omia käyttäjätietoja, yrityksen yhteystietoja ja työntekijöiden käyttäjätietoja ja oikeuksia.

Käyttötapaus 6: "Ylläpitäjä lähettää viestejä."

Ylläpitäjä voi lähettää viestejä muille käyttäjille tai työntekijöille. Ylläpitäjä voi osoittaa viestin joko yksittäisille työntekijöille, kaikille aktiivisille työntekijöille, ei-aktiivisille työntekijöille tai hakijoille. "Vastaanottajat"-kentässä on aktiivinen haku, joka esittää hakutulokset listassa. Ylläpitäjä lisää vastaanottajan valitsemalla listasta jonkun hakutuloksista. Viesti lähetetään napsauttamalla "Lähetä viesti"-nappulaa. Viesti lähetetään aina vastaanottajan käyttäjäprofiiliin. Jos vastaanottajalla ei ole käyttäjätiliä, viesti lähetetään sähköpostiin. Ylläpitäjä näkee aina myös lähetetyt viestit. Liitteessä 1 on

malli viestisivusta.

Käyttötapaus 7: "Ylläpitäjä muuttaa työvuoroja."

Työvuorojen muuttamisen työnkulku on seuraavanlainen: "Työvuorot"-sivulla valitaan ensin vuorojen alkamispäivämäärä ja päättymispäivämäärä. Sitten painetaan "Valitse tekijät"-nappia ja valitaan esille tulevasta valikosta halutut aktiiviset työntekijät ja klikataan "Ok"-nappia. Valittujen työntekijöiden työvuorot valitulta ajalta tulevat näkyviin. Työntekijöillä on kaksi saraketta: "suunniteltu" ja "toteutunut".

Päivät kulkevat riveittäin ylhäältä alaspäin, jokaiselle päivälle on jokaiselle työntekijälle suunniteltu vuoro ja mahdollisesti toteutunut vuoro. Kaksoisnapsauttamalla työvuorosolua tulee esille työvuorolomake, jolla muutetaan tai luodaan valitun päivän työvuoroja. Jos työvuoro on jo olemassa, lomakkeessa on nappi, jolla voidaan poistaa vuoro. Muuttamalla lomakkeen tietoja ja napsauttamalla "Tallenna"-nappulaa työvuoron tiedot päivitetään tietokantaan ja näkymä päivitetään.

Käyttötapaus 8: "Ylläpitäjä muuttaa työpisteitä."

Työpisteitä voi muutta "Työpisteet"-sivulla. Siellä listataan kaikki työpisteet. Työpisteen tietoja pääsee muuttamaan napsauttamalla työpisteen nimeä. Lomake tulee esiin, sillä voi muuttaa pisteen nimeä, kuvausta ja tilaa. Työpisteen tila voi olla aktiivinen tai ei-aktiivinen. Jos tila on ei-aktiivinen, se ei esiinny työpisteen valintalistassa työvuorolomakkeessa. Jos työpiste on jo olemassa, lomakkeessa on nappi, jolla voidaan poistaa se. Uusia työpisteitä voidaan myös lisätä "Työpisteet"-sivulla.

Käyttötapaus 9: "Työntekijä näkee omat työvuorot."

Työnantajan halutessa työntekijä näkee omat työvuoronsa. Työnantaja voi myös lähettää sähköpostilla työntekijälle linkin työvuorolistan vedokseen.

Käyttötapaus 10: "Työntekijä muuttaa omia toteutuneita työvuoroja."

Työnantajan niin halutessa työntekijä näkee omat työvuoronsa ja pääsee muokkaamaan

toteutuneita työvuoroja. Työntekijän kirjautuessa sisään "Työvuorot"-sivulla näkyy vain työntekijän omat vuorot. Työntekijä voi muuttaa toteutunutta työvuoroaan mutta ei voi poistaa vuoroa eikä muuttaa suunniteltua vuoroa.

Käyttötapaus 11: "Työntekijä näkee omat palkkakuitit."

Työntekijän kirjautuessa sisään "Palkat"-sivulla näkyy lista työntekijän palkkakuittien vedoksista. Palkkakuitin vedos tulee näkyviin vain silloin, kun palkkojenlaskija on hyväksynyt ja vedostanut palkkajakson.

Käyttötapaus 12: "Työntekijä voi muuttaa omia työntekijätietojaan."

Työntekijä voi muuttaa omia tietojaan "Asetukset"-sivulla. Työntekijä voi muuttaa peruskäyttäjätietoja mutta myös työntekijätietoja, kuten yhteystietoja, henkilötietoja tai profiilikuvan.

Käyttötapaus 13: "Ylläpitäjä näkee työntekijöiden palkkakuitit."

Ylläpitäjä valitsee halutun vuoden, palkkajakson ja työntekijät, joille palkkakuitti lasketaan. Sovellus laskee palkkakuitit valituille työntekijöille ja listaa ne taulukkoon. Kun ylläpitäjä hyväksyy palkkakuitit, ne tallennetaan tietokantaan.

3.3 Käyttöliittymä

Käyttöliittymä toteutetaan selaimeen HTML-, CSS- ja Javascript-tekniikoita käyttäen. Javascriptillä, HTML:llä ja CSS:llä saadaan hyvinkin näyttäviä käyttöliittymiä. Ongelmana selainpohjaisissa käyttöliittymissä on toisistaan poikkeavat loppukäyttäjien käyttämät selaimet ja päätelaitteet. Kaikkia selaimia on vaikea tukea. Päätettiin, että sovellus tukisi virallisesti Mozilla Firefox-, Safari- ja Chrome-selaimia. Mainitut selaimet ovat yrityksen käytössä.

Sovelluksen perusnäkymät ovat valittavissa välilehdistä. Tiedot esitetään taulukoissa. Tietojen muokkaus toteutetaan Ajax-lomakkeina ja ikkunoissa. Ikkunat avautuvat samassa

selainikkunassa eikä uusia selainikkunoita tarvitse aukoa.

4 Menetelmät

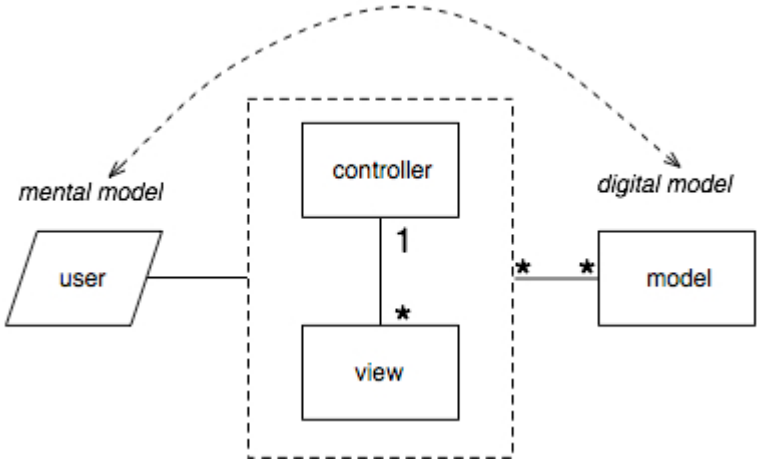
4.1 Ruby

Ruby on Yukihiro “matz” Matsumoton kehittämä dynaaminen ja tulkattava olio-ohjelmointikieli. Tekijä itse kuvaa luomustaan sanoin: "Ruby on ulkoapäin yksinkertainen mutta erittäin monimutkainen sisältä, aivan niin kuin ihmiskeho". Tekijä on ottanut vaikutteita lempikielistään Perl, Smalltalk, Eiffel, Ada ja Lisp. Rubyssa kaikki muuttujat ovat olioita, jopa primitiiviset tyypit. Rubysta tekee hyvän web -ohjelmointikielen sen dynaamisuus, syntaksi ja alustariippumattomuus. [1.]

Ruby on Rails, RoR tai Rails on David Heinemeier Hanssonin kehittämä avoimeen lähdekoodiin perustuva web-sovelluskehys. Se on kehitetty käyttäen Ruby-ohjelmointikieltä. RoR:n versio 1.0 on julkaistu vuonna 2005 ja vuoteen 2010 mennessä viimeisin juuri tullut versio on 3.0, jonka tekemisessä on ollut mukana yli 1600 ohjelmoijaa. [2.]

MVC -malli

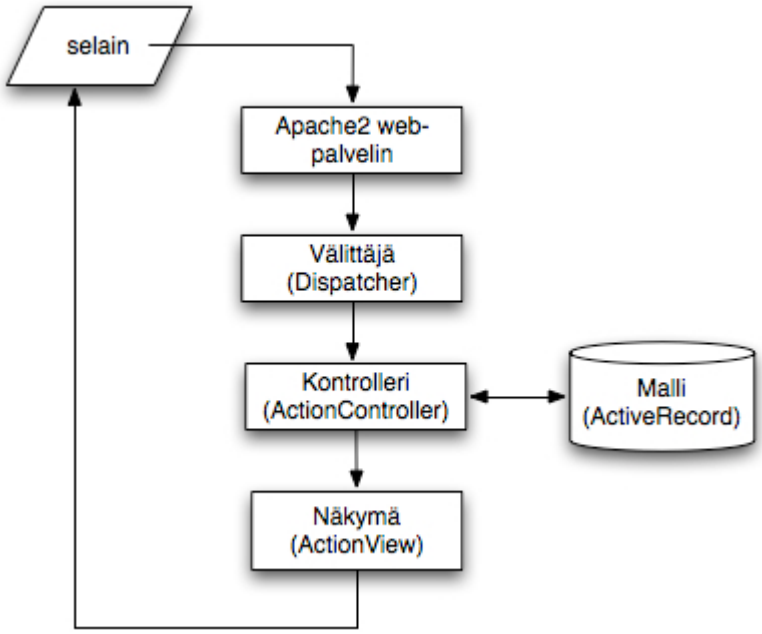
Ruby on Rails on skaalautuva web-sovelluskehys, joka perustuu MVC (Model-View-Controller) -ohjelmointiparadigmaan. Mallin alkuperäinen tarkoitus oli luoda silta ihmisen ajatteleman mallin ja tietokoneen sisältämän mallin tai datarakenteen välille niin, että ideaalitapauksessa käyttäjä näkee ja hallitsee tuota mallia. Tämä osoittautuu käytännölliseksi ratkaisuksi, kun käyttäjän täytyy nähdä sama malli useissa eri asiayhteyksissä ja näkymissä. Kuvassa 6 havainnollistetaan asiaa. [2;4.]



Kuva 6. MVC -malli

Ruby on Railsissa MVC-malli näkyy hyvin vahvasti. MVC-mallin sydän on kontrolleri (controller). Se on vastuussa sovelluksen etenemisestä. Kun käyttäjä tekee pyynnön sovellukselle, kontrolleri on se, joka ottaa pyynnön vastaan ja valmistelee puitteet näkymän (view) luomiseksi. Kontrolleri hakee tiedot käyttämällä malleja. Mallit ovat vastuussa tietojen hakemisesta ja viemisestä. Tietolähde voi olla esimerkiksi relaatiotietokanta. Malleihin tulisi sisällyttää myös kaikki bisneslogiikka. Näkymä ja kontrolleri ovat vahvasti sidottu toisiinsa. Ne jakavat paljon tietoja pääasiassa muuttujien avulla. Kontrollerin tehtävä on antaa näkymälle tarvittavat tiedot ja esittää näkymä palvelimelle, joka lähettää näkymän käyttäjälle. [10.]

Kuvassa 7 on yksinkertaistettu kuvaus, jossa näkyy, kuinka Ruby on Rails toimii tässä sovelluksessa.



Kuva 7. Ruby on Rails sovellukssa

RubyGems

RubyGems on Ruby -ohjelmistopakettien jakamispalvelu. Sillä voidaan asentaa lisäosia ja kirjastoja Rubylle. RubyGems paketteja on jo lähes 18 000. RubyGems on perustettu vuonna 2009 Nick Quaranton toimesta.

Object-relation mapping (ORM)

ORM on ohjelmointimalli, joka toimii siltana erilaisten tyyppijärjestelmien välillä olio-ohjelmoinnissa. Esimerkiksi Java-ohjelmointikielen ja MySQL-tietokannan välillä.

4.2 Active Record

Active Recordin on nimennyt Martin Fowler kirjassaan "Patterns of enterprise application architecture". Active Record on tapa päästä käsiksi tietoihin tietokannassa. Se on toteutus Object-relation mapping (ORM) -mallista. Tietokannan taulut vastaavat luokkia mutta luokan instanssi vastaa kuitenkin vain yhtä riviä taulussa. Luokka sisältää usein myös siihen liittyviä toimintoja ja logiikkaa. Active Record -malli on helppo ymmärtää ja se toimii

silloin hyvin, kun se vastaa mahdollisimman hyvin alla olevaa tietokantaa. [5.]

Rubylle tehdyssä Active Record -versiossa on viety alkuperäistä ajatusta hieman pidemmälle. Siinä ratkaistaan kaksi puuttuvaa osaa: relaatiot ja periytyminen. Muutamilla makroilla voidaan esittää relaatiot muihin tauluihin. Olioon haetaan relaatioiden varsinaiset tiedot oletuksena vasta kun niitä pyydetään. Seuraavassa on esimerkki, kuinka luodaan Active Record -malli yksinkertaisimmillaan. [6.]

```
class Product < ActiveRecord::Base; end
```

Edellä oleva lause määrittelee Active Record -mallin, joka vastaa seuraavaa tietokannan taulua:

```
CREATE TABLE products (
   id int(11) NOT NULL auto_increment,
   name varchar(255),
   PRIMARY KEY  (id)
 );
```

Ruby on Railsissa käytetään paljon konventioita, mikä tarkoittaa tiettyjä oletuksia ja nimeämis- ja menettelytapoja. Active Record olettaa, että luomalla "Product"-nimisen luokan halutaan sen vastaavan taulua nimeltä "products" ja että sillä on pääavaimena id niminen kenttä. Ruby on Railsissa on paljon tällaisia konventioita mutta se antaa myös mahdollisuuden asettaa oletukset toisin. [6.]

REST

REST on lyhenne englannin kielen sanoista Representational State Transfer. Alkujaan Ruby on Rails käytti kevyttä versiota SOAP:sta web-palveluissa mutta se muutettiin myöhemmin käyttämään RESTiä. REST on tapa tehdä web-palveluja käyttämällä hyväksi URI-mallia.

WEBrick

WEBrick on Ruby on Railsin mukana tuleva http-palvelinkirjasto. Sitä ei tarvitse erikseen asentaa vaan jokainen Ruby on Rails -sovellus sisältää WEBrick-palvelimen ja

komentosarjan, jolla se käynnistetään palvelemaan juuri sitä sovellusta. WEBrick-palvelinta käytetään yleensä kehitys- ja jopa testausvaiheessa.

SQLite

Sovelluksen kehitysvaiheessa käytetään SQLite-tietokantaa. Se on nopea, helppo asentaa ja toimii Ruby on Rails -sovelluskehyksen kanssa. SQLite on täysiverinen SQL-relaatio-tietokanta. Tietokantaan kirjoitetaan ja siitä luetaan täysin tavallisista tiedostoista. Palvelinprosesseja ei tarvita ollenkaan. SQLite-kirjasto vie vähimmillään vain 4 kilotavua, joten siitä on tullut suosittu myös pienissä mobiiliympäristöissä. SQLitea käytetään paljon myös testauksessa ja ad hoc -projekteissa, kun tarvitaan nopeasti siirrettäviä tietokantoja.

Rubyn Active Record mahdollisti helposti SQLite-tietokannan käytön sovelluksen kehitysvaiheessa ja MySQL-tietokannan käytön testi- ja tuotantovaiheessa.

MySQL

MySQL on suosittu SQL-relaatiotietokanta. Sen saa kohtalaisen vaivattomasti toimintakuntoon. MySQL-työkalut ovat kattavat. MySQL on hyvin tuettu ja nopea tietokanta, joten sitä käytetään paljon kehitystyössä mutta myös tuotantoympäristöissä. Sen varmuuskopiointi ja palautus on helppoa, joten käytin MySQL-tietokantaa työn lopullisena tietokantana tuotanto- ja testausvaiheessa.

Rubylle on kirjoitettu Ruby-pohjainen "ruby/mysql"-rajapinta ja nopeampi C-pohjainen "mysql/ruby"-rajapinta. ActiveRecordin mukana tulee Ruby-pohjainen "ruby/myqsl"-rajapinta mutta ActiveRecord toimii myös C -pohjaisella rajapinnalla.

4.3 Javascript-kirjastot

Prototype

Prototype-javascript-kirjasto sisältää toimintoja, jotka helpottavat html-elementtien ja Ajaxin hallintaa. Prototype tulee valmiina Ruby on Railsin mukana ja on vahvasti integroituna Ruby on Railsin RJS -malleissa. [9.]

Scriptaculous

Scriptaculous -javascript-kirjasto tulee Ruby on Rails -asennuksen mukana. Scriptaculous-kirjastolla saa aikaan näyttäviä tehosteita html-elementeille. Tämäkin kirjasto on kytketty vahvasti Ruby on Railsin RJS -malleihin. Scriptaculous-kirjastoa voi siis halutessaan käyttää kirjoittamalla Ruby-koodia Javascriptin sijaan. [8.]

Prototype UI Window

Prototype-kirjastoon nojaava Prototype UI-kirjasto on muutaman Javascript- ja käyttöliittymäammattilaisen tekemä kirjasto, jonka Window-moduulia käytetään tässä sovelluksessa. Sillä on mahdollista tehdä modernien käyttöjärjestelmän ikkunoiden näköisiä ikkunoita, joissa on sisäänrakennettuna ikkunoille tavalliset perusominaisuudet.

Tableorderer

Greg Schurgast on tehnyt loistavan Tableorderer-moduulin datataulukoiden näyttämiseen. Se käyttää hyväkseen Prototype-kirjastoa, mikä oli suurin syy, miksi päätin käyttää sitä. Koodi on täysin avoin. Jouduinkin hieman muokkaamaan sitä omiin tarpeisiini. Taulukon toimintoja on sisällön hakeminen Ajaxilla, sisällön järjesteleminen minkä tahansa sarakkeen mukaan, valmis käyttöliittymä ja sivutus. [7.]

5 Toteutus

5.1 Työvuorojen suunnittelu

Työvuorojen suunnittelu osoittautui vaikeaksi toteuttaa. Henkilö, joka suunnittelee työvuorot, oli sunnitellut vuorot aikaisemmin kynällä ja paperilla ja tekstinkäsittely-ohjelmalla. Käyttöliittymässä täytyi olla näkyvissä mahdollisimman paljon tietoa mutta samalla mahdollisimman vähän. Luonnollinen työnkulku tuntui olevan sellainen, että ensin valitaan tarkasteltava aikaväli, sitten tarkasteltavat työntekijät. Kaikki valitut työntekijät pitäisi nähdä kerralla koko tarkastasteltavalta aikaväliltä.

Ikkunointi selaimen sisällä kuten työvuorolomakkeen näyttäminen ikkunassa toimii Ajaxilla

Prototype UI Window -kirjaston avulla. Sen avulla päästään eroon turhista sivulatauksista ja vain olennaiset tiedot haetaan palvelimelta ja näytetään lomakkeella ikkunassa, jonka saa nopeasti piiloon tai kokoa muutettua. Yksi suurimmista hyödyistä ikkunoita käyttämällä tulee siitä, että ikkunoita voi avata niin paljon kuin haluaa. Käyttäjä voi halutessaan vertailla usean työntekijän työvuoroja kerralla avaamalla usean ikkunan auki. Myös tietojen muokkaaminen ja päivittäminen toimii samalla tavalla.

5.2 Palkkojen laskeminen

Työsopimus

Työsopimus sisältää tiedon työn alkamispäivästä ja päättymispäivästä, tuntipalkasta ja kiinteästä palkasta, työsuhde-eduista ja niiden euromääräisestä arvosta. Työsopimus voi olla toistaiseksi voimassa oleva tai määräaikainen.

Verokorttitiedot

Työntekijällä on verokortti, joka voi muuttua. Verokorttitietoja voidaan hallita työntekijälomakkeella. Verokortista tallennetaan tiedot voimaantulopäivästä, perusprosentista, lisäprosentista ja vuosittaisesta tulorajasta.

Palkkakuitit

Palkkojen laskeminen perustuu tarkasteltavan palkkajakson aikana työntekijän voimassa olevaan työsopimukseen. Palkkakuitit tarkasteltavalta jaksolta lasketaan vain, jos palkkakuittia ei jo löydy tietokannasta. Palkkakuitti tallennetaan tietokantaan kun se hyväksytään eli laitetaan maksuun. Palkkakuitti on käsite, jolla on oma taulunsa tietokannassa ja myös siis oma ActiveRecord -malli, jossa on sen sisältämä toiminnallisuus. Palkkojenlaskuun liittyvät toiminnot löytyvät siis mallista.

Palkkajaksot perustuvat ylläpitäjän asetuksissa määrittelemiin palkanmaksupäiviin. Palkanmaksupäivä voi olla kuukauden ensimmäinen, viimeinen tai mikä tahansa muu kuukauden päivä. Palkanmaksupäiviä voi olla myös useita kuukaudessa. Palkkajaksojen

valintalista palkat-sivulla perustuu palkanmaksupäiviin.

Palkkojen laskeminen tapahtuu ActiveRecord-mallissa. Funktiolle, joka laskee palkkakuitin annetaan parametreina työntekijä ja palkkajakson alkamis- ja päättymispäivä. Ajankohtaiset verokorttitiedot ja palkkatiedot haetaan palkkajaksolle ja lasketaan palkka käymällä jakson työpäivät läpi yksi kerrallaan. Päivän laskettuun palkkaan vaikuttaa viikonpäivä, kellonaika ja tuntipalkka. Kun jakson bruttopalkka on laskettu, siitä vähennetään ennakonpidätys, sosiaaliturvamaksu ja työeläkemaksut.

Palkkakuitti, joka syntyy laskennan pohjalta, sisältää ennakonpidätykset, sosiaaliturvamaksun ja työeläkemaksut, joita tosin ei ole alle 18-vuotiailla. Kuittia on tarkoitus käyttää palkkatietona, joka lähetetään ulkoistetulle palkanlaskijalle.

5.3 Viestien lähetys

Uuden viestin lähetykseen käytetään lomaketta. Vastaanottajat valitaan kirjoittamalla tekstikenttään vastaanottajan nimi. Tekstikentässä on aktiivinen haku, joka hakee työntekijöitä kirjoittaessa. Tekstikentän alle ilmestyy lista hakutuloksista. Hakutulos koostuu käyttäjistä, joita voi valita tekstikenttään. Hakutulosten listassa on myös valinta "aktiiviset työntekijät", jonka avulla voi lähettää viestin kaikille aktiivisille työntekijöille.

Viestin voi lähettää sovelluksen sisäisesti ja myös sähköpostina. Sähköpostin lähetys Ruby on Railsissa tehdään käyttämällä hyväksi ActionMailer-kirjastoa. Sähköpostia varten tehdään malli: luokka, joka periytyy ActionMailer::Base-luokasta. Malliin kirjoitetaan funktio, joka valmistelee tarvittavat tiedot, kuten vastaanottajat, lähettäjän, viestin sisällön ja asian. Funktio välittää tiedot sapluunalle. Sapluunoja voi olla useita. Esimerkiksi HTML- ja tekstisähköposteille on omat sapluunansa.

6 Testaus

Testaaminen Ruby on Railsilla on varsin toimivaa ja hyvin dokumentoitua. Yksikkö-testaukseen ja toiminnalliseen testaukseen on olemassa valmiit kirjastot, jotka ovat

helppokäyttöisiä. Yksikkötestausten ja toiminnalisen testauksen lisäksi Ruby on Railsille on valmiina RSpec- ja Cucumber-kirjastot, joilla voidaan testata sovelluksen käyttäytymistä. Cucumber ja RSpec ovat Behaviour Driven Development (BDD) -tapa kehittää sovelluksia. Se tarkoittaa sitä, että kirjoitetaan miltei selkokielellä, miten sovelluksen tulisi käyttäytyä. Sitten ajetaan testi ja huomataan, että sovellus ei käyttäydy niin kuin pitäisi. Sitten ohjelmoidaan sovellus käyttäytymään niin kuin halutaan. Tämän jälkeen testien on tarkoitus mennä läpi ja voidaan siirtyä seuraavaan ominaisuuteen.

BDD-menetelmän etu on siinä, että kuka tahansa pystyy lukemaan ja ehkä jopa kirjoittamaankin testejä. Menetelmä pienentää asiakkaiden ja kehittäjien välistä kuilua. Samalla tulee kirjoitettua dokumenttia siitä mitä sovellus tekee. Cucumber-kirjastoon on olemassa myös suomenkielinen versio, jolloin testejä voi kirjoittaa myös omalla kielellä.

7 Asennus

Käyttöönotto

Sovelluksen käyttöönotto esiasennetulle palvelimelle sisältää seuraavat toimenpiteet: edellisen version arkistointi, uuden version lataaminen versiohallinnasta, tarvittavien tietokantamuutosten ajaminen tietokantaan ja web-palvelimen uudelleenkäynnistys. Tällaisissa käyttöönottoon littyvissä tehtävissä auttaa Ruby-kirjasto nimeltä Capistrano. Sillä voi ohjelmoida edellä mainitut tehtävät ja ajaa ne miltä koneelta hyvänsä kirjautumatta palvelimelle itse ollenkaan.

Sovelluksen versiohallinta on Githubissa. Github on web-palvelu, joka hyödyntää Git-versionhallintajärjestelmää. Sovelluksen vieminen palvelimelle tapahtuu Capistranon avulla. Sen avulla versionhallinnasta haetaan viimeisin versio, joka asennetaan palvelimelle ja korvataan vanha versio.

Palvelin

Käytettävä tuotantopalvelin on Rackspace-yrityksen pilvipalvelin. Pilvipalvelin eroaa virtuaalipalvelimesta siinä, että sen täysi hallinta kuten käynnistys, sammutus ja

laitekokoonpanon muutokset, tapahtuvat työkaluilla webin kautta. Koko palvelimesta voi helposti ottaa myös ajastettuja varmuuskopioita, jotka voi palauttaa hetkessä takaisin. Käyttöjärjestelmänä toimii Linuxin Debian Lenny-jakeluversio.

Ylläpito

Tietokannasta otetaan varmuuskopioita kerran päivässä, ne saadaan helposti palautettua tarvittaessa. Palvelimen kaatuessa ja käynnistyessä uudestaan ei tarvita erillisiä toimenpiteitä ylläpitäjältä vaan web-palvelin ja tietokantapalvelin käynnistyvät automaattisesti ja sovellus on takaisin toimintakunnossa, kun palvelin on käynnistynyt.

8 Dokumentointi

Koodin dokumentointi on tärkeä osa sovellusta. Ilman dokumentteja on myöhemmin vaikea ymmärtää koodia. Dokumentoinnin helpottamiseksi on olemassa työkaluja, joiden avulla voidaan generoida dokumentteja suoraan sovelluksen koodista. RDoc on tällainen työkalu, se tulee Ruby on Railsin mukana. RDoc generoi HTML-dokumentin koodista. RDoc lukee automaattisesti luokat, moduulit, metodit, attribuutit ja tietyt kommentit. Tällaisella työkalulla ja kommentoimalla koodia saadaan ajantasaiset ja siistit dokumentit koodista.

9 Tietoturva

Tietoturva sovelluksessa on olennaista, koska tietokannassa säilytetään työntekijöistä henkilökohtaisia tietoja, kuten sosiaaliturvatunnus, tilinumero ja palkkatiedot. Tietoturvaa voi lähestyä monesta eri näkökulmasta eikä aukotonta tietoturvaa pystytä koskaan takaamaan niin kauan kun on ihmisiä, jotka järjestelmiä käyttävät. Usein järjestelmien suurin tietoturvauhka onkin juuri järjestelmän käyttäjät.

Tässä sovelluksessa varsinaisia tietoturvauhkia voivat olla käyttäjien lisäksi Internetissä leviävät haittaohjelmat. Internetin uhka voitaisiin poistaa asentamalla sovellus palvelimelle, joka ei ole yhteydessä Internettiin. Silloin sovellus ei olisi kaikkialla saatavilla eivätkä esimerkiksi työntekijät voisi käyttää sovellusta.

Haittaohjelmat tai krakkerit käyttävät hyväksi järjestelmien ohjelmointivirheitä. Tärkeintä on, että virheet pystytään tarpeen vaatiessa korjaamaan. Virheetöntä sovellusta ei ole. Tässä sovelluksessa on otettu huomioon tietoturva asentamalla palvelin niin, että vain tarvittavat ohjelmistot ovat toiminnassa eivätkä ne ole yhteydessä muualle. Web-palvelimen julkinen kansio ei sisällä mitään salaista. Miltei kaikki käytettävät ohjelmat ja kirjastot ovat avoimia, joten myös niiden virheitä päästään tarpeen vaatiessa korjaamaan. Hankkimalla SSL-sertifikaatin voidaan salata käyttäjän ja palvelimen välinen yhteys ja estää väliintulohyökkäykset (man-in-the-middle). Kaikilta hakukoneilta on myös evätty sivujen indeksointi tämän sovelluksen kohdalta.

Ruby on Rails perustuu avoimeen lähdekoodiin. Sillä on jopa oma kehitystiimi tietoturva-asioihin littyvien ohjelmointivirheiden korjaamiseen. Ruby on Rails on web-sovelluskehys ja sillä tehdyillä sovelluksilla on edessä samat tietoturvaongelmat kuin millä tahansa muullakin web-sovelluksella. Vaikka joissain tietoturvaongelmissa Ruby on Rails -sovelluskehys auttaa kehittäjää niin kehittäjän täytyy itse olla myös tietoinen yleisimmistä ongelmista ja krakkereiden käyttämistä tekniikoista. [1.]

Yleisimpiä krakkereiden käyttämistä tekniikoista on palvelunesto (denial-of-service), sivupyyntöväärennös (cross-site request forgery) ja injektiohyökkäykset, kuten SQL-injektiot. palvelunestohyökkäyksillä krakkeri yrittää jumittaa palvelun lähettämällä todella paljon pyyntöjä lyhyessä ajassa. Jos palvelin ei ole varautunut tällaiseen pommitukseen, se jumittuu eikä kukaan voi enää käyttää palvelua ennen kuin se käynnistetään uudelleen.

sivupyyntöväärennöshyökkäykset pyrkivät käyttämään hyväksi auki jääneitä istuntoja. Ruby on Railsissa on sisäänrakennettu suojaus sivupyyntöväärennöshyökkäyksiä vastaan. Kaikkien pyyntöjen mukana lähetetään todennusavain, jonka avulla palvelinpuolella voidaan todeta, että kyseessä todella on oikea käyttäjä. Todennusavain on satunnainen merkkijono, joka tallennetaan istunnon ajaksi ja luodaan piilokentäksi kaikkiin lomakkeisiin.

SQL-injektiohyökkäyksillä krakkeri pyrkii vahingoittamaan tietokantaa lähettämällä palvelimelle krakkerin oman vahingollisen SQL-kyselyn. Kaikki lomakkeiden tekstikentät, joiden avulla muodostetaan SQL-kyselyjä ovat potentiaalisia uhkia. Käyttämällä Rubyn

ActiveRecord-mallien hakumetodeja kyselyt pakotetaan turvalliseen muotoon eikä vahingollinen kysely pääse ajoon koskaan. [11.]

10 Jatkokehitys

Sovelluksen kehitys ei pääty vielä. Sovellusta tehdessä asiakkaan vaatimuksia tuli lisää ja paljon lisätoimintoja on toivottu. Viestien osalta on toivottu, että julkiset viestit voitaisiin halutessa julkaista myös blogina yrityksen kotisivuilla. Yksityiset viestit voitaisiin halutessa lähettää myös kännykkään tekstiviestinä (SMS). Tilien kirjanpitoa helpottamaan on toivottu moduuli, jolla voidaan hallita tilejä ja tilitapahtumia. Varmuskopiointi eli tietojen vieminen ja tuominen jossain säännönmukaisessa formaatissa, kuten XML tai CSV, oli myös toivottu ominaisuus. Työvuorojen lähettäminen työntekijöille täytyy olla mahdollista myös sähköpostilla paperin säästämiseksi.

Työvuorojen ja palkkojen erilaiset kertymäraportit olivat myös toivottu ominaisuus. Raportointi voi olla esimerkiksi valittujen kenttien esittäminen jossain rakenteellisessa muodossa. Kaavioita voi yritys piirtää itsekin esimerkiksi jollain taulukkolaskentaohjelmalla.

Vaikka sovellus on kehitetty tilaajayritystä varten, sen ratkaisemat ongelmat ovat yleisiä. Sovelluksen voisi jatkokehittää web-palveluksi muille vastaaville pienyrityksille.

11 Yhteenveto

Sovellus täyttää ne vaatimukset, jotka sille alkumäärittelyn mukaan asetettiin. Työn tilaaja ei kuitenkaan ollut aivan tarkkaan selvillä, mitä kaikkea sovelluksella pitäisi pystyä tekemään, joten vaatimukset kasvoivat projektin edetessä.

Sovelluksen testaaminen jäi vähäiseksi. Testejä on tarkoitus lisätä myöhemmin, jotta jatkokehitys ei rikkoisi olemassa olevia perustoimintoja. Käyttöliittymän testaus ja parannukset ovat myös osa testausta. ja parannuehdotuksia asiakkaan taholta tulee todennäköisesti lisää.
Työllä saavutettiin merkittävä helpotus aikaisempaan työnkulkuun verrattuna. Työvuoroja

suunnittelemaan riittää nyt yksi henkilö. Työntekijöiden tiedot löytyvät aina nopeasti ja helposti samasta paikasta. Työvuorolistojen tulostaminen käy helposti, listat voi tulostamisen lisäksi lähettää myös sähköpostilla työntekijöille. Työntekijät voivat nyt olla mukana täyttämässä toteutuneita työvuoroja.

Sovelluksen kehitys jatkuu yhä. Pääasia on, että välttämättömimmät toiminnot saatiin toimimaan aikarajojen sisällä ja sovellus pääsee todelliseen käyttöön. Asiakas oli kaiken kaikkiaan tyytyväinen tähänastiseen työhön.

Lähteet

1 Ruby. (WWW-dokumentti.) Ruby-lang organization. <http://www.ruby-lang.org/en/about/>. Luettu 18.10.2010.

2 Ruby on Rails. (WWW-dokumentti.) Ruby on Rails organization. <http://guides.rubyonrails.org/getting_started.html>. Päivitetty 12.7.2010. Luettu 24.10.2010.

3 SQLite. (WWW-dokumentti.) SQLite organization. <http://www.sqlite.org/about.html>. Luettu 24.10.2010.

4 Trygve Reenskaug. MVC. (WWW-dokumentti.) <http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>. Luettu 25.10.2010.

5 Fowler, Martin. Patterns of enterprise application architecture. Boston. Addison-Wesley, 2003.

6 Active Record documents. (WWW-dokumentti.) Ruby on Rails organization. <http://ar.rubyonrails.org/>. Päivitetty 27.6.2008. Luettu 27.10.2010.

7 Greg Schurgast. Tableorderer. (WWW-dokumentti.) <http://prototools.negko.com/demo/tableorderer>. Päivitetty 11.6.2009. Luettu 5.10.2010.

8 Scriptaculous. (WWW-dokumentti.) Scriptaculous. <http://script.aculo.us>. Luettu 6.10.2010.

9 Prototype. (WWW-dokumentti.) Prototype. <http://prototypejs.org>. Päivitetty 12.10.2010. Luettu 17.10.2010.

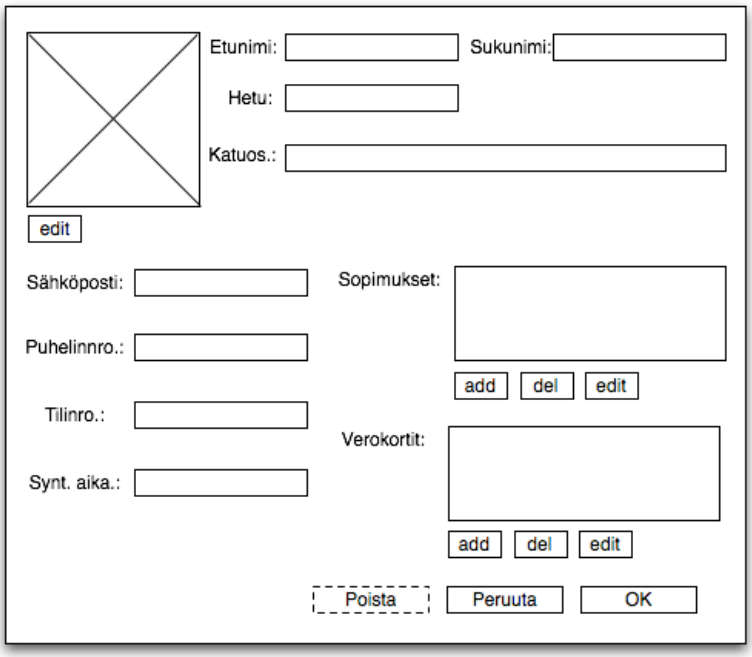
10 Fernandez , Obie. The Rails way. Boston. Pearson Education Inc, 2008.

11 Ruby on Rails security. (WWW-dokumentti.) Ruby on Rails organization. <http://guides.rubyonrails.org/security.html>. Päivitetty 1.11.2010. Luettu 13.11.2010.

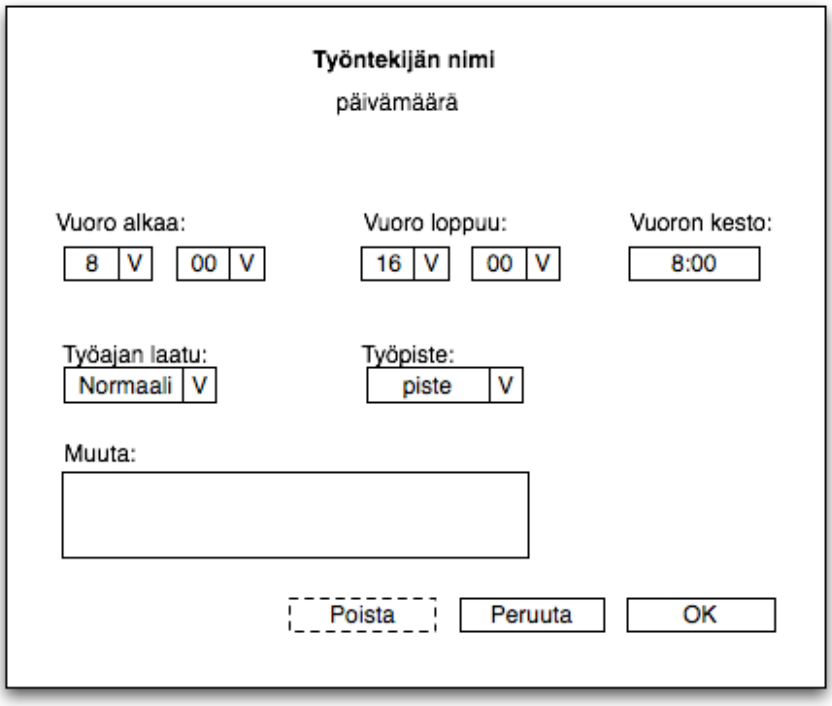
12 Ruby on Rails testing. (WWW-dokumentti.) Ruby on Rails organization. <http://guides.rubyonrails.org/testing.html>. Päivitetty 4.4.2010. Luettu 13.11.2010.

Liitteet
Liite 1: Lomakkeet

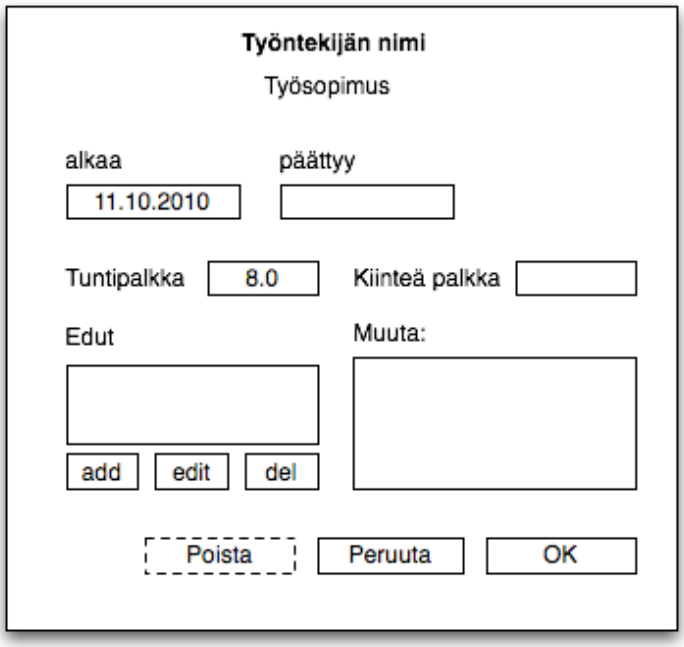
Sisäänkirjautumislomake



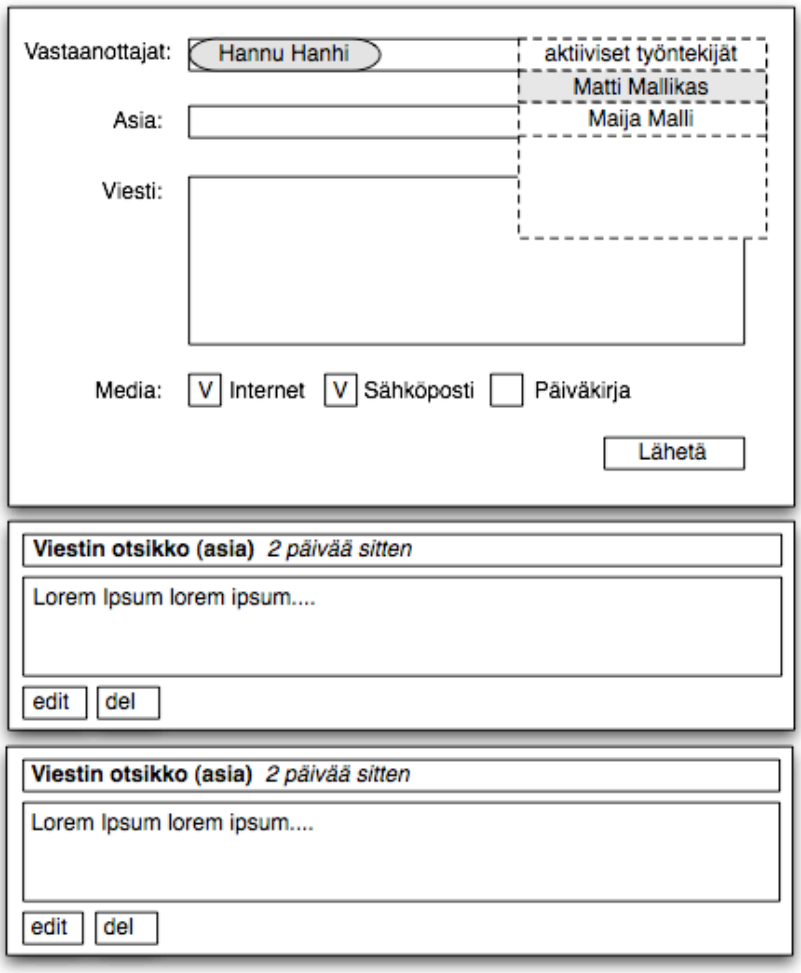
Työntekijälomake



Työvuorolomake



Työsopimuslomake



Viestit-sivu ja viestilomake

Liite 2: Sovelluksen tietokanta (schema.rb).

```
ActiveRecord::Schema.define(:version => 20101109223231) do

  create_table "companies", :force => true do |t|
    t.string   "name"
    t.string   "street"
    t.string   "zip"
    t.string   "phone"
    t.string   "fax"
    t.string   "place"
    t.integer  "ref_image"
    t.string   "company_id"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "contracts", :force => true do |t|
    t.integer  "ref_employees"
    t.date     "starting"
    t.date     "ending"
    t.float    "salary"
    t.string   "description1"
    t.string   "code1"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "employees", :force => true do |t|
    t.string   "name"
    t.string   "firstname"
    t.string   "lastname"
    t.string   "street"
    t.string   "city"
    t.string   "zip"
    t.string   "phone"
    t.string   "email"
    t.integer  "ref_images"
    t.integer  "ref_companies"
    t.integer  "ref_group"
    t.date     "birthdate"
    t.string   "bankaccount"
    t.string   "person_id"
    t.boolean  "currently_hired"
    t.boolean  "is_removed"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "images", :force => true do |t|
    t.string   "filename"
    t.string   "path"
    t.integer  "bytes"
    t.string   "size"
    t.string   "sizename"
    t.integer  "width"
    t.integer  "height"
    t.string   "title"
    t.text     "exif_json"
    t.text     "description"
    t.integer  "ref_parent"
```

Liite 2: Sovelluksen tietokanta (schema.rb).

```
    t.string   "url"
    t.string   "server"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "notifications", :force => true do |t|
    t.text     "message"
    t.string   "title"
    t.string   "media"
    t.integer  "author"
    t.string   "type"
    t.text     "recipients"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "salaries", :force => true do |t|
    t.integer  "season"
    t.integer  "ref_companies"
    t.string   "season_type"
    t.string   "employee_name"
    t.string   "employee_person_id"
    t.string   "employee_bankaccount"
    t.date     "contract_start"
    t.integer  "ref_employees"
    t.float    "plan_total"
    t.float    "real_total"
    t.integer  "year"
    t.date     "starting"
    t.date     "ending"
    t.integer  "workdays"
    t.float    "tnormal"
    t.float    "tfree"
    t.float    "tsick"
    t.float    "tfree_nosalary"
    t.float    "ttraining"
    t.float    "tevening"
    t.float    "tnight"
    t.float    "tsaturday"
    t.float    "tsunday"
    t.float    "pr50"
    t.float    "time_bonus"
    t.float    "season_money"
    t.float    "year_money"
    t.float    "tax_base"
    t.float    "tax_extra"
    t.float    "income_limit"
    t.boolean  "is_signed"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "sessions", :force => true do |t|
    t.string   "session_id", :null => false
    t.text     "data"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
```

Liite 2: Sovelluksen tietokanta (schema.rb).

```
  add_index "sessions", ["session_id"], :name =>
"index_sessions_on_session_id"
  add_index "sessions", ["updated_at"], :name =>
"index_sessions_on_updated_at"

  create_table "shift_types", :force => true do |t|
    t.string   "name1"
    t.string   "name2"
    t.string   "description"
    t.string   "code"
    t.integer  "ref_companies"
    t.float    "unit_factor",   :default => 1.0
    t.float    "unit_add",      :default => 0.0
    t.float    "add_per_go",    :default => 0.0
    t.datetime "created_at"
    t.datetime "updated_at"
  end

  create_table "shifts", :force => true do |t|
    t.date     "shift_date"
    t.integer  "plan_start_h",   :default => 0,  :null => false
    t.integer  "plan_start_min", :default => 0,  :null => false
    t.integer  "plan_end_h",     :default => 0,  :null => false
    t.integer  "plan_end_min",   :default => 0,  :null => false
    t.integer  "real_start_h",   :default => 0,  :null => false
    t.integer  "real_start_min", :default => 0,  :null => false
    t.integer  "real_end_h",     :default => 0,  :null => false
    t.integer  "real_end_min",   :default => 0,  :null => false
    t.integer  "real_minutes",   :default => 0,  :null => false
    t.integer  "real_hours",     :default => 0,  :null => false
    t.integer  "plan_minutes",   :default => 0,  :null => false
    t.integer  "plan_hours",     :default => 0,  :null => false
    t.integer  "ref_employees"
    t.string   "employee_name"
    t.integer  "ref_companies"
    t.integer  "ref_stores_real"
    t.string   "store_name_real"
    t.integer  "ref_stores_plan"
    t.string   "store_name_plan"
    t.float    "unit_salary"
    t.float    "total_salary"
    t.float    "real_units"
    t.float    "planned_units"
    t.float    "extra1_units"
    t.float    "extra2_units"
    t.float    "extra3_units"
    t.float    "extra4_units"
    t.float    "extra5_units"
    t.float    "extra6_units"
    t.string   "shift_type_name_real"
    t.string   "shift_type_name_plan"
    t.integer  "ref_shift_types_real"
    t.integer  "ref_shift_types_plan"
    t.string   "notes", :default => ""
    t.string   "code1"
    t.string   "code2"
    t.string   "code3"
    t.string   "code4"
    t.boolean  "write_lock"
    t.datetime "created_at"
```

Liite 2: Sovelluksen tietokanta (schema.rb).

```
    t.datetime "updated_at"
  end
  create_table "stores", :force => true do |t|
    t.integer  "ref_companies"
    t.string   "name"
    t.string   "name2"
    t.string   "description1"
    t.string   "description2"
    t.string   "code"
    t.boolean  "is_active",  :default => true
    t.datetime "created_at"
    t.datetime "updated_at"
  end
  create_table "transaction_accounts", :force => true do |t|
    t.string   "name1"
    t.integer  "ref_companies"
    t.string   "description1"
    t.string   "code1"
    t.string   "type"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
  create_table "transaction_types", :force => true do |t|
    t.integer  "ref_transaction_accounts"
    t.string   "name1"
    t.string   "description1"
    t.string   "code1"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
  create_table "transactions", :force => true do |t|
    t.integer  "ref_transaction_types"
    t.integer  "ref_transaction_accounts"
    t.string   "description1"
    t.boolean  "cleared"
    t.float    "amount1"
    t.float    "amount2"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
  create_table "users", :force => true do |t|
    t.string   "firstname"
    t.string   "lastname"
    t.string   "publicname"
    t.string   "username"
    t.string   "password"
    t.string   "email"
    t.string   "role"
    t.integer  "ref_employees"
    t.integer  "status"
    t.boolean  "can_login"
    t.datetime "last_login"
    t.integer  "ref_companies"
    t.datetime "created_at"
    t.datetime "updated_at"
  end

end
```