Olli Puurunen

VERKKOMAINEN TIETORAKENNE ORGANISAATION TIEDON HALLINNASSA

Web-sovellus Intopii Oy:n sisäiseen viestintään

VERKKOMAINEN TIETORAKENNE ORGANISAATION TIEDON HALLINNASSA

Web-sovellus Intopii Oy:n sisäiseen viestintään

Olli Puurunen
Opinnäytetyö
Syksy 2013
Tietojenkäsittelyn koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä: Olli Puurunen
Opinnäytetyön nimi: Verkkomainen tietorakenne organisaation tiedon hallinnassa
Työn ohjaaja: Sinikka Viinikka
Työn valmistumislukukausi ja -vuosi: Syksy 2013 Sivumäärä: 39

Opinnäytetyön tavoitteena oli kehittää toimeksiantaja Intopii Oy:lle verkkoselaimessa toimiva järjestelmä, jolla yritys voisi hoitaa kaiken sisäisessä toiminnassaan tarvittavan informaation hallinnan ja näin ollen korvata aikaisemmat useat keskenään yhteensopimattomat järjestelmät yhdellä sovelluksella. Sovelluksen toteutuksen ohella työn tavoitteena oli tutkia, miten organisaatiossa tuotettavaa digitaalista tietoa hallitaan ja samalla pohtia tallennuksessa käytettyjen tietorakenteiden eroja.

Työn tuloksena oli nykyaikaisia tekniikoita käyttäen toteutettu web-sovellus, jossa käyttäjät voivat luoda ja hallita yksinkertaisia muistiinpanoja sekä asiakas- ja yhteystietoja. Lisäksi kaikkea tietoa voidaan jakaa muiden käyttäjien ja käyttäjäryhmien kesken monipuolisen käyttöoikeusjärjestelmän avulla. Mitä tahansa tietoa voidaan liittää muihin tietoihin, joten informaation haku ja organisointi on erittäin helppoa. Työn tietoperustassa käydään läpi, miksi sovelluksessa päädyttiin käyttämään tietokannan rakenteessa verkkomallia tavanomaisen relaatiomallin sijaan.

Sovellusta toteutettaessa huomattiin, että tietynlaisissa sovelluksissa verkkomainen tietorakenne on omiaan, joskaan ihan täysin verkkomaista tietorakenteesta ei voi tehdä tehokkuussyistä. Sovelluksen käyttöliittymän toteutukseen käytetty Google Web Toolkit –kirjasto havaittiin erittäin varteenotettavaksi työkaluksi toteuttaa dynaamisia ja vuorovaikutteisia käyttöliittymiä, jotka toimivat lähes sellaisinaan eri selaimissa. Vaikka Google Web Toolkit lupaakin mainospuheissaan poistavan selainyhteensopivuusongelmat, joitain pieniä seikkoja kehittäjä joutuu kuitenkin korjaamaan kehitystyön aikana.

Iloinen yllätys oli myös se, kuinka moneen sovellus taipuu jo sellaisenaan pelkkiä muistiinpano- ja kontaktielementtejä käyttäen. Jaetun kauppalistan, tehtävälistan tai yhteystietohakemiston tekeminen onnistuu vaivatta ja pienellä ohjelmointityöllä sovelluksesta voi tehdä esimerkiksi kätevän työkalun tutkivan lääkärin tutkimuspotilaiden ja -tulosten hallintaan. Käyttötapaukset ovat lähes rajattomat.

Asiasanat: Web-sovellus, GWT, XML-RPC, tiedonhallinta, verkkomalli

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business Information Systems

Author: Olli Puurunen
Title of thesis: Information Management in Organization Using Network Data Structure
Supervisor: Sinikka Viinikka
Term and year when the thesis was submitted: Autumn 2013 Number of pages: 39

The primary objective of the thesis was to develop a web application for Intopii Oy to manage all information in the company and to replace all the existing, incompatible systems with only one system. Another goal was to investigate how digital information is produced in an organization and also to compare the different data structures used when storing information digitally.

The result of the thesis was a web application created using the latest technologies. In the application, the users can produce and manage simple notes, as well as create contact data and customer information. Furthermore, all data can be shared with other users and user groups using a versatile access control system. Any piece of information can be attached or linked to other information, enabling extremely easy searching and organizing of the data. The theoretical background describes why the network model was chosen as the data structure in the application database instead of the more traditional relational model.

While developing the application, it was discovered that the network data structure is a good choice in certain kinds of applications, although the model is not usually applied completely to secure good performance of the application. The Google Web Toolkit development library was used to implement the user interface of the application and it seems to be an excellent toolkit to create dynamic and responsive user interfaces without the developer having to spend time on different browser quirks. Even though GWT does a great job removing most of the browser incompatibilities, still, some of the issues are left to the developer.

It was a pleasant surprise nice to notice the many different ways the developed application can be used even with the default data types including notes and contacts. A task list, a shared shopping list or even a list of family contact information can be created with ease. With a little coding, the application can be extended, for example, to be a handy tool for a researcher to manage the results and research subjects of the study. The possibilities seem limitless.

Keywords: Web-application, GWT, XML-RPC, data management, network model

SISÄLLYS

SANASTO

Abstrakti luokka	Luokka, josta ei voi luoda ilmentymää ja jonka osa metodeista vaaditaan toteutettavaksi aliluokissa.
ACL	Access Control List, käyttöoikeuslista.
Ajax	Asynchronous JavaScript And XML. Yleensä tarkoitetaan joukkoa web-tekniikoita, joilla web-sovelluksesta saadaan vuorovaikutteisempi.
CRM	Customer Relationship Management, asiakkuudenhallinta.
Elementti	Elementtityypin yksi ilmentymä, esimerkiksi yksi asiakas tietoineen.
Elementtityyppi	Esimerkiksi asiakas, muistiinpano tai henkilö.
HTML	Hypertext Markup Language, kuvauskieli web-sivujen tekoon.
HTTP	Hypertext Transfer Protocol, verkkoselainten ja WWW-palvelinten välillä käytetty tiedonsiirtoprotokolla.
Häntä-elementti	Elementti, josta kahden elementin välinen linkki lähtee.
Kohde-elementti	Elementti, johon kahden elementin välinen linkki osoittaa.
LDAP	Lightweight Directory Access Protocol, protokolla hakemistopalveluiden käyttöön. Yleensä käytetään käyttäjätietojen tallennukseen.
Liitostaulu	Taulu, jolla kaksi (tai useampi) tietokannan taulu yhdistetään sisältäen yleensä kunkin yhdistettävän taulun pääavaimen.
Linkki	Kahden elementin välinen yhteys.
Rajapintaluokka	Luokka, jonka kaikki metodit vaaditaan toteutettavaksi aliluokissa.
SQL	Structured Query Language, kyselykieli relaatiotietokannan käyttöön.
Unix	Palvelintietokoneissa yleinen käyttöjärjestelmä, josta myös suosittu Linux on ottanut vaikutteita.
Wiki	Vapaasti käyttäjiensä muokattavissa oleva verkkosivusto.
XML-RPC	XML:ään pohjautuva tiedonsiirtoprotokolla.

1 JOHDANTO

Nykyään monissa yrityksissä lähes kaikki tieto tallennetaan digitaalisesti. Digitaalisen tiedon määrä kasvaa jatkuvasti ja sen hallintaan ja organisointiin tulee kiinnittää entistä enemmän huomiota, jotta työntekijöiden aika ei kulu tiedonhakuun valtavasta tietomassasta. Tiedon hallintaan on kehitetty runsaasti erilaisia järjestelmiä ja ohjelmistoja, jotka helpottavat sekä tiedon tuottamista että sen järjestelyä ja siitä hakemista. Ongelmaksi kuitenkin saattaa muodostua se, että tiedon hallintaan tarvitaan useita erillisiä järjestelmiä. Tällöin eri järjestelmissä saattaa olla päällekkäisiä tietoja ja tietojen päivitys on vaikeaa ja toisiinsa liittyvän tiedon yhdistäminen eri järjestelmien kesken on haastavaa ja aikaa vievää. Toinen vaihtoehto on hankkia yksi kaiken kattava järjestelmä, jolloin taas käyttäjät saattavat hukkua ylimääräisten ominaisuuksien tulvaan, mikäli ylimääräisiä kustannuksia aiheuttavaa räätälöintiä ei tehdä.

Opinnäytetyön toimeksiantajalla Intopii Oy:llä on käytössään monia täysin erillisiä järjestelmiä, johon yrityksen työntekijöiden, projektien ja asiakkaiden tietoja tallennetaan. Tärkeimmät projektien tiedot ja tiedostot tallennetaan yrityksen verkkolevylle projektikohtaiseen hakemistoon. Tämän lisäksi projektien tietoja on Redmine-tehtävänhallintajärjestelmässä, mutta myös työntekijöiden Hours-tuntiseurantajärjestelmässä. Tuntiseurantaohjelmistossa on myös tietoja asiakkaista ja samoja tietoja löytyy tämän lisäksi yrityksen käytössä olevasta vtiger CRM -asiakkuudenhallintajärjestelmästä.

Opinnäytetyön tarkoituksena oli suunnitella ja kehittää ohjelmisto, jolla pystyttäisiin myöhemmin hallitsemaan kaikkea yrityksessä käytettävää digitaalista informaatiota. Opinnäytetyötä varten ohjelmistosta oli tarkoitus tehdä ainoastaan perusominaisuudet sisältävä versio, jolla pystytään tallentamaan ja hakemaan muistiinpanoja, yhteyshenkilöitä ja asiakkaita. Lisäksi järjestelmään tuli kehittää tuki usealle käyttäjälle ja mahdollisuus muokata minkä tahansa tiedon käyttöoikeuksia sekä yksittäiselle käyttäjälle että käyttäjäryhmälle.

Toimeksiantaja Intopii Oy on vuonna 2000 perustettu oululainen konenäköön ja kuvan analysointiin keskittynyt ohjelmistoyritys. Kun puhutaan konenäöstä teollisuuden käytössä, tarkoitetaan yleensä kamerasta, tietokoneesta, ohjelmistosta ja valaistuksesta koostuvaa järjestelmää, jossa

kameralla otetaan valaistusta kohteesta kuvia, joita analysoidaan tietokoneella olevalla ohjelmistolla. Analysointi voi liittyä kuvauksen kohteena olevan objektin muodon, asennon tai muiden fyysisten ominaisuuksien tarkasteluun. Analysoinnin tuloksen perusteella tehtaan valmistusjärjestelmää voidaan ohjata tekemään haluttuja toimenpiteitä tarkasteltavalle tuotteelle. (Hirvelä 2013, hakupäivä 3.11.2013.)

Yritys on tehnyt ohjelmistotuotteita sekä teollisuuteen että julkiselle sektorille. Yksi tunnetuimmista Intopiin valmistamista sovelluksista lienee vuonna 2007 eduskuntavaaleja silmällä pitäen lanseerattu valtavirrasta poikkeava vaalikone nimeltään Naama-vaalikone, jolla Intopii sai runsaasti positiivista julkisuutta. Muita Intopiin tärkeimpiä tuotteita ovat rainantarkastukseen tarkoitettu järjestelmä Kide, sekä sen pienempi versio Vikavahti. Molemmat järjestelmät ottavat tehtaan tuottamasta tuotteesta jatkuvaa kuvamateriaalia sen edetessä tuotantolinjalla, analysoivat siitä mahdollisesti löytyvät viat ja ongelmat ja raportoivat löytämistään havainnoista esimerkiksi tehtaan omille järjestelmille tai antavat hälytyksen järjestelmän omaa valotornia käyttäen.

Lisäksi Intopiin nimeä on tehnyt tutuksi erityisesti liikuntapaikkojen suoritusmäärän ja käyttöasteen laskentaan tarkoitettu Otos. Otos ottaa liikuntapaikasta (kuten uima-allas tai kuntopolku) kuvia etäohjattavalla kameralla, analysoi ja laskee kuvassa näkyvät ihmiset ja lähettää tiedon verkkopalveluun, josta käyttäjät voivat selkeistä kuvaajista nähdä, milloin esimerkiksi uimahalliin kannattaa mennä välttääkseen pahimman ruuhka-ajan.

2 ORGANISAATION TIETOJEN HALLINTA

Teknologian ja ohjelmistojen kehittyessä tiedon sähköinen tuottaminen on entistä helpompaa ja sähköisiä dokumentteja syntyy yrityksissä kiihtyvää tahtia. Kun dokumentteja ja muuta informaatiota on paljon, kuluu suuri osa toimistotyöntekijän ajasta dokumenttien etsintään. Paitsi että tämä aiheuttaa yrityksille täysin tarpeettomia kuluja nykyisten työntekijöiden osalta, myös uusien työntekijöiden kouluttaminen vie entistä enemmän aikaa ja vaikeuttaa näin yrityksen kasvua. Samoja asioita saatetaan tehdä moneen kertaan, kun haettavaa tietoa ei osata tai jakseta etsiä. (Anttila 2001, 1-3)

Kunnon tietojärjestelmien puuttuessa yrityksissä saatetaan tiedon organisoinnissa turvautua pelkkään tallennusmedian hakemistorakenteeseen. Tämä ei kuitenkaan ole riittävän hyvä tapa, sillä tiedon haku ja erityisesti luokittelu on vaikeaa pelkästään tiedoston nimen ja hakemiston perusteella. Yhden käyttäjän järjestelmissä hakemistorakenteeseen perustuva järjestely saattaa olla riittävä, mutta ongelmia tulee heti, kun järjestelmää käyttää useita käyttäjiä. (Anttila 2001, 3-4)

2.1 Tietojärjestelmät

Informaation hallintaa, organisointia ja käyttöä helpottamaan on kehitetty erilaisia tietojärjestelmiä. Järjestelmien ansiosta käyttäjä voi keskittyä tiedon tuottamiseen ja käyttöön, eikä käyttäjän tarvitse välittää tiedon varsinaisesta sijainnista, jolloin tieto voi olla tallennettuna joko paikallisesti oman koneen tiedostojärjestelmään tiedostoina, tietokantaan taulujen riveinä ja sarakkeina tai vaikkapa verkon välityksellä pilveen. Tiedon tallennusformaatti on usein piilotettu käyttäjältä ja tieto näytetään käyttäjälle miellyttävässä muodossa. Tietomassasta voi tehdä hakuja paitsi sisällön, myös metatiedon avulla. Metatietoa voi olla esimerkiksi tiedon muokkaus- tai luontipäivä tai kuvien tapauksessa jopa kuvassa esiintyvien henkilöiden nimet. Hyvä tietojärjestelmä pitää huolen tiedon automaattisesta versioinnista, jolloin kukin muokkauskerta tallentaa tiedosta historiatiedon, johon käyttäjä voi halutessaan palata milloin tahansa. Useiden käyttäjien ympäristöissä hyödyllisiä ominaisuuksia ovat myös tiedon lukituksen hoitaminen sekä mahdollisten käyttöoike-

uksien hallinta. Kun tieto on lukittu asianmukaisesti käyttäjän sitä käyttäessä, voidaan varmistua siitä, ettei vahingollista ylikirjoitusta tapahdu eikä tietoa pääse häviämään yhtäaikaisten muokkausten johdosta. (Anttila 2001, 4)

Tietojärjestelmien käyttöönotto yrityksissä ei ole kuitenkaan aina ongelmatonta. Erityisesti yrityksissä, joissa tietyt konventiot ovat olleet pitkään käytössä, on vaikeaa muuttaa juurtuneita toimintatapoja. Järjestelmät vaativat yleensä aikaa ja kustannuksia vievää koulutusta organisaation koosta riippuen suurellekin työntekijäjoukolle. Kustannuksia aiheuttaa koulutuksen lisäksi tietysti myös järjestelmien hankinta. Yksi järjestelmä ei välttämättä kata yrityksen tarpeita, jolloin tarvitaan useita järjestelmiä tai yksi massiivinen järjestelmä, johon taas joudutaan tekemään kallista räätälöintiä, jotta se saataisiin sovitettua mahdollisimman ongelmattomasti yrityksen toimintaympäristöön. (Anttila 2001, 4)

2.2 Intopii Oy:n tietojärjestelmät

Opinnäytetyön toimeksiantajalla, Intopii Oy:llä luotetaan toiminnassa vahvasti erilaisiin avoimen lähdekoodin ratkaisuihin. Jo useita vuosia yrityksen asiakkaista on pidetty kirjaa avoimen lähdekoodin vtiger CRM -asiakkuudenhallintajärjestelmällä. Järjestelmään on tallennettu paitsi asiakasyritykset, myös nykyisten ja potentiaalisten asiakkaiden yhteyshenkilöitä. Intopiin käytössä vtiger CRM on kuitenkin ylimitoitettu, sillä järjestelmässä on lukemattomia ominaisuuksia, joita yrityksessä ei kuitenkaan tarvita lainkaan ja tämä ominaisuuksien tulva vaikeuttaa ja hidastaa järjestelmän käyttöä. Toinen Intopiin ydinjärjestelmistä on Oulun seudun ammattikorkeakoulun opiskelijaprojektina teetetty työntekijöiden tuntiseurantajärjestelmä, josta tuntimäärät lähetetään kuukausittain tilitoimistolle. Ongelmana on se, että myös tuntiseurantajärjestelmässä tarvitaan sekä asiakastietoja, että projektitietoja. Koska näiden kahden järjestelmän integrointi ei ole yksinkertaista, täytyy asiakkaiden ja projektien tietojen päivitykset, lisäykset ja poistot tehdä useaan paikkaan.

Edellä mainittujen järjestelmien ohella Intopiillä on käytössä DokuWiki-wikiohjelmisto, johon on kirjattu yrityksen yleisiä käytäntöjä ja toimintaohjeita. Intopii käyttää myös avoimen lähdekoodin Redmine-projektinhallintajärjestelmää. Redminessä on tallennettuna projektien lisäksi kuhunkin projektiin liittyvät tehtävät ja myöhemmin sinne on lisätty myös lisäosat wikille, asiakkaille ja yhteyshenkilöille. Redminen CRM-lisäosa ei ole kuitenkaan niin monipuolinen kuin yrityksen käytös-

sä oleva vtiger CRM -järjestelmä, joten yrityksessä ei ole voitu kokonaan luopua vtiger CRM:n käytöstä. Koska Redminen wikisivut ovat projektikohtaisia, on DokuWiki-järjestelmään pitänyt jättää osa yrityksen ohjedokumenteista, kun taas Redminen wikissä on pelkästään projekteihin liittyvät ohjeet. Ongelmaa on yritetty korjata luomalla Redmineen oma wiki-projekti, johon kaikki projekteihin liittymättömät ohjeet kirjataan.

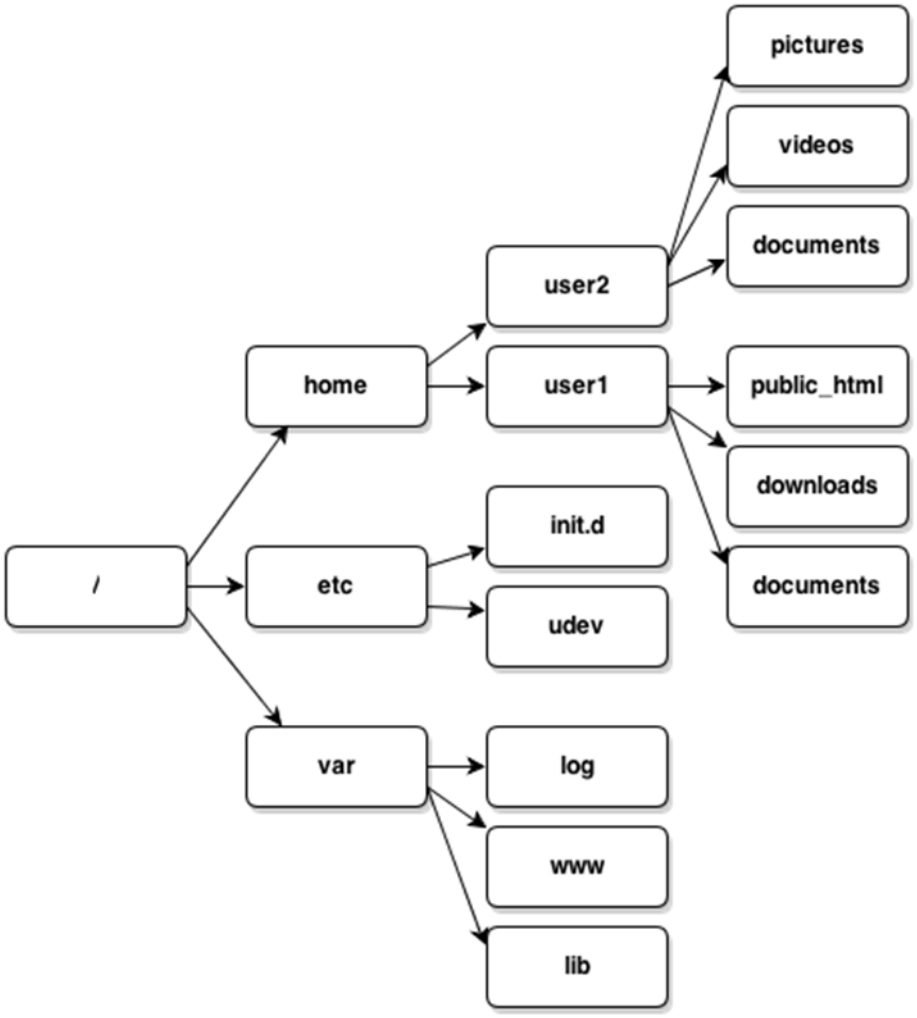
Projekteihin kiinteästi liittyvät tiedostot tallennetaan yrityksen omassa sisäverkossa olevan verkkolevyn projektihakemistoon niin, että kullakin projektilla on oma hakemistonsa, jossa on yrityksen oman käytännön mukaiset alihakemistot. Tämäkin käytäntö on yritetty siirtää Redmine-järjestelmään, jossa voi kuhunkin projektiin liittää mitä tahansa tiedostoja. Redminessä on myös mahdollisuus projekti-/tehtäväkohtaiseen ajankäytönseurantaan, mutta ominaisuus ei ole niin monipuolinen kuin yrityksen omassa tuntiseurantajärjestelmässä, joten sen käyttöä ei ole yrityksessä edes harkittu.

Opinnäytetyössä lähdettiin siis ratkomaan tätä toimeksiantajan monen järjestelmän ongelmaa suunnittelemalla ja toteuttamalla sovellus, jolla voitaisiin hoitaa kaiken organisaatiossa syntyvän tiedon hallinta. Tavoitteena oli saada aikaan mahdollisimman monipuolinen järjestelmä, jossa ei olisi rajoitteita sen suhteen, mitä tietoa voitaisiin liittää mihinkin tietoon. Jos sovellusta myöhemmin laajennettaisiin uusilla ominaisuuksilla, pitäisi tämän olla ennen kaikkea vaivatonta ja nopeaa kuitenkin niin, että se ei rikkoisi aikaisempaa toiminnallisuutta eikä jo olemassa olevia tietokannan tauluja tarvitsisi lähteä muokkaamaan.

Kun sovelluksen tietokantarakennetta ryhdyttiin suunnittelemaan, huomattiin perinteisen relaatiomallin sopimattomuus toteutettavaan sovellukseen, sillä mikäli mitä tahansa tietoa haluttaisiin yhdistää mihin tahansa toiseen tietoon, jokaisella elementtityypillä (kuten asiakas tai muistiinpano) tulisi olla tietokannassa erillinen liitostaulu kaikkien muiden elementtityyppien kanssa ja tietokannan rakenteesta tulisi erittäin monimutkainen. Myös tietokantaan suoritettavat kyselyt olisivat vaikeita muodostaa ja uusien ominaisuuksien ja elementtityyppien lisääminen aiheuttaisi runsaasti ongelmia tietokannan päivityksen yhteydessä. Tästä syystä sovelluksen tietokantarakenteesta päätettiin muodostaa verkkomainen.

2.3 Hierarkkinen tietorakenne

Tavallisesti toisiinsa jollakin tapaa linkitetty tieto esitetään hierarkkisena puurakenteena. Hyviä esimerkkejä ovat muun muassa kuviossa 1 havainnollistettu tietokoneen hakemistorakenne, Internet-sivujen HTML-koodi sekä vaikkapa ihan tavallinen sukupuu. (Mäenpää 2007, 3.) Rakenne lähtee ylimmäisestä solmusta, jota kutsutaan juurisolmuksi. Juurisolmulla on lapsisolmuja, joilla taas on lapsisolmuja ja niin edelleen. Esimerkiksi kuviossa 1 käyttäjän ”user 2” videoita löytyy hakemistorakenteen kohdasta /home/user2/videos.



KUVIO 1. Esimerkki Unix-järjestelmän hakemistorakenteesta

Hierarkkinen puurakenne onkin varsin toimiva aina tiettyyn pisteeseen saakka. Esimerkkinä voisi olla vaikkapa tilanne, jossa halutaan löytää käyttäjän X projektiin Y lisäämät kuvat. Ongelmana on se, että kuvien tallennukselle on aina vähintään kaksi eri kohdetta, jotka kummatkin ovat täysin loogisia. Kuvien tallennushakemistona voisi toimia /projektit/Y/kuvat/X, /kuvat/projektit/Y/X tai jopa

/kayttajat/X/kuvat/projektit/Y. Mikä tahansa näistä vaihtoehdoista olisi täysin hyväksyttävä, mutta kussakin on omat hyvät ja huonot puolensa.

Unix-järjestelmissä ongelmaa on lähdetty ratkomaan linkeillä, jolloin samaan tietoon pääsee käsiksi useista tiedostojärjestelmän eri osista. Tiedostoja voi linkittää sekä kovilla että symbolisilla linkeillä. Kova linkki viittaa suoraan alkuperäisen tiedoston osoittamaan tietoon, kun taas symbolinen linkki viittaa linkitetyn tiedoston tai hakemiston polkuun (Bozidar 2002, 44). Ratkaisu linkkejä käyttämällä on varsin toimiva siihen saakka, kunnes tietoa ryhdytään poistamaan. Mikäli symbolisia linkkejä on luotu useita samaan kohteeseen, tulee ne kaikki poistaa varsinaisen kohteen poistamisen jälkeen. Muussa tapauksessa järjestelmä täyttyy vähitellen kuolleista linkeistä. Mikäli kyseessä taas on kova linkki, tieto säilyy järjestelmässä, vaikka sen olisikin kuulunut jo poistua. (Mäenpää 2007, 3.)

2.4 Verkkomainen tietorakenne

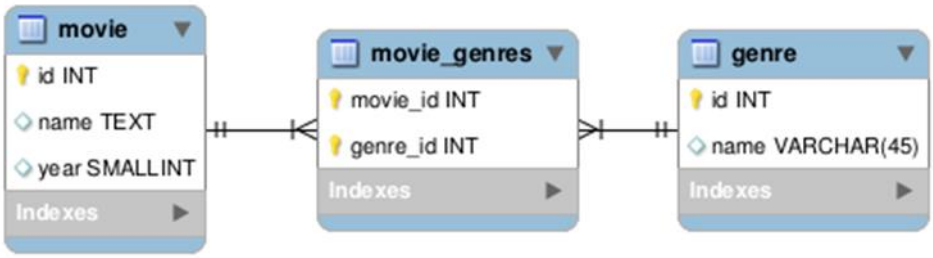
Hierarkkisen tietorakenteen rajoituksista johtuen verkkomainen tallennusrakenne on monissa tapauksissa parempi vaihtoehto. Esimerkiksi hakemistorakenteen ollessa kyseessä hakemistolla A voi olla yhden ylähakemiston sijaan useita ylähakemistoja ja aiemmassa kuvaesimerkissä kuvat löytyisivät mistä tahansa esimerkin hakemistovaihtoehdoista.

3 TOTEUTUSPROSESSI

Toteutusprosessi alkoi sovelluksen toteutukseen käytettävien välineiden arvioinnilla ja vertailulla. Välineiden ja tekniikoiden valintaan piti kiinnittää erityisen suurta huomiota, sillä sovelluksen luonteen huomioon ottaen vääristä ratkaisuista olisi tullut jatkossa valtava ongelma ei pelkästään käytön vaan myös jatkokehityksen kannalta. Kun sovelluskehitykseen käytettävät välineet oli valittu ja sovellukselle esitetyt vaatimukset määritelty, oli varsinaisen toteutuksen vuoro.

3.1 Tietokantarakenteen suunnittelu

Tavallisesti tietokannan taulujen yhteydet ovat yhden suhde moneen (1:N) -yhteyksiä eli yhtä riviä tietokannan taulussa vastaa N riviä toisessa taulussa. Esimerkiksi tietokannassa voisi olla taulut ”asiakas” ja ”projekti”. Asiakastaulussa listataan kaikki yrityksen asiakkaat ja projektitaulussa kaikki eri asiakkaiden projektit. Yhdellä asiakkaalla voi olla monta projektia, joten tauluilla sanotaan olevan yhden suhde moneen -yhteys. Joskus tulee kuitenkin tilanteita, joissa yhtä tietokannan ensimmäisen taulun riviä vastaa toisessa taulussa useita rivejä ja toisen taulun yksittäistä riviä vastaa taas ensimmäisessä taulussa useita rivejä. Tällöin puhutaan monen suhde moneen (M:N) -yhteydestä ja esimerkkinä siitä on esimerkiksi tilanne, jossa on taulut ”movie” ja ”genre”. Yksi elokuva (movie) voi kuulua useaan genreen (kuten toiminta ja komedia) ja yhteen genreen voi kuulua useita elokuvia. Tietokannan rakennetta suunniteltaessa monen suhde moneen -yhteyksissä luodaan yleensä kuviossa 2 havainnollistettu liitostaulu, johon laitetaan sarakkeet molempien liitettävien taulujen pääavaimille.



KUVIO 2. Esimerkki tilanteesta, jossa taulut yhdistetään erillistä liitostaulua käyttäen.

Kuvion 2 tietokantarakennetta mukaillen tietokannan sisältö voisi olla kuvion 3 kaltainen, jossa kuvataan elokuvatietokannan sisältö. Tietokannassa on kolme elokuvaa ja kolme genreä. Kullakin elokuvalla on useita genrejä, joten elokuvan ja genren yhteys tallennetaan liitostauluun. Esimerkiksi elokuva Nykyaika kuuluu lajityyppeihin komedia ja draama.

Movie

Id	Name	Year
1	Rita Hayworth - avain pakoon	1994
2	Millerit	2013
3	Nykyaika	1936

Movie_genres

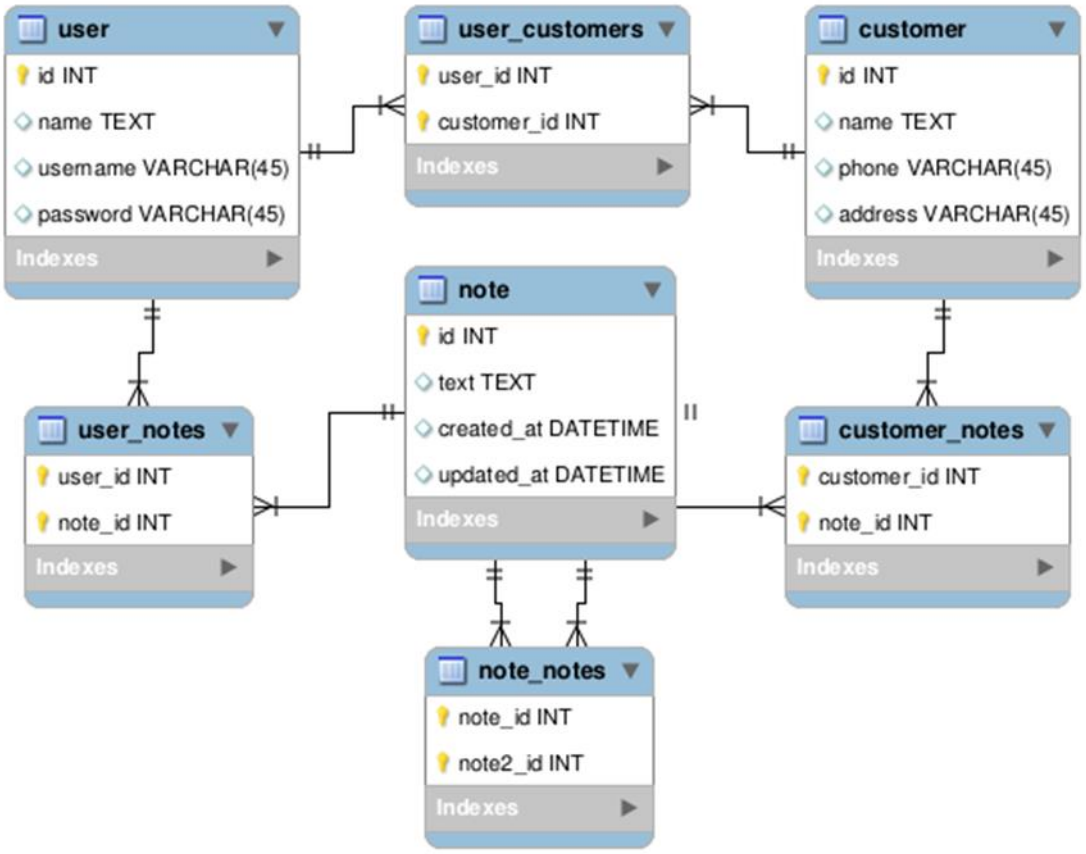
Movie_id	Genre_id
1	1
1	2
2	1
2	3
3	3
3	2

Genre

Id	Name
1	Rikos
2	Draama
3	Komedia

KUVIO 3. Elokuvatietokanta

Opinnäytetyön sovelluksessa tilanne ei ole kuitenkaan näin yksinkertainen. Sovelluksessa on esimerkiksi käyttäjiä (user), muistiinpanoja (note) ja asiakkaita (customer). Kullakin käyttäjällä voi olla useita muistiinpanoja ja samoja muistiinpanoja pystyy linkittämään myös muihin tietoihin, kuten toisiin muistiinpanoihin tai asiakkaisiin. Kuten kuviosta 4 käy ilmi, muodostuu sovelluksen tietokantarakenteesta nopeasti kaoottinen, mikäli tauluja ryhdytään yhdistämään perinteisellä tavalla liitostauluja käyttäen. Lisäksi kyseisellä rakenteella ei voida määrittää tietojen omistussuhdetta. Oletetaan, että käyttäjällä on kaksi toisiinsa liittyvää muistiinpanoa, ”kauppalista” ja ”kurkku”. Tällä tietorakenteella sovelluksen ei ole mahdollista tietää, kuuluuko kurkku kauppalistan alle vai toisinpäin, vaikka se ihmiselle onkin ilmiselvää. Kun sovellukseen lisätään uusi elementtityyppi, joudutaan tietokantaan lisäämään elementin oman taulun lisäksi neljä liitostaulua.

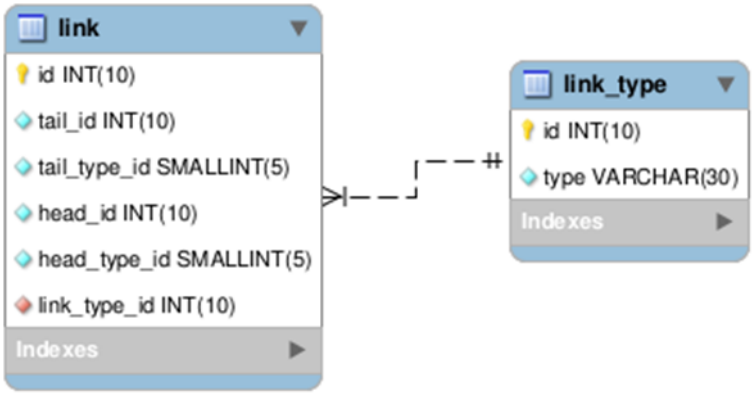


KUVIO 4. Monen suhde moneen -yhteys liitostauluja käyttäen

Kun tavallinen relaatiotietokantarakenne koostuu tauluista ja niiden yhteyksistä, määritellään verkkomaisen tietokantarakenteen koostuvan kokoelmista tietueita, joiden suhteet määritellään linkeillä. Tietue sisältää nimettyjä kenttiä, jotka taas sisältävät arvoja. Linkki määrittää täsmälleen kahden eri tietueen välisen suhteen. Verkkomainen malli mahdollistaa relaatiotietokantaa luonnollisemman tavan kuvata eri asioiden välisiä suhteita. (Silberschatz, Korth & Sudarshan 2010, 1.)

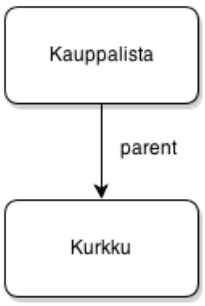
Aiemmin mainituista ongelmista johtuen sovelluksen tietokantarakenne päätettiin toteuttaa relaatiomallin ja verkkomallin yhdistelmänä. Sovellus käyttää relaatiotietokantaa niin, että tietokannan taulu vastaa verkkomallin tietuekokoelmaa tietokannan sarakkeiden vastatessa tietueen kenttiä. Lisäksi elementtien väliset suhteet tallennetaan erilliseen linkkitauluun kuvion 5 esittämällä tavalla. Linkkitaulun yhdellä rivillä määritellään kahden eri elementin välinen yhteys. Kullakin linkillä on linkin yksilöivä juokseva numero (id). Toinen sarake (tail_id) määrittelee sen elementin id-numeron, josta linkki lähtee (häntäelementti) ja sarake tail_type_id taas kertoo häntäelementin elementtityypin (kuten muistiinpano tai asiakas). Sarake head_id kertoo sen elementin id-

numeron, johon linkki kohdistuu (kohde-elementti) ja head_type_id määrittelee kohde-elementin elementtityypin.



KUVIO 5. Sovelluksen linkkitaulut

Edellä mainittujen sarakkeiden lisäksi linkkitaulussa on vielä sarake link_type_id, joka kertoo, minkä tyyppisestä linkistä on kysymys. Erilaisia linkkityyppejä on esimerkiksi elementin omistajan kertova owner (omistaja) ja elementin isäelementin määrittävä parent (vanhempi). Kahden elementin välillä voi olla useita linkkejä, kunhan kunkin linkin tyyppi on eri. Kuviossa 6 on esimerkki kahden elementin välisestä yhteydestä. Kummankin elementin elementtityyppi on muistiinpano. Koska linkki lähtee Kauppalista-elementistä, on kyseessä häntäelementti. Kohde-elementti taas on linkin kohde, eli Kurkku-elementti. Linkin suunta määrittää elementtien omistussuhteen, eli tässä tapauksessa Kauppalista on Kurkun isäelementti.



KUVIO 6. Kahden elementin välinen yhteys

Tällä toteutustavalla sovelluksen jatkokehitys on helppoa. Kun sovellukseen halutaan lisätä uusi elementtityyppi, ei olemassa olevia tauluja tarvitse muokata, vaan ainoastaan luoda uusi taulu uusien elementtien tietojen tallentamiseksi. Tietokantakyselyiden muodostaminen linkkitaulua käyttäen on suoraviivaista. Yhtä elementtiä tarkastellessa on yksinkertaista hakea, mitkä elemen-

tit linkittävät kyseiseen elementtiin ja vastaavasti mihin elementteihin valitusta elementistä kohdistuu linkki.

3.2 Luokkahierarkian suunnittelu

Tietokannan suunnittelun jälkeen seuraava looginen askel oli hahmotella palvelinsovelluksen luokkahierarkiaa. Jokaisen sovelluksen elementtityypin tulee toimia tiettyjen sääntöjen mukaan, joten oli luonnollista luoda ensin yksi rajapintaluokka (PiiElement), jonka kaikkien elementtityyppien tulisi toteuttaa. Koska kaikki elementit toteuttavat suurimman osan rajapinnan metodeista täsmälleen samalla tavoin, oli järkevää tehdä yksi abstrakti yläluokka (PiiElementBase), josta varsinaiset elementtityyppiluokat periytyvät. Luokkien Pii-etuliite tulee toimeksiantajan omista nimeämiskäytännöistä.

Suurin osa sovelluksen tiedoista on tallennettu SQL-tietokantaan. On kuitenkin mahdollista, että tietoja haetaan myös muista tallennusjärjestelmistä, kuten tiedostojärjestelmästä tai LDAP-hakemistosta. Tällöin tiedon tallennus ja haku tapahtuvat eri tavoin verrattuna tietokantaan tallennettuun tietoon. Tästä syystä jokaiselle tietolähteelle tarvittiin oma abstrakti yläluokkansa, jotka hoitavat tiedon tallennuksen ja lukemisen oman tietolähteen vaatimalla tavalla. Opinnäytetyön laajuutta ja toimeksiantajan vaatimuksia ajatellen tarvittavat yläluokat ovat SQL-tietokannan kanssa kommunikoinnin hoitava PiiSqlElement, sekä LDAP:n kanssa kommunikoiva PiiLdapElement-luokka.

Tietokantaan tietoa tallennettaessa elementit on helppo identifioida käyttämällä tiedonhallintajärjestelmän tarjoamaa juoksevaa, uniikkia numeroa. MySQL-tiedonhallintajärjestelmässä juokseva tunnistenumero voidaan luoda käyttämällä taulun sarakkeessa AUTO_INCREMENT-määrettä (Nixon 2009, 244). Muualle kuin tietokantaan tallennetun tiedon yksilöinti ei kuitenkaan onnistu niin helposti, sillä esimerkiksi tiedostojärjestelmään tallennettu tieto yksilöidään tiedostopolun perusteella. Tämä aiheuttaa ongelmia elementtien välisiä yhteyksiä luotaessa. Elementtien väliset yhteydet tallennetaan tietokannan linkkitauluun, jossa kerrotaan linkin häntäelementin tyyppi ja id-numero, linkin kohde-elementin tyyppi ja id-numero ja lisäksi linkin tyyppi. Elementin tyyppi on tallennettu kokonaislukuna, joka määritellään tyyppikohtaisesti sovelluksen asetustiedostossa. Linkin tyyppi taas on viittaus link_type-tauluun, jossa eri linkkityypit on määritelty. Linkkitaulussa kaikkien sarakkeiden arvot ovat kokonaislukuja lähinnä tilan säästämiseksi ja yhtenäisen taulura-

kenteen varmistamiseksi. Jotta muista tietolähteistä haetut elementtien tiedot saataisiin sopimaan tähän tietokantarakenteeseen, päätettiin tietokantaan luoda erityiset kartoitustaulut kutakin tietokannan ulkoista elementtityyppiä varten. Kartoitustauluun tallennetaan elementin alkuperäinen yksilöivä tieto (kuten tiedostoilla tiedostopolku) ja tiedonhallintajärjestelmältä automaattisesti saatu AUTO_INCREMENT-tyyppinen juokseva numero. Tällöin linkkitauluun voidaan tallentaa kokonaisluku myös tietokannan ulkopuolelta haettavien elementtien osalta.

3.3 Käyttötapaukset

Tässä vaiheessa oli hyvä ryhtyä suunnittelemaan erilaisia käyttötapauksia. Koska yksi tyypillisimmistä sovelluksen käyttötavoista on luoda tavallinen muistiinpano (note), oli ensimmäinen käyttötapaus loogisinta rakentaa tuon nimenomaisen toiminnan tukemiseksi.

Esimerkki käyttötapauksesta ”Note-elementin luonti”:

1. Asiakas lähettää XML-RPC -pyynnön ”createElement('note', properties)”.
2. Haetaan käyttäjä-olio istuntomuuttujasta. Jos käyttäjää ei löydy, heitetään poikkeus.
3. Luodaan Note-olio.
4. Tallennetaan Note-elementti tietokantaan käyttäen Note-olion tietoja ja heitetään poikkeus, mikäli tallennus epäonnistui.
5. Tehdään käyttäjästä juuri luodun Note-elementin omistaja. Jos omistajan lisääminen epäonnistui, poistetaan Note-elementti ja heitetään poikkeus.
6. Jos kaikki sujui kuten pitääkin, palautetaan luodun elementin id muodossa X:Y, missä X on luodun elementin tyyppi-id ja Y on elementin id. Jos aiemmin on heitetty poikkeus, palautetaan FALSE.

Käyttötapaukset luotiin aluksi varsin yleisellä tasolla ja niitä tarkennettiin jatkuvasti kehitystyön edetessä. Kun käyttötapaukset määriteltiin tarpeeksi tarkasti, oli niiden pohjalta helppo aloittaa varsinainen ohjelmointityö.

4 VÄLINEET JA VALINTAKRITEERIT

Kun sovellusta lähdettiin suunnittelemaan, tärkeimmiksi vaatimuksiksi valittiin nopeus, helppokäyttöisyys ja intuitiivisuus. Sovellusta tultaisiin käyttämään tiiviisti työpäivän aikana, joten käytön tulisi olla sujuvaa ja vaivatonta. Ohjelmiston muodoksi valittiin WWW-sovellus, jotta käyttö onnistuisi mistä tahansa lähes millä laitteella tahansa ja lisäksi itselläni sovelluksen kehittäjänä oli eniten kokemusta juuri verkkosovellusten kehittämisestä. Ohjelmiston kehittämisessä päätettiin käyttää mahdollisimman paljon valmiita komponentteja, sillä pyörää on turha keksiä uudelleen. Lisäksi selainyhteensopivuusongelmien välttämiseksi oli tarkoitus valita joku valmis ohjelmistokirjasto, joka ratkaisee kyseiset ongelmat ilman, että kehittäjän tulee käyttää siihen omaa aikaansa.

4.1 Välineiden valinta

Sovelluksen luontiin käytettäviä välineitä valittaessa tarkoitus oli löytää sopivat ohjelmointikielet ja -kirjastot sekä käyttöliittymän ohjelmointiin, että palvelimella ajettavan sovelluksen kehitystä varten. Tässä vaiheessa oli myös tarkoitus valita palvelimella ajettavan sovelluksen ja selaimessa käytettävän käyttöliittymän välillä kulkevan tiedon formaatti.

Staattisia HTML-sivuja käytettäessä selain tekee HTTP-pyynnön palvelimelle, jossa sivut sijaitsevat. Staattisella sivulla tarkoitetaan WWW-sivua, jossa ei ole minkäänlaista dynaamista vuorovaikutusta käyttäjän kanssa esimerkiksi lomakkeiden muodossa. Palvelinohjelmisto (kuten Apache tai Nginx) ottaa pyynnön vastaan ja lähettää vastauksena pyydetyn sivun HTML-koodin, jonka selain muotoilee ja näyttää käyttäjälle WWW-sivun muodossa. Dynaamisia toimintoja sisältäviä sivuja käytettäessä palvelimella suoritetaan kehittäjän luomaa ohjelmakoodia, jonka tuloksen palvelinohjelmisto lähettää takaisin selaimelle. (Nixon 2009, 2-4.) Tavallisesti sivun sisältö vaihtuu, kun selain tekee pyynnön palvelimelle sivunlatauksen yhteydessä. Jos käytön nopeus ja reaaliaikaisuus ei ole tavoitteena, on tämä tekniikka täysin sopiva ja käyttökelpoinen tapa hoitaa kommunikaatio selaimen ja palvelimen välillä. Opinnäytetyön aiheena olevassa sovelluksessa kuitenkin yksi tärkeimmistä kriteereistä oli sovelluksen käytön sujuvuus, joten tiedon haku ja päivitys sivulatauksen yhteydessä ei tullut kysymykseen.

Vielä vuosituhannen alkupuolella JavaScript-komentosarjakieltä käytettiin verkkosivuilla lähinnä yksinkertaisiin seikkoihin, kuten lomakkeiden tarkistukseen ja erilaisiin visuaalisiin tehosteisiin. Vuonna 2005 joka puolella web-kehittäjien yhteisössä ryhdyttiin kuuluttamaan muotisanaa Ajax (Asynchronous JavaScript and XML, Asynkroninen JavaScript ja XML), jolla yleensä tarkoitetaan tiedon välitystä selaimen ja palvelimen välillä tausta-ajona ilman sivulatausta. Tekniikka sinällään ei ollut millään tavalla uusi, sillä jo vuonna 1999 Microsoft esitteli tuolloin uudessa Internet Explorer 5 -selaimessaan Ajax:n tiedonvälityksessä käytettävän XMLHttpRequest-olion. Tuolloin olio käytti Internet Explorer -selaimen omaa ActiveX-tekniikkaa, mutta vasta kun XMLHttpRequest-olio tuli osaksi muiden selaimien JavaScript-tulkkia, alkoi kyseisen tekniikan suosio kasvaa. (Nixon 2009, 377-378.)

Näistä lähtökohdista lähdimme tarkentamaan sovelluksen kehittämisessä käytettävien välineiden vaatimuksia. Sovelluksen tiedot tultaisiin tallentamaan tietokantaan ja palvelinsovelluksen ohjelmointikieleksi valittaisiin jokin sopiva ja kehittäjälle tuttu komentosarjakieli. Palvelimen ja käyttöliittymän kommunikointi tulisi tapahtumaan Ajaxin avulla ja tiedonsiirtomuodon tulisi olla jokin standardoitu ja hyväksi jo aiemmin todistettu formaatti.

4.2 Käyttöliittymä

Sovellusta suunniteltaessa järkevät vaihtoehdot sovelluksen käyttöliittymän toteutukseen oli laskettavissa yhden käden sormin. Nyt vuosia myöhemmin erilaisia varteenotettavia kirjastoja ja viitekehyksiä on tullut kymmeniä uusia. Valitsin ennen varsinaisen kehitystyön alkua lopulliseen vertailuun kolme tuolloin merkittävintä teknologiaa, joissa jokaisessa oli omat hyvät ja huonot puolensa.

JavaFX

JavaFX on Sun Microsystemsin vuonna 2007 julkistama tuoteperhe, joka suunniteltiin yksinkertaistamaan Java-asiakassovellusten tekoa kuluttajalaitteisiin (selaimet, matkapuhelimet, televisiot) monimutkaisten Java Swing ja Java 2D -kirjastojen tilalle (Weaver, Gao, Chin, Iverson & Wos 2012, 1-2).

JavaFX:n hyvät ominaisuudet:

- JavaFX:llä tehdyt sovellukset toimivat useilla laitteilla ja alustoilla ilman että kehittäjän tarvitsee ne erikseen kyseiselle alustalle sovittaa.
- JavaFX sisältää valmiiksi monipuolisen kirjaston valmiita käyttöliittymäkomponentteja (Weaver ym. 2012, 1-4.).

JavaFX:n puutteet:

- Vaatii kohdejärjestelmältä suuren JRE:n (Java Runtime Environment) toimiakseen. Java SE 6 Update 10 -päivityksen jälkeen tätä ongelmaa ei tosin enää ole ollut. (Weaver ym. 2012 1-2.)
- JavaFX:n esimerkkisovelluksia tutkittaessa huomattiin käyttöliittymän sulavuudessa samankaltaisia ongelmia kuin tavallisten Java-sovellusten kanssa.
- JavaFX on vasta varhaisessa kehitysvaiheessa, joten ongelmia ja puutteita varmasti ilmenee.

OpenLaszlo

OpenLaszlo on vuonna 2004 alkunsa saanut ohjelmistoalusta, joka kehitettiin helpottamaan sovelluskehitystä verkkoon. OpenLaszlo-sovelluksia tehtäessä ohjelmointi tapahtuu alustan omalla XML-pohjaisella LZX-kielellä, jonka alusta kääntää haluttuun ympäristöön, joko Flash-formaattiin tai DHTML:ksi. (Laszlo Systems 2006, 4, hakupäivä 28.8.2013.)

OpenLaszlon hyvät ominaisuudet:

- Alustan XML:n ja JavaScriptin yhdistävä LZX-ohjelmointikieli vaikuttaa erittäin mielenkiintoiselta.
- Kehitystyökaluna toimii monipuoliseen Eclipse-ohjelmointiympäristöön perustuva IDE4Laszlo (Laszlo Systems 2006, 15, hakupäivä 23.11.2013).
- Sun Microsystems on kehittämässä Orbit-nimistä kääntäjää, jolla on mahdollista kääntää OpenLaszlo-sovellukset Java ME –alustalle (Laszlo Systems 2006, 13, hakupäivä 23.11.2013).

OpenLaszlon puutteet:

- OpenLaszlo ei ole kovinkaan suosittu kehittäjien keskuudessa.

- Osa LZX-ohjelmointikielen funktioista on suunnattu tiettyyn kohdeympäristöön (Flash tai DHTML).

Google Web Toolkit

Google Web Toolkit (GWT) on Googlen luoma, vuonna 2006 ensimmäisen versionsa saanut web-kehittäjän työkalupakki. GWT:tä käytettäessä sovelluskehittäjä ohjelmoi Java-kielellä, jonka GWT-kääntäjä kääntää ja optimoi JavaScript-kielelle. GWT on perustuu avoimeen lähdekoodiin. (Tacy, Hanson, Essington. & Tökke 2013, 3-4.)

GWT:n hyvät ominaisuudet:

- Sai heti alussa kehittäjien keskuudessa suuren suosion ja käyttäjiä on runsaasti edelleen.
- Koska käyttäjiä on paljon ja taustalla on GWT:tä itsekin käyttävä valtava Internet-jätti Google, ovat tulevaisuuden näkymät hyvin valoisat.
- GWT:lle löytyy runsaasti kolmannen osapuolen kirjastoja ja skriptejä.
- Päivitys- ja julkaisutahti on ollut kiitettävän tiheää.

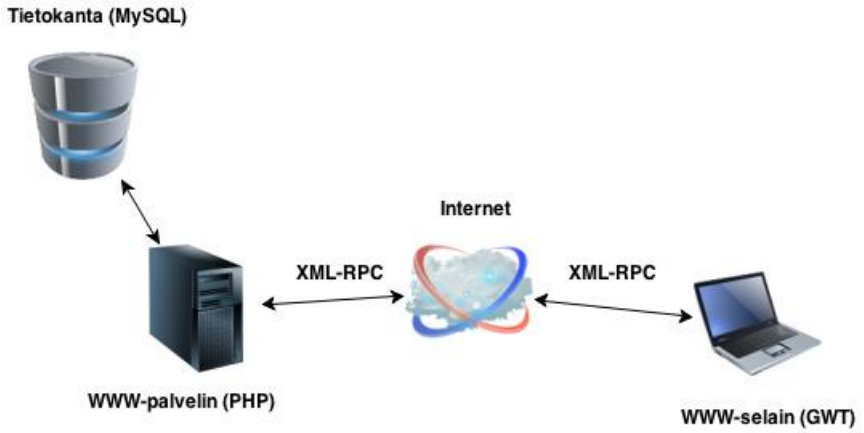
GWT:n puutteet:

- GWT ei itsessään tarjonnut ominaisuuksia luoda sivuista mobiiliversioita.
- GWT:ssä on mahdollista käyttää vain osaa Javan standardikirjastosta.
- Vaikka GWT:n pitäisi ratkaista selainyhteensopivuusongelmat, on silläkin jonkin verran vaikeuksia saada sovellukset toimimaan täsmälleen samalla tavoin eri selaimilla.
- Web-sovelluksen kehittäminen mahdollisimman paljon työpöytäsovelluksen kaltaiseksi ei ole täysin ongelmatonta. Näin on myös GWT:n kanssa. Useissa tilanteissa yksinkertaisen toiminnon tekeminen vaatii turhauttavan paljon ohjelmakoodia.

4.3 Palvelin

Palvelinsovelluksen ohjelmointikieleksi valittiin PHP, sillä itselläni oli siitä eniten kokemusta jo etukäteen, eikä suunnitteluvaiheessa katsottu tarpeelliseksi opetella uutta kieltä tätä projektia varten. Lisäksi PHP:lle löytyy valtava määrä kolmannen osapuolen skriptejä, joista sovelluksessa käytössä on PEAR (PHP Extension and Application Repository) -skriptikirjaston sisältämiä laa-

jennoksia. Sovelluksen tietokannaksi valittiin toimeksiantajalla yleisesti käytössä ollut MySQL-tiedonhallintajärjestelmä.



KUVIO 7. Sovelluksessa käytetyt teknologiat.

Palvelimen ja käyttöliittymän välisen tiedon välitysmuodoksi valittiin XML-RPC –protokolla. XML-RPC oli tuolloin paras vaihtoehto, sillä sitä oltiin käytetty useassa eri projekteissa aikaisemmin ja lisäksi siihen löytyi apukirjastot sekä PHP:lle, että GWT:lle. Kuviossa 7 näkyy sovelluksessa käytetyt teknologiat yhteenvetona.

4.4 Valitut välineet

Sovellusta kehitettäessä eniten perehtymistä vaativat osa-alueet olivat sovelluksen käyttöliittymän tekoon käytetty Google Web Toolkit, sekä käyttöliittymän ja palvelinohjelmiston välisen kommunikoinnin protokollaksi valittu XML-RPC. Näistä ensimmäinen oli itselleni täysin uusi tuttavuus, kun taas jälkimmäistä oli käytetty yrityksen sisäisissä projekteissa aikaisemminkin.

4.4.1 Google Web Toolkit

Kun ensimmäinen versio GWT:stä julkaistiin vuonna 2006, oli web-kehittäjän elämä varsin erilaista nykyhetkeen verrattuna. Vaikka kehitystyötä ja selainyhteensopivuusongelmia helpottavia apukirjastoja oli jo olemassa, olivat JavaScriptin ohjelmointiin, testaamiseen ja vianetsintään tarkoitetut työkalut varsin harvinaisia ja puutteellisia. Java-ohjelmointiin sen sijaan työkaluja ja ohjelmointiympäristöjä löytyi lukuisia ja näistä lähtökohdista GWT sai alkunsa. Haluttiin tehdä väline, jolla

web-sovellusten kehitys olisi yhtä vaivatonta kuin perinteisten Java-sovellusten kehitys. (Tacy ym. 2013, 6.)

Vaikka GWT:llä onkin jo ikää, ei se ole jäänyt paikalleen web-tekniikkojen kehittyessä. Isoja päivityksiä julkaistaan säännöllisin väliajoin ja versiossa 2.5 merkittävimpiä uusia ominaisuuksia ovat HTML5-uutuudet, kuten tuki äänelle ja videolle. Myös mobiilituki on parantunut aikojen saatossa vastaamaan jatkuvasti kasvavan mobiilikäyttäjien joukon vaatimuksiin (Tacy ym. 2013, 4).

Ominaisuudet

Google Web Toolkit sisältää lukuisia hyödyllisiä komponentteja ja ominaisuuksia, jotka tekevät siitä monipuolisen työkalun web-kehittäjän avuksi. Merkittävin niistä lienee kuitenkin GWT:n ydin, eli Java-koodin JavaScriptiksi muuttava kääntäjä. Toinen tärkeä ominaisuus on GWT:n kehitystila (Development Mode), jonka avulla sovelluskehittäjä voi käyttää tavallisia Java-kehitystyökaluja (virheenetsintä, kutsupino, profilointi) sovellusta selaimessa testatessaan. Testausta helpottaa lisäksi GWT:stä löytyvä tuki suositulle Junit-yksikkötestausympäristölle. Myös tuki käytetyimmille kehitysympäristöille on auttanut GWT:n suosion kasvussa. Lisäosat löytyvät muun muassa Eclipseen ja NetBeansiin. (Wikimedia Foundation, hakupäivä 30.8.2013.)

GWT:llä voi luoda uudelleenkäytettäviä käyttöliittymäkomponentteja ja käyttöliittymien luontiin voi käyttää ohjelmakoodin kirjoittamisen sijaan XML-pohjaista UiBinder-työkalua, mikä taas parantaa sovelluksen ylläpidettävyyttä, kun ohjelmalogiikka on erillään ulkoasusta. UiBinderia käytettäessä sovelluksen käyttöliittymän voi tehdä ilman varsinaista ohjelmointiosaamista. Lisäksi GWT:stä löytyy valmiiksi ominaisuus sovelluksen lokalisointiin eri kielille ja kulttuureille. Ohjelmakoodin voi myös jakaa erikseen ladattaviin moduuleihin, jolloin selaimen ei tarvitse ladata koko ohjelmakoodia sivulatauksen yhteydessä, vaan vasta tarvittaessa. (Wikimedia Foundation, hakupäivä 30.8.2013.)

Tyypillinen ongelma JavaScript-sovelluksissa on selaimen Back-painikkeen toimimattomuus. Käyttäjät ovat tottuneet siihen, että Back-painiketta painamalla päästään edelliselle sivulle. JavaScript-sovelluksissa kaikki toiminnot ovat yleensä samalla sivulla, jossa ainoastaan sisältö vaihtuu käyttäjän toimintojen mukaan. GWT:n avulla kehittäjä voi kuitenkin määritellä, mitä tapahtuu käyttäjän painaessa Back- ja Forward-painikkeita. GWT:iin voi yhdistää myös tavallista Ja-

vaScript-koodia ja käyttää kolmannen osapuolen JavaScript-kirjastoja. GWT:llä ei sellaisenaan pysty tekemään erilaisia käyttöliittymiä mobiilialustoja ajatellen, mutta kolmannen osapuolen kirjastoja mobiiliversioiden tekoon löytyy lukuisia. (Wikimedia Foundation, hakupäivä 30.8.2013.)

Käyttöönotto

Kehittäminen Google Web Toolkitin avulla vaatii Java SDK:sta version 1.6 tai uudemman. Jos GWT:tä käyttää komentoriviltä, tulee ladata ja asentaa myös Apache Ant. Google Web Toolkit -sovelluskehitykseen tarvittavan GWT SDK:n saa ladattua osoitteesta http://www.gwtproject.org/download.html. Ladattu paketti puretaan haluttuun hakemistoon, minkä jälkeen asennushakemistosta löytyy webAppCreator-skripti, joka luo tarvittavat tiedostot ja hakemistorakenteen sovellukselle. Linux-käyttöjärjestelmässä sovelluksen runko luodaan seuraavalla komennolla:

```
webAppCreator -out MyApplication
com.mycompany.myapplication.MyApplication.
```

Out-parametri määrittää hakemiston, johon sovelluksen tiedostot ja hakemistot luodaan ja viimeinen parametri on luotavan moduulin nimi (täydellinen Java-luokan nimi). Kun hakemistorakenne on luotu, komentamalla `ant devmode` kyseisessä hakemistossa käynnistyy GWT:n paikalliseen kehitykseen ja virheenetsintään tarkoitettu palvelin. Kun auenneesta ikkunasta painaa ”Launch Default Browser”-painiketta, aukeaa sovellus selainikkunaan. Kun sovelluksen kehitysversion aukaisee selaimessa ensimmäistä kertaa, tulee selaimeen asentaa vielä GWT Developer Plugin -lisäosa selaimen ohjeiden mukaan. Asennuksen jälkeen sovellus on valmiina testattavaksi. (Getting Started, Google, hakupäivä 30.8.2013.)

4.4.2 XML-RPC

XML-RPC on alustariippumaton tapa tehdä etäkutsuja asiakasjärjestelmästä kohdejärjestelmään HTTP-protokollan välityksellä. XML-RPC sai alkunsa vuonna 1998, kun UserLand Software -yrityksen piti saada eri alustoilla toimivat sovelluksensa kommunikoimaan keskenään. Kyseisen yrityksen Dave Winer ratkaisi ongelman käyttämällä yleistä RPC (Remote Procedure Call, etä-

proseduurikutsu) -protokollaa, josta Winer jalosti yhdessä muutaman muun alan asiantuntijan kanssa myöhemmin XML-RPC -protokollan. (St.Laurent, Dumbill, Posner & Johnston 2001, 4.)

Käyttö

XML-RPC -kutsu lähtee siitä, kun asiakasohjelma tekee kutsun XML-RPC -asiakasta käyttäen kertoen sille kutsuttavan metodin nimen, metodin parametrit ja XML-RPC -palvelimen osoitteen. XML-RPC -asiakas muuttaa saamansa tiedot XML-muotoon ja tekee HTTP POST -pyynnön palvelimelle. Tämän jälkeen HTTP-palvelin vastaanottaa pyynnön ja välittää sen XML-RPC -kuuntelijalle, joka parsii XML:stä metodin nimen parametreineen ja tekee metodikutsun. XML-RPC -prosessi muuttaa metodilta saadun vastauksen XML-muotoon ja HTTP-palvelin palauttaa sen HTTP POST -pyynnön vastauksena XML-RPC -asiakkaalle, joka taas parsii saadun XML:n ja palauttaa vastauksen asiakasohjelmalle (St.Laurent ym. 2001, 16).

Yksi suosituimmista XML-RPC -kirjastoista PHP:lle on PEAR:n XML_RPC2. Vaikka edellinen versio on kirjoitushetkellä julkaistu yli kaksi vuotta sitten, on kyseinen kirjasto hyvä vaihtoehto XML-RPC:n käyttöön PHP-koodissa. Jos koneelta löytyy jo PEAR, onnistuu kirjaston asennus helposti komennolla `pear install XML_RPC2` (XML_RPC2, The PHP Group, hakupäivä 30.8.2013). Kun kirjasto on asennettu, yksinkertaisen XML-RPC -palvelimen luonti on helppoa:

```
<?php
require_once ”XML/RPC2/Server.php”;

class MyXmlRpcServer
{
  public static function hello($name)
  {
    return ”Hello, ” . $name . ”!”;
  }
}

XML_RPC2_Server::create(”MyXmlRpcServer”)->handleCall();
?>
```

Tällöin ylläolevalle palvelimelle tehty XML-RPC -pyyntö näyttäisi seuraavalta:

```
<?xml version="1.0"?>
<methodCall>
  <methodName>hello</methodName>
  <params>
    <param>
      <value>
        <string>Olli</string>
      </value>
    </param>
  </params>
</methodCall>
```

Ja palvelimelta saatu vastaus olisi seuraavanlainen:

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>Hello, Olli!</string>
      </value>
    </param>
  </params>
</methodResponse>
```

Tietotyypit

XML-RPC -kutsut muistuttavat tavallisten ohjelmointikielten funktiokutsuja. Kutsun mukana tulee paitsi suoritettavan metodin nimi, myös sille annettavat parametrit. Lisäksi metodi palauttaa vastauksena yhden arvon. Näitä arvoja varten protokollassa on määritelty taulukossa 1 esitelty joukko tietotyyppejä, joita XML-tieto voi sisältää. (St.Laurent ym. 2001, 17-18.)

TAULUKKO 1. XML-RPC:n tietotyypit (St.Laurent ym. 2001, 18-23)

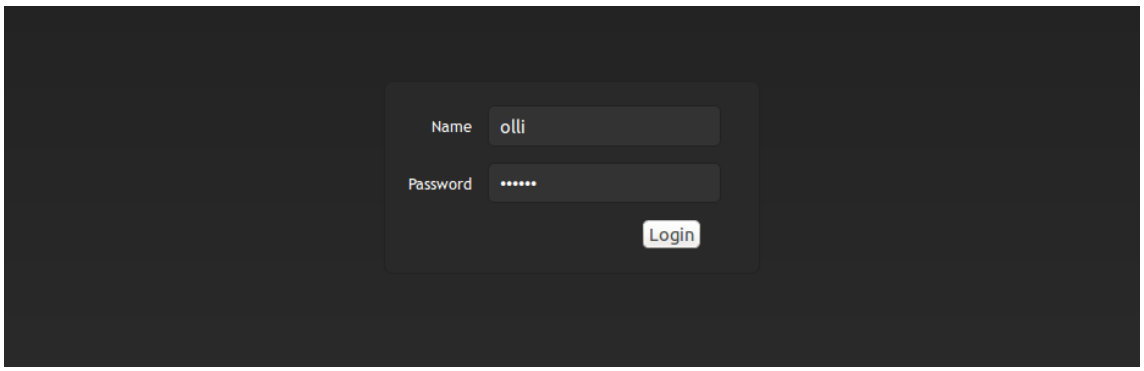
Tietotyyppi	Tyyppinimi	Esimerkkejä
Kokonaisluku	int (tai i4)	14, -78, +803
Liukuluku	double	2.5, -3.824
Totuusarvo	boolean	1, 0
Merkkijono	string	Hei, maailma!
Päivämäärä	dateTime.iso8601	20130830T12:54:31
Binääri	base64	SGVpLCBtYWFpbG1hIQ==
Taulukko	array	<data> <value>A</value> <value>B</value> </data>
Tietue	struct	<member> <name>Nimi</name> <value>Matti Meikäläinen</value> </member> <member> <name>Ikä</name> <value><int>57</int></value> </member>

5 TOTEUTETTU SOVELLUS

Opinnäytetyötä varten sovellukseen toteutettiin perustoiminnot, joita ovat elementtien luonti, poisto, muokkaus, linkitys ja käyttöoikeuksien hallinta. Näiden lisäksi sovellukseen tuli tehdä muutamia eri elementtityyppejä, kuten muistiinpano-, asiakas- ja yhteyshenkilötyypit sekä erikoiselementit koti, haku ja roskakori.

5.1 Kirjautuminen

Koska sovelluksen elementit sidotaan aina tiettyyn käyttäjään, esitetään sovelluksen pääsivulle ensimmäistä kertaa mentäessä käyttäjälle kuviossa 8 näkyvä kirjautumisruutu. Kirjautuminen tapahtuu syöttämällä kenttiin käyttäjätunnus ja salasana, joita sitten verrataan sovelluksen käyttäjätietokannasta löytyviin tietoihin. Opinnäytetyössä käyttäjätietojen sijainniksi voi valita joko tietokannan tai LDAP:n. Onnistuneen kirjautumisen jälkeen käyttäjälle luodaan palvelimella istunto, ja käyttäjälle näytetään oma kotielementti. Kotielementti vastaa lähinnä tietokoneen työpöytää, eli siihen voi luoda ja linkittää elementtejä, joiden halutaan olevan mahdollisimman nopeasti saatavilla käyttäjän kirjautuessa sovellukseen.



KUVIO 8. Sovelluksen kirjautumisruutu

5.2 Elementit

Sovellus koostuu elementeistä ja niiden keskinäisistä suhteista. Kullakin elementillä on omistaja, käyttöoikeuslista ja kyseisen elementin elementtityypin määrittämä sisältö. Kuhunkin elementtiin voi johtaa loputon määrä linkkejä muista elementeistä, ja kustakin elementistä voi myös olla rajaton määrä linkkejä muihin elementteihin. Myös elementin omistaja ja käyttöoikeuslista ovat omia elementtejään, ja molemmat on liitetty nimetyillä linkeillä elementtiin.

5.2.1 Elementtityypit

Sovellusta suunniteltaessa siitä haluttiin saada mahdollisimman modulaarinen ja laajennettava. Vaikka opinnäytetyötä varten sovelluksen elementtityypeiksi valittiin vain pieni joukko, on uusien elementtityyppien luonti suoraviivaista ja näin sovelluksesta on mahdollista tehdä sopiva minkälaiseen käyttöön tahansa.

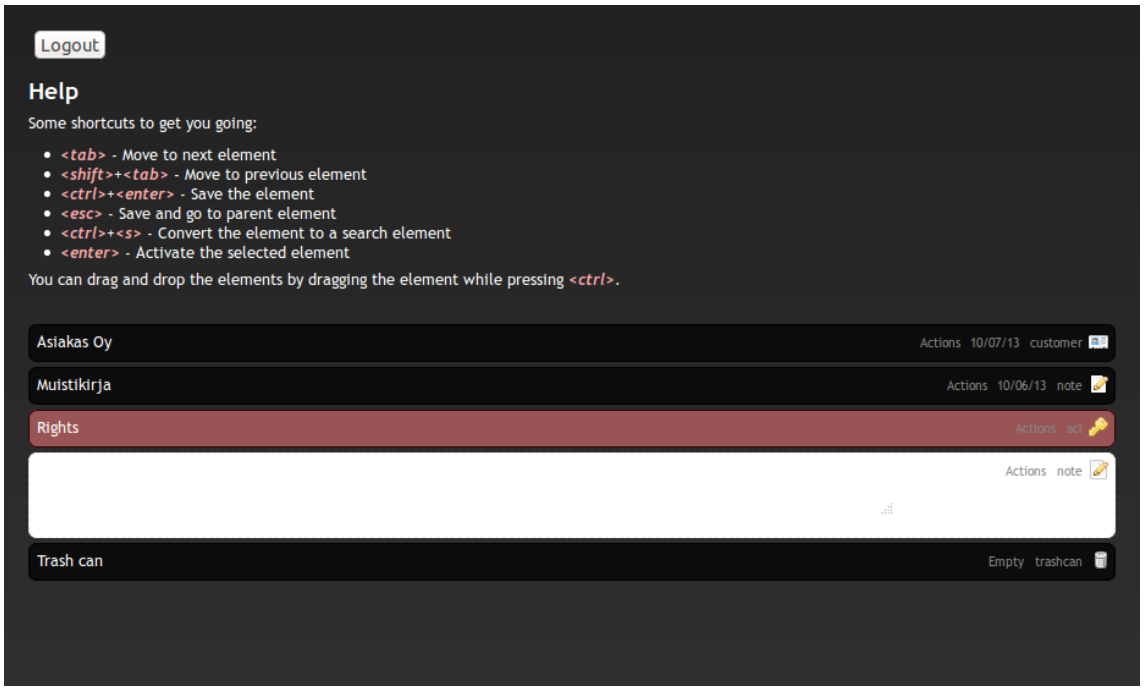
Tavalliset elementit

Tavallisilla elementeillä tarkoitetaan sovelluksessa elementtejä, joilla ei ole mitään erityistarkoitusta, vaan niiden tehtävänä on ainoastaan hallita kyseiseen elementtityyppiin liittyvää tietoa. Pelkistetyin elementtityyppi on muistiinpano, joka sisältää ainoastaan tekstiä. Kun käyttäjä luo uuden elementin, se on oletuksena muistiinpano-tyyppinen ja sen voi myöhemmässä vaiheessa muuttaa muun tyyppiseksi. Muita tavallisia elementtejä sovelluksessa ovat asiakas ja yhteystieto. Asiakkaalla on nimen lisäksi esimerkin vuoksi osoitetieto ja sähköpostiosoite. Yhteystiedolla on nimen lisäksi sähköpostiosoite.

Koti

Kuviossa 9 näkyvä kotielementti on elementti, joka liittyy aina tiettyyn käyttäjään ja joita voi olla ainoastaan yksi käyttäjää kohden. Se tehtiin sovellukseen, jotta elementtien organisointi käyttöliittymässä olisi selkeämpää. Jos käyttöliittymässä näytettäisiin oletuksena kirjautuneen käyttäjän oma käyttäjä-elementti, olisi näyttö täynnä käyttäjään linkitettyjä elementtejä ja käytöstä tulisi erittäin hankalaa. Nyt käyttäjän kirjautuessa sisään näytetään ruudulla ensimmäisenä käyttäjän kotielementti, johon käyttäjä voi ryhtyä luomaan uusia elementtejä. Kotielementti luodaan auto-

maattisesti käyttäjän kirjautuessa sisään ensimmäistä kertaa. Kuviossa 9 näkyy käyttäjän kotielementti, johon on luotu yksi asiakas (Asiakas Oy), sekä yksi muistiinpano (Muistikirja). Lisäksi kotielementtiin on linkitetty käyttöoikeuslista (Rights) ja roskakori (Trash can). Valkoisella pohjalla oleva elementti on tyhjä elementti, joka on aina näkyvissä uuden elementin nopeaa luomista varten.



KUVIO 9. Käyttäjän kotielementti

Roskakori

Roskakori on nimensä mukaan elementti, johon käyttäjä voi siirtää turhia elementtejä. Kun elementti siirretään roskakoriin, kaikki siihen johtavat linkit omistaja-linkkiä lukuun ottamatta poistetaan ja roskakorista luodaan linkki elementtiin. Kun roskakori tyhjennetään, poistetaan siellä olevat elementit lopullisesti.

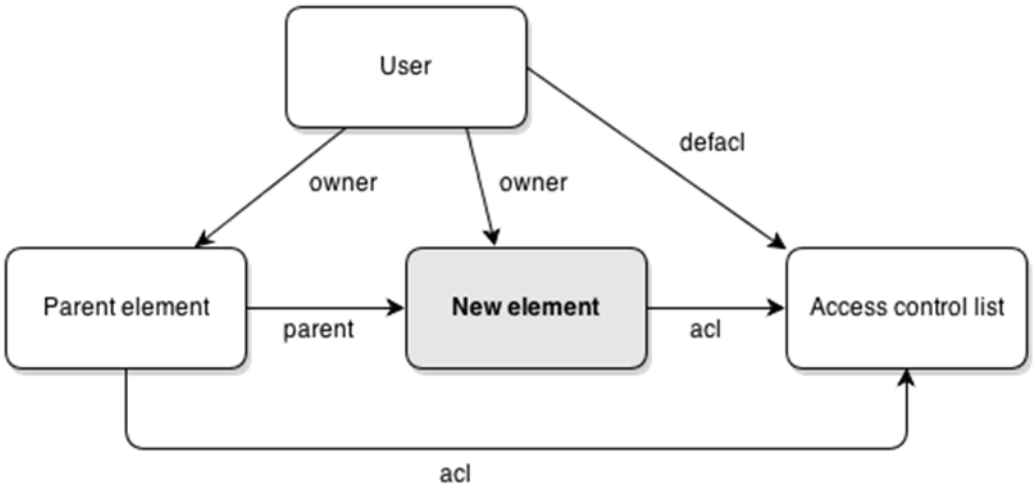
Hakuelementti

Hakuelementti on elementti, johon kirjoitetaan hakutermi, jonka perusteella elementtiin linkitetään kaikki hakuun täsmäävät elementit, joihin käyttäjällä on lukuoikeus. Haku suoritetaan elementin tallennuksen yhteydessä, joten mikäli uusia hakuun täsmääviä elementtejä on edellisen tallen-

nuksen jälkeen luotu, linkitetään ne hakuelementtiä tallennettaessa automaattisesti hakuelementtiin.

5.2.2 Luonti

Elementin luonti tapahtuu yksinkertaisesti siirtymällä tyhjään elementtiin ja täyttämällä elementin vaaditut kentät. Elementin tallennus tapahtuu joko manuaalisesti näppäinyhdistelmällä `control+enter` tai automaattisesti kun toinen elementti aktivoidaan. Kun elementti tallennetaan ensimmäisen kerran, luodaan käyttäjästä owner-tyyppinen linkki elementtiin omistussuhteen merkiksi. Koska jokaisella luotavalla elementillä on oltava isäelementti, luodaan elementin isäelementistä parent-tyyppinen linkki juuri luotuun elementtiin. Sovelluksessa jokaiselle elementille määritellään käyttöoikeudet, joten elementtiin linkitetään myös yksi käyttöoikeuselementti, jonka arvot määräytyvät käyttäjän valitsemien oletusarvojen mukaisesti.

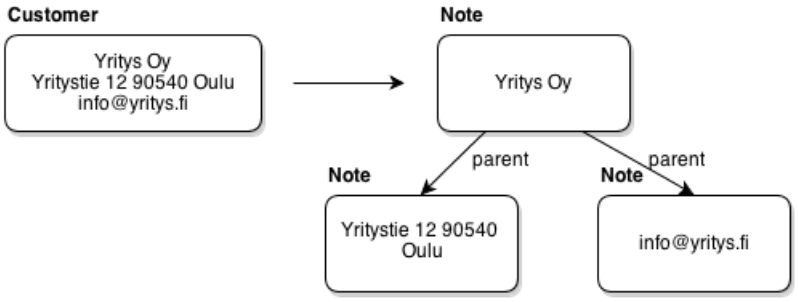


KUVIO 10. Elementin linkit luonnin jälkeen

Kuviossa 10 on luotu uusi elementti (New element) käyttäjän User toimesta toisen elementin (Parent element) alle. Käyttäjästä on tehty uuden elementin omistaja luomalla owner-linkki ja uuteen elementtiin johtaa myös parent-linkki sen isäelementistä. Lisäksi uuteen elementtiin on linkitetty käyttäjän oletusarvoinen käyttöoikeuslista-elementti. Oletusarvoisuus on määritelty defacl (Default access control list) -nimisellä linkillä.

5.2.3 Tyypin muuttaminen

Elementin tyypin muuttaminen tapahtuu valitsemalla elementin apuvalikosta uusi tyyppi. Jos alkuperäisellä elementtityypillä on samoja kenttiä uuden tyypin kanssa, muuttuvat vanhat arvot suoraan uuden tyypin vastaaviksi arvoiksi. Mikäli alkuperäisellä elementtityypillä on kenttiä, jotka uudelta tyypiltä puuttuvat, luodaan kustakin kentästä muistiinpano-tyyppinen uusi elementti, jotka linkitetään elementin alle. Hakuelementiksi muuttaminen poikkeaa muista tyyppimuunnoksista siten, että sen voi tehdä myös nopeasti näppäinyhdistelmällä `control+s`. Kuviossa 11 on esimerkki asiakas-elementin (Customer) muuttamisesta muistiinpano-elementiksi (Note). Asiakas-elementillä on kentät nimi, osoite ja sähköpostiosoite ja muistiinpanolla ainoastaan kenttä nimi. Tällöin asiakkaan nimi menee uuden tyypin nimeksi ja loput kentät, joita muistiinpanosta ei löydy, linkitetään muistiinpano-elementteinä tyyppimuunnetun elementin alle.



KUVIO 11. Asiakas-elementin muuttaminen muistiinpanoksi

5.2.4 Poistaminen

Elementin voi poistaa siirtämällä sen roskakoriin. Tämä ei kuitenkaan poista elementtiä täysin, vaan ainoastaan tuhoaa kaikki elementtiin aikaisemmin johtavat linkit omistaja-linkkiä lukuun ottamatta ja linkittää sen roskakoriin. Roskakorin voi tyhjentää roskakori-elementistä löytyvän toiminnon avulla, jolloin kaikki roskakorin elementit yritetään poistaa, mikäli käyttöoikeudet sallivat. Kun lopulta käy niin, ettei elementtiin johda yhtään linkkiä, se jää orvoksi ja se tällöin poistetaan automaattisesti.

5.2.5 Linkitys

Elementin linkitys toiseen elementtiin tapahtuu normaalisti raahaamalla elementti toisen elementin linkkialueelle. Elementin pudottamisen jälkeen voi käyttäjä vielä valita, kopioidaanko elementti vai siirretäänkö elementti. Kopioinnissa kohde-elementistä luodaan linkki raahattuun elementtiin muiden linkkien pysyessä ennallaan. Siirrossa taas häntäelementin linkki poistuu ja korvautuu kohde-elementin linkillä. Jotta elementin linkitys olisi mahdollista, tulee käyttäjällä olla kirjoitusoikeus kohde-elementtiin ja lukuoikeus linkitettävään elementtiin.

5.3 Käyttöoikeuksien hallinta

Sovelluksen tärkeimpiä ominaisuuksia on kattava käyttöoikeuksien hallinta. Sovellus koostuu tavallisten elementtien lisäksi käyttäjä- ja käyttäjäryhmäelementeistä. Käyttäjäryhmä sisältää käyttäjiä ja käyttäjät voivat taas luoda omia käyttäjäryhmiä. Sovelluksessa on myös erityinen käyttäjäryhmä Everybody, johon kaikki käyttäjät kuuluvat. Käyttäjien ja käyttäjäryhmien lisäksi käyttöoikeuksien hallintaan sovelluksessa tarvitaan käyttöoikeuselementit (access control) ja käyttöoikeuslistat (ACL, access control list). Käyttöoikeuselementillä määritetään, mille käyttäjäryhmälle tai käyttäjälle sallitaan tai evätään luku- tai kirjoitusoikeus. Yhteen käyttöoikeuslistaan voi linkittää useita käyttöoikeuselementtejä, jolloin listalla voidaan monipuolisesti määritellä useita käyttöoikeussääntöjä samanaikaisesti.

Käyttöoikeussääntöjä on neljä erilaista. Grantread sallii käyttäjän tai käyttäjäryhmän lukea elementin tietoja, kun taas grantwrite antaa käyttäjälle tai käyttäjäryhmälle kirjoitusoikeudet kyseiseen elementtiin. Kirjoitusoikeus mahdollistaa myös elementin linkkien muokkaamisen, lisäyksen ja poiston lukuun ottamatta elementin omistaja- ja käyttöoikeuslista-linkkejä. Näiden kahden sallivan säännön lisäksi on myös säännöt denyread ja denywrite, joista ensimmäinen evää lukuoikeuden ja jälkimmäinen kirjoitusoikeuden.

Käyttöoikeuslistaa voi muokata ainoastaan sen omistava käyttäjä. Teoriassa olisi mahdollista luoda käyttöoikeuslista käyttöoikeuslistalle, mutta se tekisi sovelluksen käytöstä turhan monimutkaista. Tästä syystä käyttöoikeuslistaa voi kuka tahansa lukea, mutta ainoastaan listan omistaja voi sitä muokata. Elementin käyttöoikeuksia tarkastellessa ensin katsotaan mahdolliset kiellot ja vasta sen jälkeen sallivat säännöt. Jos esimerkiksi tietylle käyttäjäryhmälle on annettu lukuoikeu-

det elementtiin, mutta yhdeltä ryhmän jäsenistä luku on evätty, ei kyseinen jäsen voi nähdä elementtiä. Toisaalta taas käyttäjäkohtaisilla säännöillä voidaan ohittaa ryhmäkohtaisten sääntöjen määritykset. Jos käyttäjäryhmä ei voi lukea elementtiä, mutta yhdelle ryhmän käyttäjistä on luku sallittu, voi kyseinen käyttäjä lukea elementin tietoja. Käyttäjän oikeuksia tiettyyn elementtiin tutkittaessa käydään seuraavat tarkistukset järjestyksessä läpi:

1. Onko käyttäjä kirjautuneena? Jos ei, estä pääsy.
2. Onko käyttäjä elementin omistaja? Jos on, salli pääsy.
3. Onko elementillä käyttöoikeuslista? Jos ei, estä pääsy.
4. Estääkö käyttöoikeuslista tämän käyttäjän pääsyn? Jos estää, estä pääsy.
5. Salliiko käyttöoikeuslista käyttäjän pääsyn? Jos sallii, salli pääsy.
6. Estääkö käyttöoikeuslista pääsyn ryhmältä, johon tämä käyttäjä kuuluu? Jos estää, estä pääsy.
7. Salliiko käyttöoikeuslista pääsyn ryhmältä, johon tämä käyttäjä kuuluu? Jos sallii, salli pääsy. (Mäenpää 2007, 12-13.)

Kullekin käyttäjälle luodaan ensimmäisen kirjautumisen yhteydessä oletusarvoinen käyttöoikeuslista, joka sallii Everybody-käyttäjäryhmälle lukuoikeudet. Oletusarvoinen käyttöoikeuslista on liitetty defacl-nimisellä linkillä käyttäjään. Kun käyttäjä luo uuden elementin, linkitetään oletusarvoinen käyttöoikeuslista uuteen elementtiin määrittäen näin ollen automaattisesti uuden elementin käyttöoikeudet. Käyttöoikeuslistan linkitys tapahtuu acl-nimisellä linkillä. Käyttäjä voi halutessaan muokata oletusarvoista käyttöoikeuslistaansa täysin vapaasti.

6 YHTEENVETO JA POHDINTA

Opinnäytetyön tavoitteena oli luoda toimeksiantajalle sovellus, jolla yritys voisi aluksi hallita asiakkaitaan ja kontaktejaan sekä luoda yksinkertaisia muistiinpanoja. Lisäksi sovelluksesta haluttiin mahdollisimman helposti laajennettava, jotta sitä voitaisiin myöhemmin jatkokehittää lisäämällä tarvittavia ominaisuuksia. Toinen toimeksiantajalle tärkeä tavoite oli, että sovelluksessa käytetty kehityskirjasto Google Web Toolkit tulisi testattua oikeassa kehitystyössä ja samalla puntaroitua, olisiko siitä mahdollisesti hyötyä myöhemmissäkin projekteissa. Olihan opinnäytetyön tekijä vielä tuossa vaiheessa toimeksiantajan työntekijänä. Näissä tavoitteissa mielestäni onnistuttiin varsin hyvin. Vaikka sovelluksesta ei saatukaan niin modulaarista kuin alun perin oli suunnitelmissa, on lisäosien tekeminen kuitenkin varsin suoraviivaista.

Varsinaiseen toteutusvaiheeseen meni moninkertaisesti tavanomaisen opinnäytetyön laajuuteen verrattuna aikaa, sillä sovelluksen käyttöoikeusjärjestelmästä haluttiin tehdä monipuolinen ja täysin varmatoiminen. Lisäksi sovelluksen käyttöliittymän toteutuksessa käytetty Google Web Toolkit –kirjasto oli tuolloin vielä täysin uusi paitsi itselleni, myös kaikille muille sen julkaisun tapahduttua samana vuonna sovelluksen toteutustyön alkaessa. Dokumentaatiota oli kyllä tarjolla, mutta se oli osin puutteellista ja lisäksi kirjasto sisälsi vielä suhteellisen paljon lastentauteja. Kehitystyön edetessä kirjasto tuli kuitenkin tutuksi ja näin jälkeenpäin olen voinut huomata GWT:n olevan varteenotettava vaihtoehto dynaamisia verkkosovelluksia kehitettäessä ja olen käyttänyt sitä myöhemmissäkin projekteissa menestyksellä.

Google Web Toolkit sisältää valmiita käyttöliittymäkomponentteja, jotka ainakin kirjaston elämänkaaren alkuvaiheessa oli toteutettu suurimmaksi osaksi käyttäen HTML-merkintäkielen taulukkoelementtiä. Taulukkoa käytettäessä sisältö on helppo asetella riveihin ja sarakkeisiin, mutta ongelmaksi tuli Internet Explorer –selainten rajoitus, jossa sivulla voi olla sisäkkäisiä taulukoita vain rajallinen määrä. Toteutetussa sovelluksessa elementtejä (kuten muistiinpanoja) piti kuitenkin pystyä asettelemaan sisäkkäin lähes rajaton määrä, joten GWT:n omia käyttöliittymäkomponentteja ei voitu tämän ongelman vuoksi käyttää elementtien asettelussa. GWT:tä käytettäessä suositellaan viimeiseen asti käytettävän sen omia käyttöliittymäkomponentteja, sillä ne on testattu huolellisesti ja tehty niin, että selainyhteensopivuusongelmia ei pitäisi esiintyä. Omia komponent-

teja voi tehdä, mutta toteutusvaiheen aikaan se ei ollut kovinkaan suoraviivaista eikä erityisen helppoa ja vaati turhan paljon ylimääräistä aikaa paitsi toteutuksen myös testauksen osalta.

XML-RPC osoittautui valmiita kirjastoja käytettäessä helpoksi tavaksi välittää tietoa käyttöliittymän ja palvelinsovelluksen välillä. Näin jälkikäteen ajateltuna JSON-formaatti (JavaScript Object Notation) olisi ollut kuitenkin parempi ratkaisu, sillä siirrettävän tiedon määrä olisi vähentynyt olennaisesti ja lisäksi sovelluksen testaus olisi helpottunut, sillä selainten kehitystyökaluista löytyy nykyään oletuksena ominaisuudet JSON-objektien tarkastelulle. XML:lle ei vastaavia ominaisuuksia löydy.

Sovelluksen käyttöliittymän testaus piti tehdä lähes kokonaan manuaalisesti, sillä tarkoitukseen sopivia järjestelmiä (kuten tätä kirjoitettaessa Behat tai Selenium) ei tuolloin ollut olemassa tai sitten olivat niin varhaisessa kehitysvaiheessa, ettei niistä ollut niin kattavaan testaukseen kuin opinnäytetyön sovellus olisi vaatinut. Manuaalinen testaus oli erittäin työlästä johtuen sovelluksen laajuudesta sekä monipuolisesta ja kattavasta käyttäjä- ja käyttöoikeusjärjestelmästä. Testauksen tueksi pitikin kehittää erilaisten visualisointien ja kuvioiden tekoon tarkoitettua sigma.js -JavaScript-kirjastoa käyttäen web-sivu, josta elementtien suhteet voidaan nähdä yhdellä silmäyksellä. Tämä oli tarpeellinen erityisesti elementtien kopiointia, siirtoa ja käyttöoikeuksien muutoksia testatessa. Ajan puutteen vuoksi myös yksikkötestaus jäi vajavaiseksi.

Vaikka alunperin sovellusta suunniteltiin toimeksiantaja Intopii Oy:n omaan käyttöön, huomattiin kehitystyön aikana, että siitä saa pienillä muutoksilla sopivan työkalun moniin muihinkin käyttötarkoituksiin. Ilman varsinaista ohjelmointia jo pelkillä muistiinpanoilla pääsee jo varsin pitkälle, varsinkin kun niitä voi jakaa eri käyttäjien kesken. Sovellus toimii mainiosti esimerkiksi perheen kauppalistana tai vaikkapa henkilökohtaisena tehtävälistana. Koska opinnäytetyön laajuudessa sovellukseen tehtiin kontaktien hallinta, voi sovelluksessa säilyttää myös omia henkilökohtaisia yhteystietoja. Jos pelkät muistiinpanot eivät riitä sujuvan käytön takaamiseksi, voi pienellä kehitystyöllä sovellukseen tehdä kauppalistakäyttöä ajatellen uusia elementtityyppejä. Sopivia tyyppejä voisivat olla ”kauppalista”- ja ”tuote”-elementtityypit. Tavallista muistiinpanotyyppiä uusi kauppalistatyyppi voisi laajentaa lisäämällä esimerkiksi kentän ”kokonaishinta” ja ”tuote”-tyyppi taas voisi sisältää kentät ”hinta”, ”lukumäärä” ja ”lisätty koriin”. Kun tällaiseen kauppalistaan annetaan käyttöoikeudet kaikille perheen ruokaostoksista vastaaville, synkronoituvat toisen tekemät muutokset automaattisesti muiden tietoon ja lista on aina kaikilla sen käyttäjillä ajan tasalla.

LÄHTEET

Anttila, J. 2001. Dokumenttien hallinta. Helsinki: Edita.

Bozidar L. 2002. UNIX Administration: A Comprehensive Sourcebook for Effective Systems & Network Management. Boca Raton, FL: CRC Press.

Getting Started, Google. 2013. Hakupäivä 30.8.2013, http://www.gwtproject.org/gettingstarted.html.

Google Web Toolkit, Wikimedia Foundation. 2013. Hakupäivä 30.8.2013, http://en.wikipedia.org/wiki/Google_Web_Toolkit.

Hirvelä, H. 2013. Konenäkö on väsymätön ja tarkka. Hakupäivä 3.11.2013, http://www.ilkka.fi/mielipide/yleisöltä/konenako-on-vasymaton-ja-tarkka-1.1482809.

Laszlo Systems. 2006. Technology White Paper. Hakupäivä 28.8.2013, http://www.openlaszlo.org/whitepaper/LaszloWhitePaper.pdf.

Mäenpää, T. 2007. Jotos - Technical Specification. (Ei julkaisupaikkaa eikä julkaisijaa).

Nixon, R. 2009. Learning PHP, MySQL and JavaScript. Sebastopol, CA: O'Reilly.

Silberschatz A., Korth H. & Sudarshan S. 2001. Database System Concepts, Sixth edition - Appendix D, Network Model. Hakupäivä 10.9.2013, http://codex.cs.yale.edu/avi/db-book/db6/appendices-dir/d.pdf.

St.Laurent, S., Dumbill E., Posner J. & Johnston J. 2001. Programming Web Services with XML-RPC. Sebastopol, CA: O'Reilly.

Tacy A., Hanson R., Essington J. & Tökke A. 2013. GWT in Action, Second Edition. Shelter Island, NY: Manning.

Weaver, J.L., Weiqi, G., Chin, S., Iverson D. & Vos J. 2012. Pro JavaFX 2. New York, NY: Apress.

XML_RPC2, The PHP Group. 2013. Hakupäivä 30.8.2013, http://pear.php.net/package/XML_RPC2.