HAAGA-HELIA
ammattikorkeakoulu

Mobiilioptimoitu käyttöliittymä

Ossi Hirvikoski, Patrik Vilja

Opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

2013

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

7.1.2013

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Ossi Hirvikoski, Patrik Vilja	**Ryhmätunnus tai aloitusvuosi** 2008 & 2009
Opinnäytetyön nimi Mobiilioptimoitu käyttöliittymä	**Sivu- ja liitesivumäärä** 84 + 68
Opettajat tai ohjaajat Juhani Välimäki	

Tässä opinnäytetyössä toteutettiin vuoden 2012 syksyn aikana mobiilioptimoidun käyttöliittymän prototyyppi, joka toimii vaihtoehtoisena käyttöliittymänä IMS-toimintajärjestelmän käyttämiseksi. Opinnäytetyö toteutettiin tilaustyönä IMS Business Solutions Oy:lle. Mobiilioptimoitu käyttöliittymä luotiin hyödyntäen uusimpia web-kehitysmenetelmiä. Opinnäytetyö on suunnattu sovelluskehittäjille ja sen ymmärtäminen vaatii osittain tietoteknistä osaamista ja tuntemusta.

Käyttöliittymäsuunnittelu, -toteutus ja -testaus vaativat paljon tutkimustyötä ja osaamista. Tässä dokumentissa kuvataan kuinka projektiryhmä suoriutui tästä monivaiheisesta toteutuksesta. Opinnäytetyössä keskityttiin prototyypin ja sen käyttöliittymän luomiseksi käytettyihin ratkaisuihin ja valittuihin teknologioihin. Käyttöliittymän suunnittelua varten toteutettiin kysely yrityksen asiakkaille. Tämän ohessa suunniteltiin alustava ulkoasu ja toiminnallisuus sekä mietittiin teknologisia ratkaisuja niiden toteuttamiseksi. Sovelluksessa pyrittiin käyttämään uusia, hyväksi todettuja teknologioita, joiden ansiosta se pysyy pitkään kilpailukykyisenä. Tämän vuoksi opinnäytetyössä tutustuttiin useisiin viime vuosina kehitettyihin teknologioihin ja teknologiatrendeihin, kuten JavaScript MVC -kehyksiin, CSS-esikäsittelyyn, REST-arkkitehtuurimalliin ja mukautuvaan websuunnitteluun (Responsive Web Design).

Käyttöliittymän käytettävyyden parantamiseksi tehtiin käytettävyystestausta. Testauksen perusteella käyttöliittymästä löydettiin joitakin oleellisia ongelmia, mutta kaiken kaikkiaan käyttöliittymä osoittautui olevan helposti käytettävä ja ymmärrettävä. Hyväksymistesteillä todistettiin prototyypin sisältävän ne toiminnallisuudet, jotka siihen oli luvattu toteuttaa. Tuloksena syntyi odotuksia vastaava prototyyppi mobiilioptimoidusta käyttöliittymästä.

Asiasanat
Käyttöliittymät, käytettävyys, mobiililaitteet, ohjelmistokehitys, esikäsittely, testaus

HAAGA-HELIA
University of Applied Sciences

Abstract

7th of January, 2013

Degree programme in Information Technology

Authors Ossi Hirvikoski, Patrik Vilja	**Group or year of entry** 2008 & 2009
The title of thesis User interface optimized for mobile devices	**Number of pages and appendices** 84 + 68
Supervisor(s) Juhani Välimäki	

The purpose of this thesis was to create a prototype of a mobile optimized user interface in the fall of 2012. The user interface functions as an alternative for using the IMS Integrated Management System. The thesis work was commissioned by IMS Business Solutions Oy. The user interface was created using the latest web development methods. The thesis is intended for software developers, and you must have some knowledge of these methods to appreciate its contents.

Designing, developing and testing the mobile optimized user interface required a lot of research and know-how. This document describes how the project group accomplished these objectives. The thesis focuses on the technologies and solutions used to create the prototype and its user interface. To assist the design of the user interface, a questionnaire addressed to company's customers was realized. At the same time, the project group started designing the preliminary appearance and the functions of the interface, and studied technological solutions to implement them. The objective was to use the latest and "best-practice" technologies that help to keep the prototype competitive in its field. For this the project group studied several latest technologies and technological trends, such as JavaScript MVC frameworks, CSS preprocessing, REST architecture model and responsive web design.

Usability testing was carried out to improve the usability of the mobile optimized user interface. The test results revealed that there are some essential usability problems in the user interface, but also that the interface is easy to use and to understand. Acceptance tests proved that the prototype contained the functions that had been promised to be implemented in the requirement specification documents. The result of the thesis was a prototype of a mobile optimized user interface that contains all the expected features and functions.

Key words
User interfaces, usability, mobile devices, software development, preprocessing, testing

Sisällys

Sanasto

Sanasto sisältää opinnäytetyössä esiintyvät keskeiset käsitteet aakkosjärjestyksessä. Opinnäytetyössä käytetään monia menetelmiä ja teknologioita, jotka saattavat olla aihealueensa tai uutuusarvonsa vuoksi tuntemattomia lukijalle, vaikka tämä olisikin perehtynyt tietotekniikkaan tai johonkin sen osa-alueeseen. Samoin esitetään termejä, jotka ovat ominaisia IMS-toimintajärjestelmälle ja siten mahdollisesti tuntemattomia lukijalle.

Ajax Tekniikka, jonka avulla voidaan asynkronisesti lähettää ja vastaanottaa resursseja palvelimelta ilman, että koko sivua tarvitsee päivittää.

AngularJS Googlen kehittämä JavaScript MV* -kehys.

Behaviour Driven Development (BDD)
Sovelluskehitysmalli, joka pyrkii luomaan IT-ratkaisut sekä liiketaloudellisesta että tietoteknisestä näkökulmasta.

Bring Your Own Device (BYOD)
Periaate, jossa työnantajat antavat työntekijöidensä tuoda ja käyttää omia laitteitaan työn tekemiseen.

CoffeeScript JavaScript-kielelle kehitetty esikäsittelykieli.

Compass Sass-esikäsittelykielelle luotu ohjelmistokehys, joka tarjoaa valmiita mixin-laajennuksia etenkin CSS3-ominaisuuksia varten. Kehyksen avulla vältytään toistamasta selainkohtaisia etuliitteitä CSS3-ominaisuuksissa.

Cucumber-testaus Behaviour Driven Development (BDD) -malliin perustuva testaamisympäristö, jossa kuvataan ensiksi selkokieliset

käyttötapaukset, sitten yksittäiset askeleet ja lopuksi luodaan koneelliset käskyt toteuttamaan edellä mainitut askeleet. Lopputuloksena saadaan automatisoitu testausympäristö, jonka askelia on myös mahdollista lukea selkokielisenä.

Dependency Injection (DI)

Arkkitehtoninen malli, jolla liitetään sovelluskoodin riippuvuudet toisiinsa. Yleensä hyödynnetään IoC-säiliön kanssa.

Java Persistence API (JPA)

Standardi tietokantarajapinta, jota tietokantakehykset, kuten Hibernate, toteuttavat.

JavaScript MV*

JavaScript-rakenteen hallinnointiin luotuja ohjelmistokehyksiä. Sisältävät hyväksi todettuja arkkitehtonisia ratkaisuja, joiden avulla ne pyrkivät auttamaan interaktiivisten SPA-sovelluksien luomiseksi.

JavaScript Object Notation (JSON)

JavaScript-pohjainen formaatti, jota käytetään esitys-, tallennus- ja tiedonsiirtomuotona.

Katsojakäyttäjä

IMS-toimintajärjestelmän käyttäjä, jolle on pääasiassa annettu ainoastaan oikeudet katsoa ja tarkastella järjestelmän sisältöä. Mobiilioptimoidun käyttöliittymän pääkohderyhmä.

Kohde

Paikka käyttöliittymässä, esimerkiksi dokumentti tai taso.

Kolmitasoarkkitehtuuri

Arkkitehtoninen malli, jossa sovelluskoodin toteutus jaetaan modulaarisiin, väljästi kytkettyihin kerroksiin.

IMS-toimintajärjestelmä

IMS Business Solutions Oy:n tarjoama ratkaisu yrityksille laadunhallinnan todistamiseen ja ylläpitämiseen.

Inversion of Control (IoC)

Tekniikka, jossa sovelluskoodin riippuvuudet ovat yhteisessä säiliössä. Tekniikalla helpotetaan viittaamista eri riippuvuuksien välillä ja saavutetaan riippuvuuksien itsenäisyys.

Mixin-laajennukset Luokka, joka määrittelee joukon toimintoja, jotka liittyvät tiettyyn tyyppiin. Mixin-luokan toiminnot kopioidaan ja lainataan sitä hyödyntäville luokille, kuitenkaan muodostamatta luokkien välille normaalia periytymisyhteyttä. Täten voidaan toteuttaa luokkien välille moniperintä.

Mobiilioptimoitu käyttöliittymä (Mobile IMS)

Vaihtoehtoinen käyttöliittymä, jolla voidaan käyttää ja tarkistella IMS-toimintajärjestelmän sisältöä; optimoitu mobiililaitteille ja toiminnallisuuksiltaan suunnattu ensisijaisesti katsojakäyttäjille.

Model, View, Controller (MVC)

Ohjelmistoarkkitehtuuri, jossa sovellus jaetaan kolmeen itsenäiseen osaan: malliin (model), näkymään (view) ja ohjaimeen (controller). Mallissa kuvataan tiedon käsittely, ylläpito ja tallennus, näkymässä määritetään käyttöliittymän ulkoasu, ja ohjaimessa vastaanotetaan käyttäjän käskyt.

Mukautuva websuunnittelu (Responsive Web Design)

Selainsovelluksen tai sivun suunnittelumalli, jonka avulla luodun sovelluksen tai sivun sisältö saadaan riippumattomaksi näytön koosta.

Nykyinen käyttöliittymä

Käyttöliittymä, jolla perinteisesti tarkastellaan ja muokataan IMS-toimintajärjestelmän sisältöä.

Service Oriented Architecture (SOA)

Palvelukeskeinen arkkitehtuuri; jaetaan useampaan itsenäiseen palveluun, jotka kommunikoivat muiden palveluiden kanssa. Käsite on abstrakti, eikä se sisällä toteutusta. Esimerkiksi REST toteuttaa SOA-pohjaisen arkkitehtuurin.

Single-page Application (SPA)

Sovelluskehitysmalli, jossa sovellus sisältää ainoastaan yhden sivun, joka toimii pohjana sovelluksen näkymille. Sovelluksen navigointi toteutetaan asynkronisesti Ajax-tekniikan avulla ja sovelluksessa ei esiinny koko sivun päivityksiä.

Spring Framework Javalle kehitetty ohjelmistokehys, joka pyrkii auttamaan sovelluskehitystä yleisten hyväksi todettujen arkkitehtonisten mallien avulla.

Syntactically Awesome Stylesheets (Sass)

CSS-tyylikielelle kehitetty esikäsittelykieli. Tuo tyyliohjeiden kirjoittamiseen funktionalisuutta muuttujien, sisäkkäisen syntaksin ja mixin-laajennuksien avulla. Sass-esikäsittelykieli kääntyy lopuksi perinteiseksi CSS-tyylikieleksi.

REST Arkkitehtoninen malli SOA-pohjaisten palveluiden toteuttamiselle. Mahdollistaa palvelujen välisen integraation HTTP-pohjaisten rajapintojen avulla.

RESTful Kuvaa tilannetta, jossa käytetään REST-arkkitehtonista mallia.

Yeoman	Työkalu, joka tarjoaa apua modernin käyttäjäpuolen arkkitehtuurin käyttöönottoon.
Taso	Kansio eli kohde, jossa on sisältöä. Käytetään IMS-toimintajärjestelmässä termin ”kansio” sijasta.
WebSockets	Teknologia, joka mahdollistaa kaksisuuntaisen kommunikoinnin TCP-yhteyden kautta. Sen avulla palvelinpuolen toteutus voi lähettää tietoa valmiiksi muodostetulle HTML-sivulle ilman, että sivulta pyydetään kyseistä tietoa (Push).

1 Johdanto

Mobiilioptimoitu käyttöliittymä -projekti tehdään HAAGA-HELIA Ammattikorkeakoulun tietojenkäsittelyn koulutusohjelman opinnäytetyönä. Työ toteutetaan toimeksiantona IMS Business Solutions Oy:lle, jossa on ilmennyt tarve tarjota yrityksen toimintajärjestelmäratkaisu mobiilikäyttöön. Mobiililaitteet ovat kasvattaneet suosiotaan jo useamman vuoden ajan, mutta yrityksen tuotekehitysosaston pienen koon vuoksi järjestelmän mobiilikäyttöön ei ole pystytty perehtymään aiemmin.

IMS-toimintajärjestelmä on IMS Business Solutions Oy:n tarjoama ratkaisu yrityksille laadunhallinnan todistamiseen ja ylläpitämiseen. Toimintajärjestelmää käytetään selainpohjaisella käyttöliittymällä. Järjestelmän nykyinen käyttöliittymä on mahdollista avata mobiililaitteiden selaimilla, mutta sen käyttö mobiiliselaimella on hankalaa ja raskasta, eivätkä läheskään kaikki toiminnallisuudet tällöin toimi. Nykyisen käyttöliittymän muokkaaminen mobiiliystävälliseksi nähdään liian monimutkaisena. Käyttöliittymässä on myös käytössä monia vanhoja tekniikoita, jotka eivät sovellu mobiilikäyttöön.

Opinnäytetyöprojektissa sovelletaan ketteriä menetelmiä, joista tärkein on Scrum. Scrum-menetelmän etuna on läheinen yhteys sovelluksen tilaajaan, jolloin lopputuloksena saadaan todennäköisesti tilaajaa miellyttävä tuote. Menetelmään kuuluu myös projektissa suoritettavien töiden jakaminen osiin, osien työmäärän arviointi ja toteuttaminen iteraatioissa, mikä edesauttaa projektin tilanteen ja etenemisen arviointia. Projektiryhmällä on eniten kokemusta Scrum-menetelmän mukaisesta työskentelystä, joten menetelmän käyttö takaa vahvan pohjan projektinhallinnalle.

Projektissa luotavalla prototyypillä on kaksi päätarkoitusta: ensiksikin tarjota käyttäjille mahdollisuus käyttää IMS-toimintajärjestelmän sisältöä mobiililaitteilla ja toiseksi tarjota katsojakäyttäjille mukava käyttöliittymä toimintajärjestelmän käyttämiseksi internetselainta tukevalla laitteella. Lisäksi prototyypin toteutuksessa tutkitaan uusia arkkitehtonisia malleja ja tekniikoita, joista osa otetaan käyttöön yrityksen sovelluskehitykseen jo projektin aikana.

Mobiilioptimointi toteutetaan selainsovelluksena mukautuvan websuunnitelun menetelmää hyödyntäen. Tällainen selainsovellus saadaan toimimaan suoraan nykyisen järjestelmän oheen sekä tietokoneilla että kaikilla selaimia tukevilla mobiililaitteilla. Ratkaisun ansiosta projektiryhmän ei tarvitse tutkia, mille mobiilialustalle sovellusta tulisi lähteä tekemään, vaan projektissa pystytään keskittymään suoraan sovelluksen ulkoasuun, käyttöliittymään ja käytettävyyteen.

Selainsovelluksen luomista pohjustavina töinä projektissa toteutetaan käyttäjäkysely, pohditaan sovellukselle tärkeitä toiminnallisuuksia ja mietitään sovelluksen arkkitehtonisia ja tietoteknisiä ratkaisuja. Käyttäjäkyselyllä pyritään saamaan selville toimintajärjestelmän käyttäjien eniten käyttämiä ja arvostamia ominaisuuksia sekä toiveita uudelle käyttöliittymälle. Tulosten perusteella päätetään, minkälaisia alustavia ominaisuuksia mobiilioptimoidun käyttöliittymän tulisi sisältää.

Sovelluksen ulkoasua, käytettävyyttä ja toimintoja kartoitetaan myös käytettävyystestauksen avulla opinnäytetyön aikana. Pääpainona selainsovelluksen kehittämisessä ovat asiakaslähtöisyys, helppous, käytettävyys ja nopeus. Nämä ovat suuntaa-antavia periaatteita, joilla projektiryhmä pyrkii kehittämään sovellusta oikeaan suuntaan, eivätkä virallisia lupauksia.

Opinnäytetyö on systeemityötyyppinen. Siinä suunnitellaan ja luodaan mobiilioptimoidun käyttöliittymäsovelluksen prototyyppi. Lisäksi tutustutaan yleisesti selainsovelluksen toteuttamiseen ja sen optimoimiseen mobiilialustoille. Opinnäytetyössä kootaan projektin sovelluskehityksen aikana valmistunut vaatimusmääritys, käyttäjäkyselyn ja käytettävyystestauksen avulla kerätyt tulokset ja niiden analysoinnin pohjalta ilmenneet muutokset. Lisäksi esitellään projektissa valmistunut selainsovellus ja sen toteuttamisessa käytetyt arkkitehtoniset ratkaisut ja teknologiat. Vaatimusmäärityksessä keskitytään osittain prototyyppiin ja osittain valmiiseen selainsovellukseen sekä käsitellään joitakin alustavia toiminnallisuuksia.

1.1 Opinnäytetyön rajaus

Opinnäytetyön tarkoituksena ei ole ainoastaan luoda uutta käyttöliittymäprototyyppiä toimintajärjestelmälle, vaan samalla tuoda uusia teknologioita käyttöön ja kokeiltaviksi yrityksen tuotekehityksen työnkulkuun. Selainsovellusta kehitettäessä pyritään käyttämään uusia teknologioita, joiden avulla sovellus pidetään kilpailukykyisenä ja varmistetaan sen toimivuus ja ajantasaisuus myös tulevaisuudessa. Itse opinnäytetyössä ei kuitenkaan opeteta näiden teknologioiden käyttöä, eikä niitä oteta yleisesti käyttöön yrityksen tuotekehitykseen. Uusia teknologioita tutkitaan, kokeillaan ja ne pidetään erillään muusta tuotekehityksestä, kunnes mobiilioptimoitu käyttöliittymä todetaan julkaisukelpoiseksi. Tästä on poikkeuksena palvelinpuolen arkkitehtuuri, joka haaroitetaan projektin aikana muihin tuotekehityksen projekteihin.

Opinnäytetyöhön ei kuulu valmiin sovelluksen rakentaminen tai sen täydellinen määrittely. Projektissa valmistuva sovellus on vasta prototyyppi, joka sisältää joitakin alustavia toiminnallisuuksia ja joka on helposti jatkokehitettävissä ja yhdistettävissä yrityksen toimintajärjestelmän nykyisen käyttöliittymän rinnalle. Prototyypin toiminnallisuudet on rajattu koskemaan toimintajärjestelmän katsojakäyttäjäryhmälle olennaisia toiminnallisuuksia järjestelmän Dokumentit-osiossa.

Vaatimusmäärityksessä keskitytään pääosin prototyyppiin. Projektissa kuitenkin pyritään ottamaan huomioon jatkokehitykseen ja lopulliseen sovellukseen vaikuttavia tekijöitä sikäli, kun niitä tulee vastaan ja niiden huomioon ottamiselle riittää resursseja. Prototyypistä kehitetään toimiva tuote, joka on tarkoitus ottaa käyttöön tulevaisuudessa heti, kun se on todettu julkaisuvalmiiksi. Prototyypin kehittämiseen ei kuulu tietokantarakenteen määrittely, sillä mobiilioptimoitu käyttöliittymä tulee hyödyntämään IMS-toimintajärjestelmän nykyisiä tietokantoja.

Projektiin kuuluu prototyypin ulkoasun suunnittelu ja toteutus, mutta tälle ei asetettu erityisiä määrityksiä. Ulkoasu suunnitellaan ja toteutetaan projektin alkupuolella ja sen jälkeen sitä parannellaan projektin aikana saadun palautteen perusteella. Lisäksi opinnäytetyöprojektiin ei kuulu sovelluksen tuotteistaminen, nimeäminen tai

markkinointi. Näitä asioita pyritään kuitenkin jossakin määrin miettimään ja ottamaan huomioon prototyypin ulkoasua ja käytettävyyttä kehittäessä.

1.2 Opinnäytetyön työnjako

Opinnäyte suoritetaan parityönä, jossa molemmilla ryhmän jäsenillä on selkeät aihealueensa ja vastuunsa. Vastuualueista huolimatta molemmat jäsenet tukevat toisiaan koko opinnäytetyön ajan. Patrik Vilja on vastuussa prototyypin ulkoasun suunnittelusta ja toteuttamisesta. Hän on myös vastuussa sovellusarkkitehtuurin suunnittelusta ja rakentamisesta sekä sovelluksen teknisestä puolesta. Lisäksi Patrik Vilja vastaa siitä, että projektiryhmä noudattaa Scrum-menetelmän tapoja. Ossi Hirvikoski puolestaan on vastuussa kyselyn toteuttamisesta, demojen testaamisesta asiakkailla, hyväksymistesteistä ja projektinhallinnasta. Patrik Vilja on vastuussa opinnäytetyön luvuista 3.6–3.10 ja 7. Ossi Hirvikoski on vastuussa luvuista 3.1–3.5, 3.11, 4–5 ja 8–10. Muut luvut on kirjoitettu yhteisvoimin.

2 Opinnäytetyöprojektin työsuunnitelma

Mistä sovelluksen kehittäminen kannattaisi aloittaa ja mitä siihen tulisi sisällyttää? Sovelluksen suunnittelu aloitetaan keräämällä tilaajalta sovellukselle esitettäviä vaatimuksia ja rajaamalla ne sopivan kokoiseksi kokonaisuudeksi. Projektin päätavoitteiksi asetettiin käyttöliittymän uudistaminen mobiilioptimoiduksi, uusien teknologioiden hyödyntäminen ja asiakasläheinen testaus sekä kehitys. Näiden tietojen perusteella luodaan kokonaiskuva projektista ja kirjoitetaan aihe-ehdotus.

Alustavaa tietoa sovelluksen toiminnoista ja niiden priorisoinnista ajateltiin hankkia tekemällä käyttäjäkysely. Kyselyn muoto, kohderyhmä ja sisältö olivat kuitenkin avoimia. Mitä kannattaisi kysyä, miten ja keneltä? Ajateltiin myös tutkia, olisiko jokin muu keino kuin kysely parempi alustavan tiedon hankkimiselle.

Tämän jälkeen pyritään kokoamaan vaatimukset määrityksiksi, joiden pohjalta lähdetään suunnittelemaan sovelluksen rakennetta ja toimintoja. Vaatimusmäärityksessä pyritään osittain ottamaan huomioon valmiille sovellukselle tärkeät kriteerit. Toiminnallisuuksissa kuitenkin keskitytään vain prototyyppiin rajattuihin toimintoihin. Ryhmä pyrkii tutustumaan kirjallisuuteen sekä etsimään tietoa internetistä prototyypin ja sen toiminnallisuuksien toteuttamiselle.

Koska projektin yksi tärkeimmistä tavoitteista on keskittyä sovelluksen käytettävyyteen, pyritään mahdollisimman ajoissa suorittamaan käytettävyystestausta. Tämän vuoksi projektin aikana toteutetaan yksinkertainen mobiilioptimoidun käyttöliittymän prototyypin toimintaa matkiva demo. Käytettävyystestausta varten etsitään suosituksia, joita pyritään soveltamaan parhaiden tai vähintään hyvien tulosten saavuttamiseksi. Kohderyhmäksi otetaan yrityksen asiakkaat, koska he tulevat olemaan sovelluksen loppukäyttäjiä. Käytettävyystestauksesta saatujen tulosten perusteella on mahdollista tehdä parannuksia käyttöliittymään. Parannusten toteuttamisen jälkeen käyttöliittymälle suoritetaan uudestaan käytettävyystestausta. Tämän pohjalta rakennetaan lopullinen toimiva prototyyppi.

Teoriataustaa pyritään hankkimaan aiheista

- mobiilitarve ja mobiilikäyttö
- käytettävyys
- kysely tai vaihtoehtoiset tiedonhankintamenetelmät
- sovelluksessa käytetyt ratkaisut ja teknologiat
- käytettävyystestaus
- hyväksymistestaus.

3 Teoriatausta

Tässä luvussa esitetään syitä projektin toteuttamiselle, esitellään tärkeimpiä projektissa hyödynnettyjä teknologioita ja selitetään joitakin keskeisiä käsitteitä. Oleellisimpia asioita ovat, miksi mobiilisovellus on tarpeen toteuttaa, mitkä ovat mobiilioptimoinnin ja käytettävyyden mahdolliset kriteerit sekä mitkä ovat prototyypin toteuttamiseen ja testaamiseen valitut teknologiat.

3.1 Mobiilitarve

Internetselaaminen mobiililaitteilla on huomattavassa kasvussa, ja sen ennustetaan ohittavan selaamisen työpöytäkoneilla ja kannettavilla tietokoneilla vuosien 2013–2014 välillä (Ingram 2010; Gartner 2011). Vuonna 2011 tehdyn kyselyn mukaan joka neljäs suomalainen omisti älypuhelimen (TNS Gallup 2011).

Vuoden 2012 tammi- ja syyskuun välillä myytiin Suomessa yli 1,1 miljoonaa älypuhelinta. Edellisen vuoden samaan ajanjaksoon verrattuna älypuhelinten myynti on kasvanut lähes 57 %. (Taloussanomat 2012.) Maailmanlaajuisesti älypuhelinten myynti on jo ohittanut työpöytäkoneiden myynnin (Blodget & Cocotas 2012).

Avanaden tutkimuksen mukaan jopa 61 % yrityksistä on ottanut käyttöön ”tuo omat välineesi”-trendin[1], jossa työntekijät saavat käyttää työssään omia työvälineitä, kuten älypuhelimia ja tabletteja. Yli puolet tutkimukseen osallistuneista yrityksistä ilmoitti, että suurin osa heidän työntekijöistään käyttää älypuhelinta työpaikalla. (Butcher 2012.)

Epämiellyttävät kokemukset sivuston mobiilikäytöstä saattavat huonontaa yrityksen imagoa. Kävijät saattavat helposti tuntea, että yritys ei välitä heistä, jos palveluita ei voi käyttää myös mobiililaitteilla. (Indvik 2012.) Muutama IMS Business Solutions Oy:n asiakkaista on valittanut, ettei toimintajärjestelmää voi käyttää kunnolla mobiililaitteilla

[1] Käännös englanninkielisestä termistä BYOD eli ”Bring Your Own Device”

(Lakso 2012). Huonosti tehty mobiilisovellus puolestaan saattaa olla huonompi asia kuin se, että yrityksellä ei ole mobiilisovellusta lainkaan (McWherter & Gowell 2012, 2).

IMS Business Solutions Oy:n asiakkaille tehty kysely paljasti, että noin puolet vastanneista kokee tarpeelliseksi IMS-toimintajärjestelmän sisällön käytön jollakin mobiililaitteella nyt tai tulevaisuudessa (liite 3).

3.2 Mobiilioptimointi

Mobiilikäytettävyydelle löytyy kolme tärkeää termiä: mobiiliystävällinen (mobile friendly), mobiilioptimoitu (mobile optimized) ja mukautuva websuunnittelu (Responsive Web Design). *Mobiiliystävällisellä* sivulla sivuston tärkeimmät tiedot ovat luettavissa ja toiminnot toimivat mobiililaitteilla. Sivuilla ei käytetä suuria kuvia tai mobiililaitteilla toimimatonta tekniikkaa, kuten Flash-animaatiota. *Mobiilioptimoitu* sivusto muuttaa sivun elementit vastaamaan paremmin mobiilikäyttöä, kun sivua luetaan mobiililaitteella. Sivuston napit sopivat paremmin sormella painettaviksi ja sivuilla on vähemmän häiritseviä elementtejä. *Mukautuva websuunnittelu* on tekniikka, jolla sivusto saadaan joustavasti näyttämään hyvältä ja käytettävältä lukulaitteen tai selaimen koosta riippumatta. (Olson 2012.)

Mobiilioptimoitu sivusto järjestää uudelleen informaatiota. Mobiililaitteet ovat pieniä, joten tavoitteena on usein keventää näytettyä sisältöä ja tarjota mobiilikäyttäjälle vain tarpeellista informaatiota. (201Proof 2012; Terrill 2012.)

Mobiilikäyttöön suunniteltavissa sivuissa tulee ottaa huomioon, ettei käyttäjän sormi ole yhtä tarkka tai ketterä kuin tietokoneen hiiri (Terrill 2012). Terrill (2012) suosittelee painettavien kohteiden kooksi 40x40 pikseliä. Applen vastaava suositus on 44x44 pistettä, Microsoftin 9 mm tai 34 pikseliä ja Nokian 1 cm x 1 cm (Wroblewski 2010). Luettavuuden kannalta fontin koon tulisi olla ainakin 14 pikseliä (Terrill 2012).

Nykyään on ongelmana, että käyttäjillä on käytössään eri laitteita, joissa on erikokoisia näyttöjä, resoluutioita jne. Tulevaisuudessa laitteiden kirjo ja vaihtelevuus tulee olemaan yhä suurempi verrattuna nykypäivään. (Terrill & Sherrett 2012, 5.)

Suorituskyky on olennainen osa mobiilikäytettävyyttä. Sitä voi parantaa vähentämällä HTTP-pyyntöjen määrää, optimoimalla sivulla olevat kuvat tarpeeksi pieniksi, hallitsemalla tyylejä ja skriptejä oikein sekä käyttämällä CSS-tyylikieltä animointiin JavaScriptin sijaan. (Terrill & Sherrett 2012, 8.)

3.3 Sovellustyypit

Mobiilisovelluksella tarkoitetaan sovellusta, joka on ohjelmoitu käytettäväksi mobiilikäyttöjärjestelmään[2]. (La Counte 2012, 1) Mobiilisovelluksesta käsin pystytään suoraan käyttämään mobiililaitteen ominaisuuksia, kuten kameraa tai mikrofonia (McWherter & Gowell 2012, 17–18). Mobiilisovellus nähdään usein trendikkäänä vaihtoehtona, ja se mahdollistaa sisällön esittämisen ilman internetyhteyttä (McWherter & Gowell 2012, 22). Innostus tehdä yritykselle mobiilisovellus tulee usein yrityksen johdolta, koska mobiilisovellukset ovat tämän ajan trendi. Usein ei kuitenkaan ole tarvetta lähteä rakentamaan mobiilisovellusta, vaan selaimella toimiva sovellus riittää kattamaan yrityksen tarpeet. (McWherter & Gowell 2012, 11.) Ongelmana on, että mobiilisovellus täytyy ohjelmoida erikseen eri käyttöjärjestelmille ja laitteille (La Counte 2012, 5).

Selainsovellus on internetselaimella toimiva sovellus, jolle navigoidaan samalla tavoin kuin mille tahansa muulle internetsivulle. Se rakennetaan pääasiassa HTML-kieltä ja CSS-tyylikieltä käyttäen. (McWherter & Gowell 2012, 33.) Selainsovellus ottaa huomioon puhelinten rajoitukset. (La Counte 2012, 4). Selainsovelluksen hyvänä puolena on, että se toimii kaikissa käyttöjärjestelmissä, joista löytyy internetselain (McWherter & Gowell 2012, 35). Selainsovelluksen voi rakentaa pääasiassa tavallisten internetsivujen rakentamiseen käytettyjen teknologioiden avulla, joten kehittäjien ei

[2] Android, BlackBerry, iOS, Symbian, Windows Phone jne.

tarvitse sitoutua opettelemaan uutta ohjelmointikieltä tai -tekniikkaa. Selainsovellus on helpompi päivittää, eikä se vaadi mahdollista ulkopuolista hyväksyntää päästäkseen yleiseen levitykseen. (McWherter & Gowell 2012, 35–36.)

3.4 Käyttäjäkysely

Kyselyistä on tullut olennainen osa lähes mitä tahansa tutkimustyötä. Selainpohjaiset kyselyt ovat edullisin vaihtoehto yrityksen asiakkaiden tarpeiden ymmärtämiseksi. Ne ovat työkaluja, joilla kerätään elektronisesti kyselytuloksia kohdeyleisöltä internetin välityksellä. (Bhaskaran & LeClaire 2012, 9.)

On lähes mahdotonta suunnitella kyselyä, jossa tulokset vastaisivat täysin käyttäjien mieltymyksiä. (De Leeuw, Hox, & Dillman 2008, 98.) Kyselyiden on kuitenkin todettu tuottavan tuloksia kerta toisensa jälkeen. Lähes kaikki kauppojen hyllyillä näkyvät tuotteet ovat sellaisen markkinatutkimuksen tulosta, johon on kuulunut kyselyn tekeminen jossakin muodossa tai jollakin tavoin. (Bhaskaran & LeClaire 2012, 10.)

Kyselyjä on pääasiassa kahdenlaisia: haastatteluja ja itsehallinnoituja kyselyjä. Näitä kyselyjä voidaan suorittaa monin eri tavoin. Haastatteluja voidaan tehdä puhelimen välityksellä tai kasvotusten. Itsehallinnoituja kyselyjä voidaan suorittaa ryhmätilanteissa kuten luokkahuoneessa, sähköpostin avulla tai internetissä. Haastattelujen etuna mutta myös heikkoutena on mahdollisuus vaikuttaa kyselyyn. Haastattelija pystyy tarkentamaan kysymyksiä, mutta toisaalta saattaa olemuksellaan ja sanoillaan vaikuttaa haastateltavan tekemiin valintoihin. (De Leeuw ym. 2008, 113–115.)

Kyselyn suoritustapa vaikuttaa myös käyttäjän mahdollisuuteen keskeyttää kyselyyn vastaaminen. Internetissä suoritettava kysely on helpointa jättää kesken. Tästä syystä tällaisen kyselyn tulisi olla suhteellisen lyhyt: jo 10–15 minuuttia kestävä kysely saattaa tuntua vastaajasta puuduttavalta. Puhelimessa tai kasvotusten tehty kysely saa kestää huomattavasti kauemmin ilman, että haastateltava kyllästyy. (De Leeuw ym. 2008, 121.)

Haastatteluna toteutetusta kyselystä voidaan tehdä joustava ja monimuotoinen. Näitä ominaisuuksia on huomattavasti vaikeampi toteuttaa sähköpostin tai internetin välityksellä tehtävässä kyselyssä. Niiden etuna on kuitenkin se, että kyselyyn voi vastata vapaasti omalla ajallaan ja missä vain haluaa. Haittapuolena taas on, etteivät osanottajat välttämättä ota internetissä tehtävää kyselyä vakavasti, vaan antavat pikaisia vastauksia tarkemmin miettimättä asiaa. (De Leeuw ym. 2008, 122–123.)

3.5 Käytettävyys

International Standards Organization (ISO 9241-11) esittää kolme käytettävyyden aspektia. Käytettävyys määritetään sen perusteella, mikä on se tehokkuus, hyötysuhde ja tyytyväisyysaste, jolla käyttäjät voivat käyttää tuotetta saavuttaakseen haluttuja tuloksia. (Tullis & Albert 2008, 4.)

Suurimmaksi osaksi käytettävyys tarkoittaa turhautumisen puuttumista jotakin asiaa käytettäessä. Tuote tai palvelu voidaan todeta käytettäväksi, kun käyttäjä voi tehdä sillä mitä haluaa sellaisella tavalla, jolla kuvittelee pystyvänsä tekemään tämän asian, eikä tämä suoritus aiheuta hänessä epäröintiä, kysymyksiä tai haittaa. (Rubin & Chisnell 2008, 4) Käytettävyys muodostuu siitä, kuinka helposti ihmiset (käyttäjät) pystyvät suoriutumaan tarvittavista tehtävistä. Käytettävyyden tärkein kriteeri on, että ohjelmisto pystyy suorittamaan käyttäjän tarvitsemat toiminnot. Toimintojen suorittaminen tulisi olla nopeaa ja vaivatonta. Ohjelmiston ja sen toimintojen oppiminen tulisi olla helppoa eikä saisi vaatia käyttäjää muistamaan mitään ylimääräistä. Ohjelmiston tulisi pyrkiä estämään virhetilanteita ja sen tulisi olla käyttäjän mielestä miellyttävää käyttää. Lyhyesti sanottuna käytettävyys koostuu opittavuudesta, tehokkuudesta, muistettavuudesta, virheettömyydestä ja miellyttävyydestä (Brinck, Gergle & Wood 2002, 2–3).

Vakavat käytettävyysongelmat ovat yleisiä, vaikka internetsivut perustuvatkin hyvin yksinkertaisista elementeistä rakennettuihin kokonaisuuksiin (Brinck ym. 2002, 4). Käyttäjiä tulisi ajatella aikaisin ja usein. Käytettävyyden tulisi olla osa jokaista suunnittelun vaihetta. Käytettävyyden parantamiseksi tulisi kerätä tietoa ohjelmiston käytettävyydestä. Tätä tietoa voi kerätä joko suoraan käyttäjiltä tai sen voi perustaa

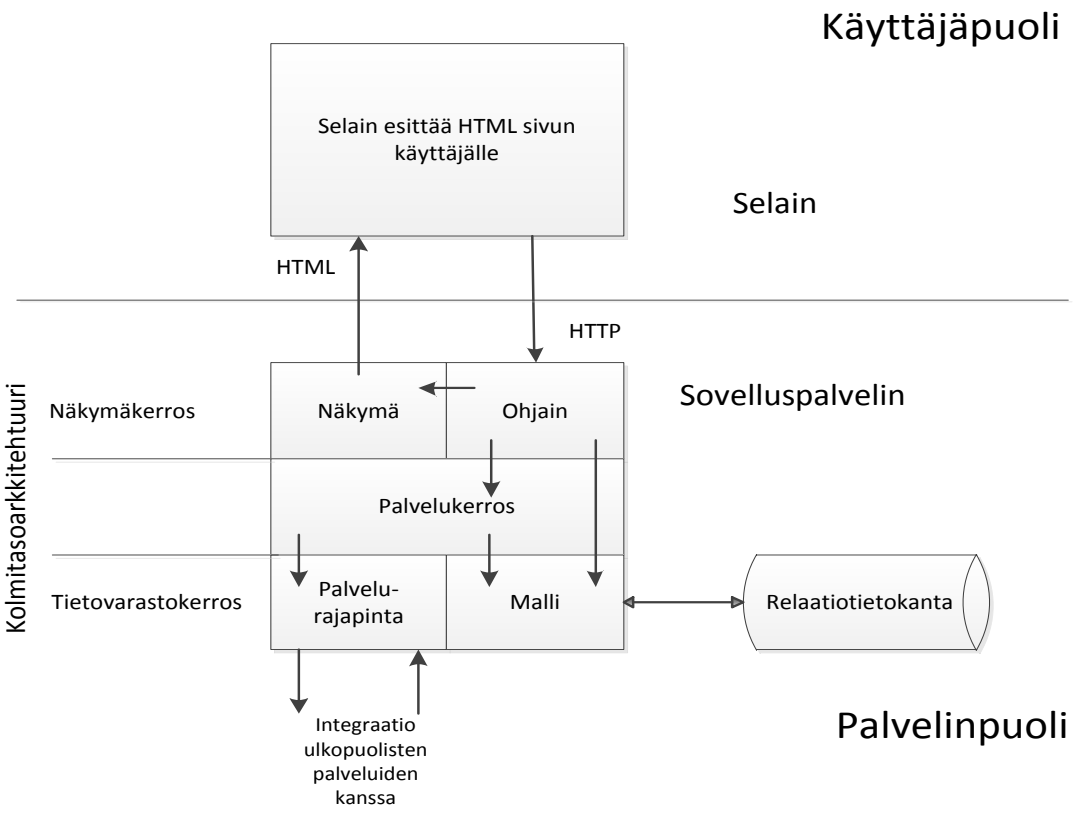
aiempiin kokemuksiin, tunnettuihin käyttäjien toimintatapoihin tai yleisiin käytettävyysperiaatteisiin. (Brinck ym. 2002, 14–15.)

Yksi suurimmista haasteista on suunnitella sivusto suurelle määrälle erilaisia käyttäjiä ja erilaisille laitteistoille, ohjelmistoille ja internetyhteyksien eri nopeuksille. Täydellistä ohjelmistoa on mahdotonta luoda. Kannattavinta on pyrkiä suunnittelemaan, kuinka hyvin haluaa sivuston toimivan eri kohderyhmän joukoille. (Brinck ym. 2002, 42.)

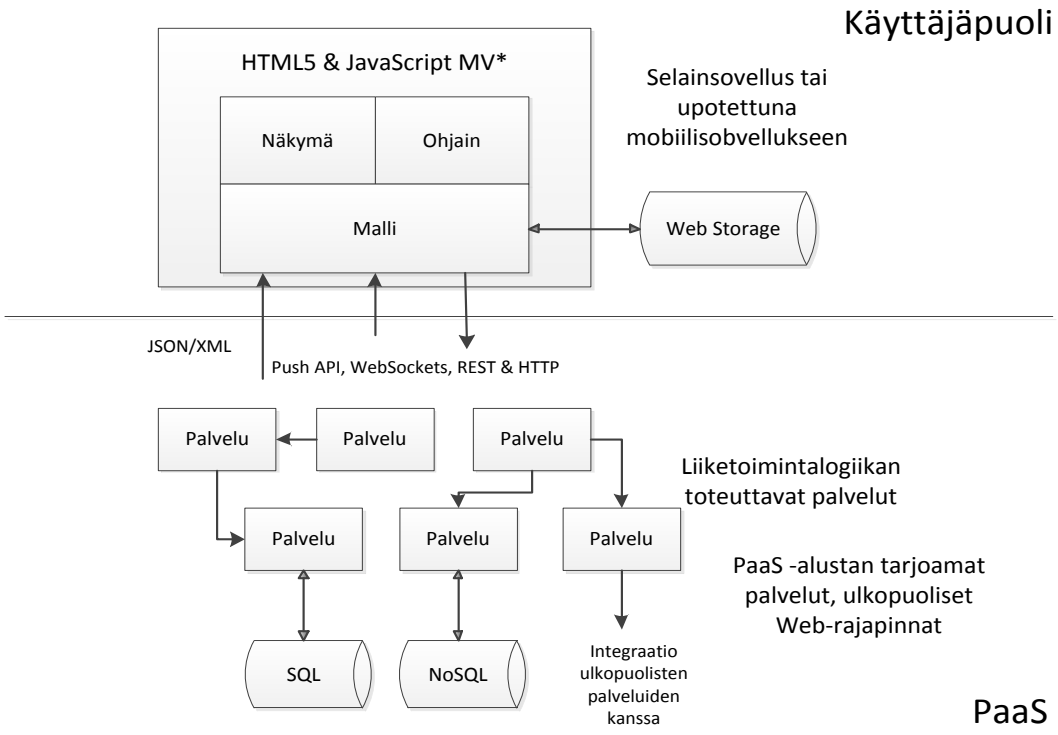
Monet yritykset ovat erehtyneet käyttämään uusia trendikkäitä teknologioita pelkän silmän ilon tuottamiseksi. Väärinkäytettynä uusimmat teknologiat vain haittaavat sivuston käyttöä. (Brinck ym. 2002, 342.)

3.6 Arkkitehtuuri

Viimeisten kymmenen vuoden aikana websovellusarkkitehtuuri on pysynyt pohjimmiltaan samanlaisena. Palvelinpuolen toteutus vastaa sivujen rakentamisesta ja lähettää ne valmiina käyttäjän selaimeen. Selaimen vastuulle on jäänyt ainoastaan sivujen esittäminen (Kuvio 1). Nykypäivän internetselaimet ovat aivan eri tasoa kuin neljä vuotta sitten. Ne kykenevät suorittamaan JavaScriptiä erittäin nopeasti osittain selaimia tuottavien yritysten välisen kilpailun ansiosta. Parantunut JavaScriptin suorituskyky on viimein tuonut sovellusarkkitehtuuriin uuden ulottuvuuden eli käyttäjäpuolen (client-side) arkkitehtuurin (Kuvio 2). (Colyer 2012.)



Kuvio 1. Perinteisessä arkkitehtuurissa palvelinpuoli toteuttaa kolmitasoarkkitehtuurin (Colyer 2012)



Kuvio 2. Palvelukeskeinen arkkitehtuuri (SOA), joka jakaa näkymäkerroksen vastuun käyttäjäpuolelle omaksi arkkitehtuurikseen (Coyer 2012)

Käyttäjäpuolen arkkitehtuurin ansiosta osa palvelinpuolen vastuista siirtyy suoritettavaksi käyttäjän selaimessa JavaScriptin avulla. Käyttäjäpuolen arkkitehtuuri vastaa interaktiivisista näkymistä ja toiminnoista hyödyntäen selaimen ominaisuuksia kuten HTML5 Web Storage -tekniikkaa. Näin saadaan aikaan entistä rikkaampi ja interaktiivisempi käyttökokemus. Erotettu käyttäjäpuoli voi toimia täysin omana sovelluksenaan, joka keskustelee HTTP-pyyntöjen ja WebSocket-tekniikan avulla useamman palvelinpuolen palvelun kanssa. (Colyer 2012.)

Palvelinpuolen arkkitehtuuri ei ole enää vastuussa näkymistä, joten sen rakenne ei vastaa vanhaa tuttua MVC-arkkitehtuuria. Palvelinpuolen arkkitehtuuri voidaan jakaa useampiin itsenäisiin palveluihin (SOA eli Service Oriented Architecture), jotka lähettävät ja vastaanottavat tietoa REST-arkkitehtuurimallin rajapintojen avulla. Rajapintojen kautta palvelut voivat kommunikoida toistensa ja ulkopuolisten websovellusrajapintojen kanssa. Lisäksi itsenäinen palvelu voi olla toteutettu millä tahansa kielellä tai kehyksellä ja se voi hyödyntää mitä tahansa tietokantaa. Tämän palvelukeskeisen arkkitehtuurimallin myötä sovellus voidaan ulkoistaa paremmin pilviarkkitehtuuriin kuten PaaS (Platform as a Service). Yksittäisen palvelun ainoa vaatimus on tarjota Web Services -rajapinta. (Colyer 2012.)

3.6.1 Monikerroksinen palveluarkkitehtuuri

Selainsovelluksen arkkitehtuurin jakaminen useampaan itsenäiseen osaan on yleisesti käytetty arkkitehtuurimalli. Jokaisella itsenäisellä osalla on oma vastuualueensa, ja tätä mallia kutsutaan usein nimellä kolmitasoarkkitehtuuri. (Dashorst & Hillenius 2009, 300.)

Kolmitasoarkkitehtuurissa sovellus jaetaan kolmeen kerrokseen: näkymäkerrokseen, palvelukerrokseen ja tietovarastokerrokseen. *Näkymäkerros* on vastuussa käyttöliittymästä. Se vastaa kaikista käyttöliittymän pyynnöistä ja siten on vastuussa

pyydetyn resurssin esittämisestä. *Palvelukerros* sisältää sovelluksen liiketoimintalogiikan. Se koostuu useammista palveluista, jotka yhdessä vastaavat liiketoimintalogiikan tiedonkäsittelystä sekä tiedon ohjaamisesta näkymä- ja tietovarastokerroksien välillä. *Tietovarastokerros* puolestaan tarjoaa yhteyden tietovarastoon, kuten tietokantaan, web-palveluun tai tiedostojärjestelmään. (Kuvio 1.) (Dashorst & Hillenius 2009, 300.)

Arkkitehtuurin jakaminen modulaarisiin, väljästi kytkettyihin kerroksiin mahdollistaa paremman ylläpidettävyyden. Itsenäinen kerros ottaa mahdollisimman vähän kantaa muiden kerroksien toteutukseen, joten yksittäinen kerros voidaan uusia ilman muutoksia muissa kerroksissa. Tavoittaakseen kerroksien itsenäisyyden sovelluksen osien on keskusteltava interface-rajapintojen kautta. Tällöin yhdellä palvelukerroksen palvelulla voi olla useampi itsenäinen toteutus ilman, että ne vaikuttavat muiden kerroksien rakenteeseen. Rakenteessa on havaittavissa selvä samankaltaisuus MVC-arkkitehtuurin kanssa, mutta ne eroavat perusperiaatteiltaan. (Dashorst & Hillenius 2009, 300–301.)

Alkuperäisessä Smalltalk-ohjelmointikielestä tutussa MVC-arkkitehtuurissa malli vastaa muutoksien ilmoittamisesta näkymille tarkastelijoiden avulla. (Gamma, Helm, Johnson, Vlissides 1994, 14.) Kolmitasoarkkitehtuuri ei puolestaan ota kantaa mallin sijoittamiseen rakenteessa, sillä mallia yleensä tarvitaan jokaisella tasolla. Kolmitasoarkkitehtuurissa näkymältä malliin tulevan muutoksen täytyy mennä palvelukerroksen ja tietovarastokerroksen läpi. (Dashorst & Hillenius 2009, 300–301.) MVC-arkkitehtuuri puolestaan mahdollistaa näkymän ja mallin välisen suoran interaktion. (Gamma ym. 1994, 14–15.)

3.6.2 Spring Framework ja JPA

Spring Framework on yksi tämän hetken johtavista avoimen lähdekoodin Java-ohjelmointikehyksistä. Se pyrkii tarjoamaan kattavan kehysratkaisun erilaisille nykypäivän Java-ohjelmistokehityshaasteille. Spring-kehys toteuttaa JSR (Java Specification Requests) -spesifikaatioita ja täten toimii vaihtoehtoisena ratkaisuna Java EE -sovelluskehykselle. (Walls 2011, 3–4.)

Spring Framework tarjoaa lukuisia erilaisia hyväksi todettuja arkkitehtonisia ratkaisuja ja tekniikoita, kuten REST-arkkitehtuurimallin, sekä integraatioita muiden spesifikaatioiden ja kirjastojen kanssa, kuten JPA-standardi. Spring-kehys pohjautuu suurelta osin Inversion of Control (IoC) -säiliöön, Dependency Injection (DI) -tekniikkaan, Aspect Oriented Programming (AOP) -ohjelmointiin ja ajon aikaiseen luokka-tiedostojen muokkaamiseen. Valmiiden arkkitehtonisten ratkaisujen pohjalta sovelluskehittäjä voi arkkitehtonisten ongelmien sijasta keskittyä omaan sovellukseensa ja liikeideaansa. (Walls 2011, 4–5.)

Java Persistence API (JPA) on standardi, joka pyrkii yhtenäistämään Object Relation Mapping (ORM) -tietokantakehyksien toimintatavat. JPA-standardia täytyy käyttää ORM-tietokantakehyksen toteuttajan kautta, sillä JPA ei sisällä omaa riippuvuuskirjastoa. JPA tarjoaa erilaisia tietokantaan liittyviä toimintoja ja ratkaisuja, kuten esimerkiksi entiteetit, tietokantatauluviittaukset ja JPQL-syntaksin. ORM-ajattelumallin myötä tietokantakeskeisestä ohjelmoinnista saadaan oliopohjaista. Toisin sanoen tietokantataulu on käytössä sovelluskoodissa oliona entiteettien avulla. Tunnetuimpia JPA-standardin toteuttajia ovat Hibernate, Toplink, OpenJPA, EclipseLink ja ObjectDB -kehykset. (Keith & Schincariol 2009, 11–16; Walls 2011, 138–142.).

3.6.3 REST

Representational State Transfer (REST) on vaihtoehtoinen arkkitehtuurimalli palvelukeskeisen (SOA) web-palvelun toteuttamiseen. Se kuvaa palvelujen välillä siirrettävän resurssin tilaa riippumatta resurssin esitysmuodosta. (Walls 2011, 278–279.)

- **Representational** määrittää REST-resurssien esitysmuodon. Yleisimmin siirrettävät resurssit ovat joko JSON-, XML- tai HTML-muodossa. REST-arkkitehtuurimalli ei kuitenkaan rajaa esitysmuotoa, ja mikä tahansa tiedonsiirtomuoto on sallittua. Lisäksi yksi resurssi voidaan esittää useammassa esitysmuodossa. (Walls 2011, 278.)

- **State** eli resurssin tila voidaan määrittää HTTP-spesifikaatioon semanttisten metodien avulla. Metodeja ovat mm. GET, POST, PUT, DELETE ja OPTIONS[3]. REST-arkkitehtuurimallissa ollaan enemmän kiinnostuneita resurssin tilasta kuin siitä, mitä resurssille voidaan tehdä. (Walls 2011, 278.) Metodi kuvaa resurssin tilaa sekä siihen kohdistuvaa semanttista toimintaa. (Walls 2011, 284–285.)
- **Transfer** kuvaa resurssin siirtämistä sovelluksien välillä. (Walls 2011, 278.)

REST-arkkitehtuurimalli pohjautuu HTTP-spesifikaatioon, joten URL nousee tärkeään asemaan. URL (Uniform Resource Locator) eli myös URI (Uniform Resource Identifier) kertoo resurssin nimen, paikan ja yksilöi sen. (Richardson & Ruby 2007, 79–82.) REST URL -periaatteelle on ominaista yksilöidä resurssi hierarkian avulla. Hierarkkista osoitetta luetaan vasemmalta oikealle. Esimerkiksi GET-metodin sisältävä pyyntö osoitteeseen *http://localhost:8080/documents* palauttaa kaikki dokumentit ja *http://localhost:8080/documents/1* palauttaa tunnisteella yksi olevan dokumentin. REST URL ei kuitenkaan yksinään riitä määrittämään toimintoa kuten resurssin päivittämistä. (Walls 2011, 281–282.)

URL ja HTTP-metodit määrittävät yhdessä REST URL -osakokonaisuuden. Kukin HTTP-metodi sisältää semanttisen tarkoituksen. Metodien ansiosta yhden URL-osoitteen takana voi olla useampi toiminto. GET-metodi suorittaa lukuoperaation ja palauttaa yhden tai useamman samantyyppisen resurssin. POST-metodi vastaa tallennusoperaatiosta. PUT-metodi puolestaan kuvaa päivitys- tai tallennusoperaatiota ja DELETE-metodi suorittaa poisto-operaation. (Burke 2009, 7–8.) Esimerkiksi DELETE-metodin sisältämä pyyntö osoitteeseen *http://localhost:8080/documents/1* poistaa dokumentin, jonka tunniste on yksi[4]. Nämä metodit ovat täysin semanttisia, ja siten jää täysin käyttäjän vastuulle noudattaa metodien semantiikkaa. Toisin sanoen esimerkiksi GET-metodin omaava pyyntö ei saisi koskaan päivittää resurssin tilaa. (Walls 2011, 284–287.)

[3] HTTP-spesifikaatio määrittää myös HEAD, PATCH, TRACE ja CONNECT metodit.
[4] Vertaa edellisestä kappaleesta löytyvään identtiseen osoitteeseen.

REST-arkkitehtuurimallin kokonaisuuden täydentää esitysmuoto. Esitysmuoto määritellään HTTP-pyynnön Accept-tunnisteen mediatyypin perusteella. Accept-tunnisteen avulla voidaan määrittää, missä muodossa tieto halutaan vastaanottaa REST-rajapinnalta. Mikäli tunnistetietoa ei ole annettu, palautetaan resurssi REST-rajapinnan tarjoajan määrittämässä oletus-esitysmuodossa. Accept-tunnisteeseen määriteltäviä mediatyyppejä ovat muun muassa text/html, application/json ja application/xml. Accept-tunnisteen ohella on myös mahdollista määrittää esitysmuoto .json, .xml ja .html URL-päätteiden avulla. (Walls 2011, 287–289.)

3.7 Mukautuva websuunnitelu

Nykyisin internetiin pääsee lukuisilla eri laitteilla aina älypuhelimista 60-tuumaisiin televisioihin. Samalla internetyhteydet vaihtelevat nopeista 100M laajakaistayhteyksistä hitaisiin 3G-yhteyksiin. Lukuisat erilaiset ympäristöt luovat haasteita web-käyttöliittymän joustavuudelle. (Carver 2012, 3).

Mukautuva websuunnittelu (Responsive Web Design) tähtää selainsovellusten käytettävyyteen riippumatta siitä, käytetäänkö sovellusta tietokoneella, mobiililaitteella vai millä tahansa selainta tukevalla laitteella. Se pyrkii sopeutumaan näytön kokoon säilyttäen toiminnallisuuden ilman, että käyttäjän tarvitse tarkentaa näytön sisältöä. Lisäksi se on vaihtoehtoinen ratkaisu mobiilisovelluksille tai omalle mobiilisivulle[5]. (Carver 2012, 8–10.) Tavoitteena ei ole mukauttaa sovellusta yleisten resoluutioiden pohjalta, vaan sen sijaan mukautua rakenteeseen vain silloin, kun se vaatii muutoksia. Näin sovellus saadaan skaalautumaan mahdollisimman moneen resoluutioon ja näytön kokoon. (Carver 2012, 17.)

Mukautuva websuunnittelu saadaan toteutettua CSS3-tyylikielen mukana tulleella uudella ominaisuudella nimeltään Media Queries. Media Queries -ominaisuus on

[5] Viittaa m- tai mobile-alkuisiin mobiilisivuihin kuten mobile.twitter.com, jotka on tarkoitettu ainoastaan mobiilikäyttöön

noussut erittäin tärkeään rooliin mukautuvien selainsovellusten mahdollistamiseksi. (Carver 2012, 15–16.)

3.8 CSS-tyylikieli

CSS3 on viimeisin Cascading Style Sheets (CSS) -sukupolvi. Sen kehitys on jaettu moduuleihin eli osakokonaisuuksiin (Hogan 2010, 1). Jokainen moduuli määrittää oman itsenäisen osuutensa CSS-spesifikaatiosta. Itsenäiset kokonaisuudet ovat helpommin hallittavissa ja spesifikaation ylläpidettävyys helpottuu. Moduuleilla on itsenäinen tasomääritys. Jokainen taso pohjautuu aikaisemman tason määritelmään ja tasomääritykset etenevät täysin omaa tahtiaan kussakin moduulissa. (Bos 2011.) CSS3-tyylikielessä viitataan kaikkeen, mikä on julkaistu CSS 2.1 jälkeen. Osa CSS3-tyylikielen myötä tulleista uusista moduuleista lähtivät etenemään tasolta kolme, ja täysin uudet moduulit, kuten Flexbox, aloitettiin tasolta yksi. Täten suositellaan käyttämään CSS-nimikettä ilman versionumerointia. (Atkins Jr 2012.)

CSS3-moduulien tuomat ominaisuudet ovat kasvattaneet tyylitiedoston mahdollisuuksia. Ennen CSS3:n tuomia ominaisuuksia oli yleistä hyödyntää kuvia ulkoasun tehostamiseksi. Kuvien avulla on mahdollistettu ulkoasusuunnittelijan piirtämien rautalankamallien tyylitehosteet, kuten varjot ja liukuvärit. Kuvilla on kuitenkin huonot puolensa. Jokainen kuva täytyy hakea palvelimelta näyttämistä varten. Tämän vuoksi syntyy jokaista kuvaa kohden yksi HTTP-pyyntö palvelimelle, jolloin kuvat kuormittavat palvelinta ja hidastavat sivun latausaikaa. Ongelman ratkaisemiseksi on kehitetty erilaisia vaihtoehtoja, kuten kuvatiedostojen asettaminen selaimen muistiin ja Sprite-kuvien käyttäminen. (Walton 2011.)

CSS3 on suunniteltu parantamaan sivun visuaalisuutta uusien, moduuleiksi jaettujen ominaisuuksiensa avulla. Ominaisuuksiin kuuluu muun muassa liukuvärit, varjot, pyöristetyt reunat, animaatiot, transitiot, läpinäkyvyys, pseudoluokat ja parannellut valitsimet mobiilioptimointia varten. (Hogan 2010, 139.) Nämä ominaisuudet vähentävät kuvien käytön tarvetta (Walton 2011).

3.8.1 Media Queries

Mukautuvan sivun muodostamiseen on jo vuosia hyödynnetty JavaScriptiä. CSS3 tarjoaa vihdoinkin standardin tavan toteuttaa mukautuvia sivuja pelkän CSS-tyylikielen avulla. Uusi ominaisuus on nimeltään Media Queries. Sen avulla voidaan muuttaa sivuston tyyliohjeita näytön ominaisuuksien pohjalta. Lisäksi selain, joka tukee Media Queries -spesifikaatiota[6], reagoi automaattisesti erilaisiin näytön muutoksiin. (Hogan 2010, 85–86.)

Näytöstä voidaan tunnistaa erilaisia ominaisuuksia, kuten korkeus, leveys, selaimen korkeus ja leveys, resoluutio, kuvansuhde, suunta ja värintoisto[7]. Tunnistettujen ominaisuuksien avulla voidaan määritellä kyseiseen näyttöön sopivat tyyliohjeet. (Hogan 2010, 85.) Media Queries -määrittelyyn voidaan vaikuttaa myös mediatyypin perusteella. Mediatyyppiasetuksia on useampia[8], kuten näyttö-, televisio- ja tulostusnäkymä, mutta esimerkiksi Firefox-selain tukee tällä hetkellä vain asetuksia print ja screen (Mozilla 2012a).

Media Queries -lohkoja voi olla useampia, ja yksittäinen lohko voi sisältää useampia ehtomäärityksiä. Näytön koko määritellään ehdossa maksimi- ja minimiraja-arvojen avulla. Raja-arvot määritellään tunnistustavan mukaisesti joko pikseleissä, resoluutiona tai tarkkuutena (dpi). Kun näytön koko vastaa määrittelyn arvoa, tulevat sen sisällä olevat tyyliohjeet voimaan. Media Queries -lohkojen täytyy olla määriteltyinä tyylitiedostossa viimeisinä, sillä ne ylikirjoittavat ja laajentavat muita tyyliohjeita. Toisin sanoen tyyliohjeet, joita ei ylikirjoiteta, jäävät voimaan. (Frain 2012, 87–96.)

3.8.2 CSS-esikäsittely

CSS-tyylikieli on staattista, eikä se sisällä funktionaalisuutta eikä muuttujia (Netherland, Weizernbaim, Eppstein & Matis 2012, 5). Tämän vuoksi isoissa projekteissa on ollut

[6] Selaintuki löytyy osoitteesta: http://caniuse.com/#feat=css-mediaqueries
[7] Lisätietoja löytyy osoitteesta: https://developer.mozilla.org/en-US/docs/CSS/@media
[8] Kaikki asetukset ovat luettavissa osoitteesta: https://developer.mozilla.org/en-US/docs/CSS/Media_queries

ongelmana ylläpidettävyys ja rakenteellinen toistuvuus. Ylläpidettävyyden ongelma on havaittu sovelluskehittäjien parissa, ja ratkaisuksi on kehitetty CSS-esikäsittely (CSS Preprocessing). (Netherland ym. 2012, 2–3.)

CSS-esikäsittely on saanut viimeisten kahden vuoden aikana suurta suosiota. Esikäsittelykielet tekevät CSS-tyylikielestä dynaamisen, kuitenkin pyrkien säilyttämään sen perusperiaatteet toimimalla laajennuksena CSS-tyylikielelle. (Kennedy & de León 2011, 261.) Kolme käytetyintä esikäsittelykieltä ovat Sass, LESS ja Stylus (Coyier 2012a.) Nämä kielet pyrkivät lisäämään funktionaalisuutta CSS-rakenteeseen. Funktionaalisella rakenteella parannetaan CSS-tyylikielen organisoinnista ja toistosta aiheutuneita ongelmia. Esikäsittelykielet mahdollistavat sisäkkäisen CSS-syntaksin, muuttujien käytön, mixin-laajennukset ja lukuisia muita ominaisuuksia, joita sovelluskehittäjät ovat kaivanneet CSS-tyylikieleen. Esimerkiksi sovelluksen päävärien määrittäminen muuttujiin lisää ylläpidettävyyttä, kun värin vaihtaminen onnistuu yhdestä paikasta. Funktionaalisen CSS-tyylikielen mahdollistamiseksi esikäsittelykielet käännetään lopuksi CSS-tyylikieleksi. (Kennedy & de León 2011, 261.)

Käännösprosessista vastaa kielen oma kääntäjä. Sass on Ruby-pohjainen toteutus, joten kielen kääntäminen vaatii Ruby-ohjelmointikieltä. (Netherland ym. 2012, 2–3.) Stylus ja LESS puolestaan ovat JavaScript-pohjaisia toteutuksia, joiden kääntäminen vaatii NodeJS:ää tai Javan Rhino -implementaatiota (LESS 2012; Stylus 2012). Käännösprosessi voidaan automatisoida tai tehdä käsin. Automatisoidussa käännösprosessissa kääntäjä tarkkailee esikäsittelykielen tyylitiedostoa. Mikäli tiedostoon tulee muutos, kääntäjä päivittää automaattisesti vastaavan CSS-tiedoston. Käännösprosessi mahdollistaa sen, että kääntäjä pystyy ilmoittamaan mahdollisesta virheellisestä syntaksista. (Coyier 2012b.)

3.8.3 Sass

Syntactically Awesome Stylesheets (Sass) on yksi tämän hetken suosituimmista esikäsittelykielistä. Se pyrkii toteuttamaan ohjelmoinnissa yleiseksi tulleen DRY (Don't Repeat Yourself) -aksiooman funktionaalisuuden avulla. (Netherland ym. 2012, 5.)

Sass tukee kahta eri syntaksia:

- **.sass** on sisennykseen pohjautuva syntaksi, joka on saanut vaikutteita Haml-merkintäkielestä. Syntaksi ei sisällä aaltosulkuja eikä puolipisteitä, vaan lohkot määritellään sisennyksien avulla. (Netherland ym. 2012, 4).
- **.scss** puolestaan pyrkii säilyttämään CSS-tyylikielestä tutun syntaksin, ja täten myös normaalin CSS-tyylikielen kirjoittaminen on mahdollista. Syntaksi esiteltiin Sass 3.0 versiossa. (Netherland ym. 2012, 4).

Nämä molemmat syntaksit ovat täysin tuettuja, eivätkä ne sisällä funktionaalisia eroavaisuuksia. Sass tulee jatkamaan molempien syntaksien tukemista, joten käyttäjä voi valita niistä mieluisemmaksi kokemansa. Sass sisältää myös muista esikäsittelykielistä tutut CSS-laajennukset, kuten muuttujat, sisäkkäisen syntaksin ja mixin-laajennukset. (Netherland ym. 2012, 6–10.)

3.8.4 Compass

Compass on Sass-kielelle kehitetty kehys, joka täydentää kieltä valmiiden yleisesti tarpeellisiksi todettujen mixin-laajennuksien avulla. Valmiit mixin-laajennukset pyrkivät toteuttamaan Sass-kielen päätarkoituksen eli toiston minimoimisen. Compass-kehyksen mixin-laajennukset tarjoavat hyväksi todettuja rakenteellisia suunnittelumalleja, kuten näytön kokoa myötäilevän ruudukko-asettelun. (Netherland ym. 2012, 13–14.)

Compass-kehyksen mixin-laajennukset on jaettu kolmeen sovelluskehittäjää auttavaan moduuliin: CSS3, Typography ja Utilities (Compass; Netherland ym. 2012, 13). *CSS3*-moduuli auttaa sovelluskehittäjiä selainetuliitteiden (prefix) käytössä. Compass-kehyksen avulla tyyliohjeissa ei tarvitse käyttää CSS3-määrittelyissä selainkohtaisia etuliitteitä. (Netherland ym. 2012, 25–26.) Compass-kehyksen ansiosta sovelluskehittäjän tarvitsee vain päivittää kehys uuteen versioon ja CSS3-määritykset pysyvät ajan tasalla. *Typography*-moduulin myötä käyttäjä voi helposti vaikuttaa yleisiin tekstimäärittelyihin ja *Utilities*-moduulin avulla puolestaan yleisiin tyylimäärittelyihin. (Compass.)

Sass-kääntäjä ei pysty kääntämään Compass-kehystä hyödyntäviä scss-tyylitiedostoja. Se ei ole tietoinen Compass-kehyksen sisältämistä ominaisuuksista. Compass täytyy ensin konfiguroida ja vasta tämän jälkeen scss-tiedostojen kääntäminen onnistuu Compass-kehyksen kääntäjällä. Compass-kehyksen kääntäjä laajentaa Sass-kielen kääntäjää mahdollistaakseen kehyksen ominaisuudet. (Compass; Netherland ym. 2012, 16–17.)

3.9 HTML5

HTML5 on HTML (Hypertext Markup Language) -standardin seuraava sukupolvi. Se tulee korvaamaan HTML 4.01, XHTML 1.0 ja XHTML 1.1 -standardit. HTML5 on tuonut mukanaan lukuisia uusia ominaisuuksia, joilla pyritään standardisoimaan nykyinen web-kehitys. (Pilgrim 2010, ix.) HTML5 on vielä kesken ja se on CSS3-tyylikielen tapaan jaettu itsenäisiin ominaisuuksiin (Pilgrim 2010, 15). Osa näistä ominaisuuksista on jo toteutettu monien selaimien[9] viimeisissä versioissa. (Can I use... 2012.)

Yksi tärkeimmistä uusista ominaisuuksista on semantiikka. Merkitsemättömät div-elementit haluttiin korvata, ja tilalle tuli toiminnallisuudeltaan poikkeamattomia semanttisia elementtejä, kuten header, section, article, nav, footer. Näillä elementeillä pyritään parantamaan HTML-merkintäkielen luettavuutta, rakenteellisuutta ja etenkin semanttisuutta. (Pilgrim 2010, 41–54.) Semanttisuutta parannetaan myös uusien elementtien attribuuttien avulla. Elementeille voidaan määrittää rooleja role-attribuutin avulla ja lomakkeen syötteille voidaan määrittää uusia tyyppejä, kuten email, search ja date. (Hogan 2011, 35–50, 91.) Uusien tyyppimäärityksien myötä mobiililaitteiden näppäimistöt osaavat tarjota syötteelle sopivaa näppäimistöä. Esimerkiksi number-tyypin syöte tuo mobiililaitteella automaattisesti esiin numeronäppäimistön. (Pilgrim 2011, 150–155.)

[9] kuten Apple Safari, Google Chrome, Mozilla Firefox, Internet Explorer ja Opera

HTML5 ei tuonut mukanaan pelkästään elementtikohtaisia muutoksia vaan myös uusia JavaScript-ohjelmointirajapintoja. Näiden rajapintojen avulla saadaan yhteys web-kehityksen rajojen ulkopuolelle. Erilaisia HTML5-rajapintoja on muun muassa History API, Geolocation API, Web Sockets API, Storage API ja Video sekä Audio API. (Pilgrim 2012, ix.)

3.10 JavaScript

JavaScript on oliopohjainen (object-oriented) dynaaminen komentosarjakieli (scripting language), joka on suunniteltu täydentämään sovelluksia kuten selaimia. JavaScriptin avulla pyritään toteuttamaan interaktiivisia sekä rikkaita internet-sovelluksia (Rich Internet Applications, RIA). (Mozilla 2012b.)

JavaScript suunniteltiin alun perin selaimien yhteiseksi alustasta riippumattomaksi standardikieleksi Netscapen aloitteesta. Nykyisin JavaScriptin suosion ja suorituskyvyn optimointien[10] myötä kieltä käytetään muuallakin kuin web-alustoissa. Lukuisat sovellukset ja ympäristöt, kuten NodeJS, Apache CouchDB ja MongoDB, pohjautuvat täysin JavaScriptiin. Näillä sovelluksilla ei ole mitään yhteyttä selaimiin. (Mikowski & Powell 2012, 4.)

Yleisimmin JavaScriptin avulla pyritään muokkaamaan DOM-rajapinnan kautta HTML-sivun rakennetta ja luomaan interaktiivisuutta selainsovellukseen. Interaktiivisuus saavutetaan mahdollisuudella vastata käyttäjän tapahtumiin kuten klikkauksiin. (Mozilla 2012b.) HTML:n muokkauksen lisäksi JavaScriptin avulla pyritään rikkaaseen interaktioon asiakkaan selaimen ja palvelimen välillä. Tätä interaktiota kutsutaan nimityksellä Ajax. (Bibeault & Katz 2008, 218.)

[10] Google V8 JavaScript moottori

3.10.1 Ajax

Ajax (Asynchronous JavaScript and XML) on yksi tärkeimmistä rikkaan selainsovelluksen mahdollistavista tekniikoista (Bibeault & Katz 2008, 218). Ajax mahdollistaa kommunikaation HTML-sivun ja palvelimen välillä. Täten jo valmiiksi ladattu HTML-sivu voi lähettää ja vastaanottaa tietoa palvelimelta eri muodoissa. Yleisimmin käytetyt tiedonsiirtokielet ovat JSON, HTML ja XML. (Bibeault & Katz 2008, 221.) Ajaxin avulla voidaan päivittää vain haluttu osa sivusta sen sijaan, että päivitettäisiin koko sivu. Asynkronisen kommunikoinnin ansiosta Ajax-kutsut eivät vaikuta muun JavaScriptin suoritusprosessiin. Asynkroniset kutsut suoritetaan taustalla, jolloin JavaScriptin ajo ei jää odottamaan palvelimen vastausta. Kun palvelin viimein palauttaa lähetettyä pyyntöä vastaavan vastauksen, se voidaan käsitellä asynkronisesti valmiiksi rekisteröidyssä funktiossa. Ideaalisessa Ajax-pohjaisessa ajattelussa sivua ei tarvitsisi ladata ensimmäisen kerran jälkeen lainkaan uudelleen. (Dashorst & Hillenius 2009, 239–241.)

3.10.2 JavaScript SPA

Single-page Application (SPA) -ajattelu syntyi jo 2000-luvun alussa Flash-teknologian ja Java applet -tekniikan myötä (Mikowski & Powell 2012, 3). JavaScriptin Ajax-ajattelu syntyi vasta vuonna 2005 Jesse James Garretin johdolla (Bibeault & Katz 2008, 218). Tätä ennen JavaScriptin rooli web-sovelluskehityksessä oli melko pieni ja sitä käytettiin yleisesti vain interaktiivisuuden luomiseen, kuten lomakkeiden validointiin ja pop-up-ilmoituksiin. Vuonna 2005 Ajax vihdoinkin mahdollisti JavaScript SPA -sovelluksen luomisen. Tätä ennen JavaScriptiä ei nähty vielä tarpeeksi kypsänä vaihtoehtona. Myös selaimien JavaScript toteutukset nähtiin puutteellisina. (Mikowski & Powell 2012, 3.)

JavaScriptin suorituskykykilpailu eri selaimien välillä ja HTML5-standardin mukana tulleet rajapinnat ovat johtaneet siihen, että on suositeltavinta käyttää JavaScriptia SPA-sovelluksen toteuttamiseksi. Kehittyneiden tekniikoiden ansiosta sovelluksessa on SPA-toteutuksessa nimensä mukaisesti vain yksi sivu, joka toimii pohjana sovellukselle. Ajax-tekniikan avulla voidaan luoda käyttäjälle vaikutelma useasta eri sivusta. SPA-toteutuksen myötä sovellus lataa vain kerran tarvittavat staattiset resurssit, kuten CSS-

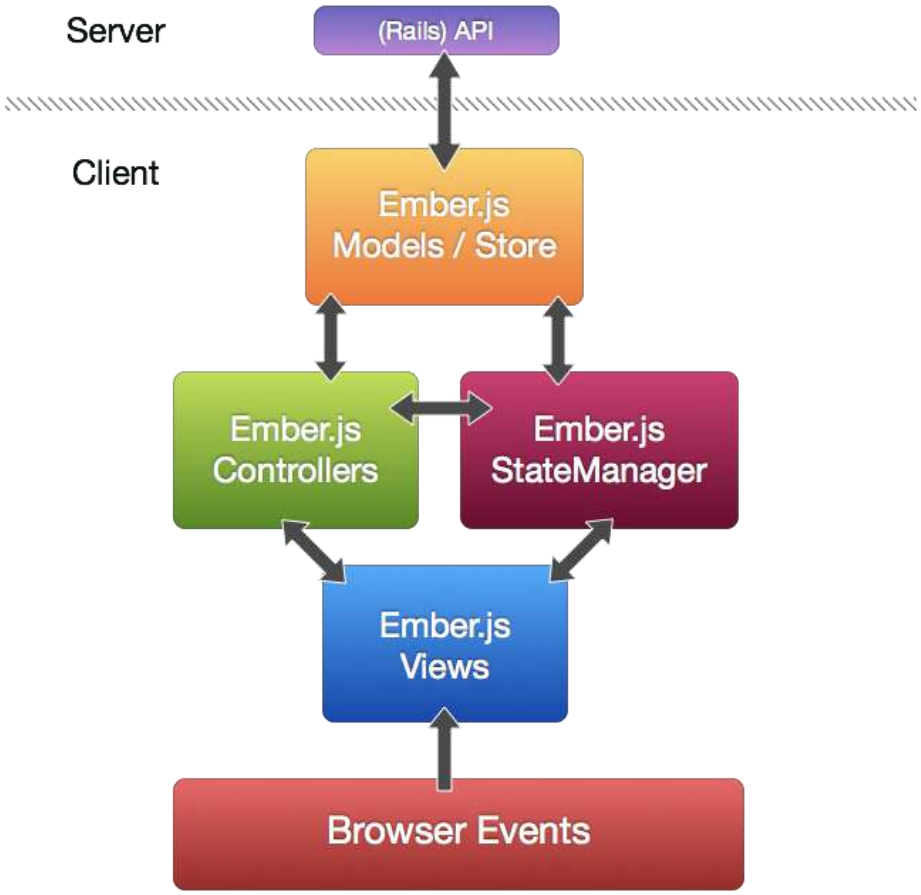
ja JavaScript-tiedostot. Palvelimelle kohdistuva rasitus pienenee huomattavasti ja sovellus muistuttaa työpöytäsovellusta. (Mikowski & Powell 2012, 2–4).

SPA-ajattelun suosio on kasvanut viimeisen kahden vuoden sisällä räjähdysmäisesti ja lukuiset JavaScript kirjastot kilpailevat uuden JavaScript MVC -teknologian markkinoista. (Osmani 2012a.)

3.10.3 JavaScript MV* -kehykset

JavaScriptin avulla toteutettu single-page sovellus muuttuu nopeasti yhä monimutkaisemmaksi ja täten syntyy tarve käyttäjäpuolen (client-side) arkkitehtoniselle ratkaisulle. Avuksi on kehitetty useita arkkitehtoniseen rakenteeseen keskittyviä kehyksiä, joita kutsutaan epävirallisesti JavaScript MV* -kehyksiksi. Ne tarjoavat modernin rakenteen interaktiivisten single-page sovelluksen toteuttamiseen. (Osmani 2012a.)

Käyttäjäpuolen JavaScript MVC (Model-View-Controller) -kehyksien arkkitehtuuri poikkeaa suurelta osin palvelinpuolen kehyksien MVC-arkkitehtuurista. Palvelinpuolella toimiva MVC-arkkitehtuuri pystyy kommunikoimaan käyttäjän kanssa ainoastaan selaimen HTTP-pyyntöjen ja vastauksien avulla. Käyttäjäpuolen JavaScript MVC puolestaan suoritetaan käyttäjän selaimessa, jolloin se pystyy suoraan kommunikoimaan käyttäjän käynnistämien tapahtumien kanssa (Kuvio 3). (EmberJS 2012.) MVC-ohjelmistoarkkitehtuuri jakaa JavaScript-arkkitehtuurin kolmeen itsenäiseen osaan. Kullakin osalla on oma vastuunsa ja roolinsa. (Osmani 2012b.)



Kuvio 3. EmberJS-kehyksen tarjoama arkkitehtoninen rakenne

Malli (Model) edustaa tietorakennetta ja liiketoimintalogiikkaa. Se vastaa kommunikoinnista palvelintason kanssa REST-rajapinnan kautta. Lisäksi JavaScipt MVC -kehyksille on ominaista, että ne hyödyntävät tarkastelija-mallia (Observer Pattern). Tarkastelija-mallin johdosta malli on ilmoitusvastuussa tarkastelijoille, mikäli sen tietoihin tulee muutoksia. Tarkastelija-malli auttaa pitämään tiedot jatkuvasti ajan tasalla. Tarkastelijana yleensä toimii Näkymä-taso. (Osmani 2012b.)

Näkymä (View) -taso vastaa käyttöliittymän templaattien luomisesta, päivittämistä ja näyttämisestä. Näkymille on ominaista kaksisuuntainen sidosyhteys (two-way databinding), joka mahdollistetaan tarkastelijoiden avulla. Näkymät toimivat tarkastelijoina, jotka reagoivat malliin tulleisiin muutoksiin. Tarkastelijan havaitsema muutos päivittyy automaattisesti tarkastelijaan liittyvään näkymään. Myös käyttäjän tekemä muutos näkymässä heijastuu sidosyhteyden ansiosta automaattisesti malliin.

Malli- ja Näkymä-tasot ovat tietoisia toisistaan tarkastelija-mallin kautta, mutta ne eivät ole koskaan suoranaisesti yhteydessä toisiinsa. (Osmani 2012b.)

Ohjain (Controller) käsittelee käyttäjän tapahtumat ja päivittää niiden pohjalta mallin tiedot. Ohjain pyrkii olemaan ottamatta kantaa näkymään, ja muutokset menevät automaattisesti kaksisuuntaisen sidosyhteyden kautta mallilta näkymälle. Koska ohjain ei ota kantaa näkymiin, siihen on helpompi soveltaa yksikkötestausta. (AngularJS 2012a; Osmani 2012b.)

JavaScript MVC -kehykset eroavat toisistaan rakenteeltaan, eikä niitä kaikkia voida pitää MVC-rakenteen mukaisina. Osa kehyksistä jakaa vastuut eri tasoille. BackboneJS-kehys käsittelee Ohjain-tason tehtäviä Näkymä-tasolla ja KnockoutJS-kehys (MVVM) ei sisällä Ohjain-tasoa, vaan se määrittää uuden tason nimeltä NäkymäMalli (ViewModel). Lukuisten eri MVC-variaatioiden vuoksi on parempi viitata single-page-kehyksiin nimellä JavaScript MV*. (Osmani 2012a.)

JavaScript MV* -kehykset eivät ainoastaan tarjoa MV*-rakennetta, vaan kuhunkin kehykseen on toteutettu tarpeellisiksi katsottuja ominaisuuksia. Yleisimmin toteutettuja ominaisuuksia ovat reititys (routing), selainhistorian säilyttäminen (Hashtag tai HTML5 History API), kaksisuuntainen sidosyhteys (two-way databinding), templaatit, XSS (Cross-site Scripting) -esto, integraatio REST-rajapintojen kanssa sekä lukuisat muut tarpeellisiksi todetut ominaisuudet. (Osmani 2012a.)

Sovellusarkkitehtuurin uusi ulottuvuus on mahdollistanut uuden markkinaraon avoimien lähdekoodisovelluksien joukossa. Kilpailu markkinoista on jo kovaa, vaikka suuntaus uuteen asiakaspuolen arkkitehtuuriin on vasta alussa. Tällä hetkellä JavaScript MV* -kehyksien määrä on jatkuvassa kasvussa ja tarkkaa arviota MV*-kehyksien lukumäärästä ei tiedetä. Varmuudella niitä on ainakin yli 30. Tunnetuimpia JavaScript MV* -kehyksiä ovat BackboneJS, EmberJS, AngularJS, KnockoutJS, MeteorJS ja SpineJS. (Osmani 2012a.)

3.10.4 AngularJS

AngularJS on Googlen kehittämä JavaScript MV* -kehys. Se erottuu muista MV*-kehyksistä templaattirakenteeltaan. Kehys hyödyntää HTML-pohjaisia templaatteja, kun taas useimmat JavaScriptit MV* -kehykset hyödyntävät String-pohjaista templaattirakennetta. (AngularJS 2012b.)

AngularJS-kehyksen templaatit hyödyntävät niin kutsuttuja direktiivejä, jotka voivat olla templaatilla Angular-elementtejä tai -attribuutteja. HTML-templaattirakenteen ansiosta templaatit saadaan käännettyä esitystä varten DOM-elementeiksi selaimella. String-pohjaisissa templaateissa DOM-rajapinnan elementtien luominen täytyy suorittaa JavaScriptin avulla. (AngularJS 2012c.)

AngularJS tarjoaa muiden JavaScript MV* -kehyksien tapaan lukuisia arkkitehtonisia ratkaisuja CRUD-sovelluksien toteuttamiseen. Se sisältää valmiina muun muassa ominaisuuksia, kuten data-binding, templaatit ja direktiivit, lomakkeen validointi, reititys, deep-linking, uudelleen käytettävät komponentit ja palvelinpuolen kehyksistä tutuksi tullut Dependency Injection -tekniikan. Lisäksi AngularJS on keskittynyt vahvasti yksikkötesteihin. (AngularJS 2012d.)

JavaScript on dynaaminen kieli, joka ei mahdollista kääntäjien tuomia hyötyjä, kuten virheiden ilmoitusta. Tästä johtuen yksikkötestaus on tärkeässä asemassa. Angular-kehyksellä on mahdollista hyödyntää kolmansien osapuolten yksikkötestauskirjastoja, kuten JasmineJS, TestacularJS ja PhantomJS. (AngularJS 2012e.)

3.11 Testaus

3.11.1 Käytettävyystestaus

Käytettävyystestauksen järjestäminen on suunniteltava tarkkaan etukäteen. On mietittävä, miten testitilanteet saadaan sujumaan mahdollisimman hyvin, jotta kerätyn materiaalin perusteella löydetään käytettävyydessä ilmenevät ongelmat ja pystytään laatimaan hyvä testiraportti. (Koskinen 2005, 190.) Huonosti suunniteltu testaus saattaa

tuhlata turhaan yrityksen rahoja ja resursseja. Ennen testaamista tulisi vähintään miettiä, minkälaisia osallistujia halutaan ottaa mukaan, kuinka monta osallistujaa tarvitaan, aiotaanko tuloksia verrata eri ryhmien välillä ja tarvitseeko testitehtävien järjestystä muuttaa. (Tullis & Albert 2008, 15–16.)

Steve Krug puolestaan ehdottaa ”Lost-our-lease”-testausmenetelmää. Menetelmän pääperiaatteena on, että testaaminen on erittäin tärkeää ja sitä tulisi tehdä, vaikka yrityksessä kuinka luultaisiin, ettei sille ole resursseja tai osaamista. Käytettävyystestauksessa tulisi olla enintään kolme tai neljä osallistujaa kullakin testauskierroksella. Näkemys perustuu siihen, että yksikin testikäyttäjä on reilusti parempi kuin ei yhtään ja että testausta tulisi suorittaa mahdollisimman usein ottaen huomioon yritykselle sopivan budjetin. Krug myös toteaa, että testaukseen osallistujaksi kelpaa lähes kuka tahansa, joka käyttää internetiä. Huomattava asia on, ettei ”Lost-our-lease”-menetelmä suosittele tieteellistä menetelmää tulosten analysoimiselle. Krug toteaa, että suurin osa saaduista tuloksista on niin ymmärrettävää, että lähes kuka tahansa osaa tulkita niitä. (Krug 2006, 134–145.) Jakob Nielsen (2000) toteaa, että testausta tulisi suorittaa enintään viidellä henkilöllä, koska tämän jälkeen suurin osa testin perusteella saadusta tiedosta on toistoa ja yrityksen budjetin tuhlausta.

Käytettävyystestauksessa tulisi keskittyä käyttäjään, ei tuotteeseen. Siten pystytään ymmärtämään, mikä käyttäjien mielestä toimii, mikä tuntuu huonolta tai ihmetyttää tai saa heidät hermostumaan. (Barnum, Carol M. 2010, 10.) Käytettävyyttä voidaan mitata käytettävyysmittareilla (usability metrics). Mittareita voi olla monia ja ne vaihtelevat tarpeen mukaan. Mittarien tulee kuitenkin aina olla määrällisiä tai vähintään muutettavissa numeraalisiksi. Käytettävyyden kohdalla voidaan mitata esimerkiksi työtehtävän onnistumista, käyttäjän tyytyväisyyttä tai virhetilanteiden määrää. (Tullis & Albert 2008, 7–8.)

Käytettävyystestauksessa on yleisesti nähtävissä kahdeksan askelta. Ensiksi suunnitellaan testi ja siihen kuuluvat tehtävät. Toiseksi kerätään materiaalit testausta varten. Kolmanneksi varmistetaan paikka testin pitämistä varten. Neljänneksi suoritetaan pilottitesti esimerkiksi yrityksen muilla työntekijöillä. Viidenneksi hankitaan

osallistujat testiä varten. Kuudenneksi pidetään testit. Seitsemänneksi analysoidaan testistä saadut tulokset. Lopuksi tehdään muutokset saatujen tulosten pohjalta ja toistetaan testi, jotta nähdään onko ongelmat saatu ratkaistua. (Brinck ym. 2002, 423.)

3.11.2 Hyväksymistestaus

Sovellustestauksessa on nähtävissä yleisesti neljä tasoa: yksikkötestaus (unit testing), integraatiotestaus (integration testing), järjestelmätestaus (system testing) ja hyväksymistestaus (acceptance testing) (Singh 2011, 368). Hyväksymistestausta tehdään, kun sovellus tuotetaan yksittäiselle tunnetulle asiakkaalle. Asiakas on mukana hyväksymistesteissä. (Singh 2011, 22.) Hyväksymistestien tarkoituksena on osoittaa asiakkaalle, että tuote toimii ja asiakas voi hyväksyä sen (Singh 2011, 373). Hyväksymistestit luodaan käyttäjätarinoiden pohjalta ja ne tulisi automatisoida, jotta niitä voidaan suorittaa usein. (Extreme Programming 1999)

Selenium-testausympäristössä on tarkoituksena tuottaa koodia, joka käynnistää selaimen ja pyrkii sen jälkeen seuraamaan sille annettuja käskyjä. Koodia ei tarvitse välttämättä kirjoittaa. Firefox-selaimeen liitettävän lisäosan avulla ohjelman käyttäjä voi luoda koodin klikkaamalla sivuilla olevia kohteita. Tämän jälkeen lisäosa kääntää klikkaukset koodiksi, jonka voi ajaa nähdäkseen, menevätkö testit läpi. (SeleniumHQ 2012a.) Koodia voi myös kirjoittaa suoraan sovelluksen omalla ohjelmointiympäristöllä (IDE eli Integrated Development Environment) tai useilla tunnetuilla ohjelmointikielillä, kuten C#, Java, PHP tai Python. (SeleniumHQ 2012b.)

Cucumber-testaus perustuu Behaviour Driven Development -ideologiaan, jossa ohjelmistokehitys toteutetaan sekä liiketoiminnallisesta että teknillisestä näkökulmasta. Cucumber-testit kirjoitetaan kolmella tasolla: liiketoimintalogiikka selkokielellä (behaviour, käyttötapaukset), yksittäiset askeleet Gherkin-ohjelmointikielellä (step definitions) ja askeleiden toimintaa ohjaava koodi Ruby-ohjelmointikielellä hyödyntäen Capybara-kirjastoa. Käytännössä Cucumber käynnistää selaimen, navigoi tietylle sivulle ja pyrkii sen jälkeen seuraamaan sille ohjelmoituja käskyjä. (Cucumber 2012.)

4 Prototyypin alustavien ominaisuuksien kartoitus

”Information about users should come as early as possible in the design process.” (Brinck ym. 2002, 14).

Ennen kuin mobiilioptimoidun käyttöliittymän suunnittelu aloitettiin, haluttiin saada tietoa siitä, mitkä ominaisuudet ovat käyttäjille tärkeitä ja miten käyttäjät toimivat nykyisessä käyttöliittymässä. Tiedon hankintaa varten päätettiin luoda sähköinen jäsennelty kysely, joka lähetettäisiin asiakkaille. Tiedon hankinnan keinoina mietittiin myös haastattelua.

Jäsennelty kysely valittiin selvitysmenetelmäksi, koska sille oli ehdottomasti parhaat resurssit ja puitteet. Haastattelulla olisi ollut mahdollista saada tarkempia tuloksia, mutta haastattelutapaamisten järjestäminen ja pitäminen sekä palautteen purkaminen ja analysointi olisivat kaikki vaatineet liikaa aikaa, jolloin kohdejoukko ja tulokset olisivat jääneet huomattavasti pienemmiksi.

Käyttäjäkysely kirjoitettiin ensin puhtaana tekstinä tekstitiedostoon. Kun kysymyksiin ja niiden esitystapaan oltiin tyytyväisiä, ne siirrettiin Webropol-kyselyohjelmaan, jolla ne lähetettiin edelleen asiakkaille. Linkki kyselyyn lähetettiin kaikille yrityksen asiakkaiden yhteyshenkilöille sähköpostin välityksellä. Sähköposteja lähetettiin 850 osoitteeseen. Palautuneiden sähköpostien perusteella kysely saavutti noin 700 osoitetta. Vastauksia saatiin 94 kappaletta. Vastausprosentti oli siis noin 13,4 %. Vastausten määrä todettiin kattavaksi analysointia varten.

4.1 Käyttäjäkyselyn suunnittelu

Käyttäjäkyselyssä pyrittiin ensisijaisesti saamaan tietoa siitä, miten tärkeitä nykyisen toimintajärjestelmän toiminnallisuudet ja ominaisuudet ovat ja millaisia toimintatapoja käyttäjillä on nykyisessä käyttöliittymässä. Nykyisen käyttöliittymän käytettävyyttä ei pyritty selvittämään, koska mobiilioptimoituun käyttöliittymään oli tarkoitus suunnitella aivan uusi ulkoasu ja käyttöliittymä. Kysymyksissä keskityttiin kartoittamaan

ominaisuuksia ja tapoja, joita oli jo alustavasti mietitty käytettäväksi uudessa käyttöliittymässä.

Prototyypin viralliseksi kohderyhmäksi oli valittu IMS-toimintajärjestelmän katsojakäyttäjät. Jo ennen kyselyn toteuttamista pystyttiin kuitenkin toteamaan, ettei tämän kohderyhmän tavoittaminen olisi mahdollista projektiryhmälle tarjolla olevilla tiedoilla ja resursseilla. Suurin osa vastaajista tulisi todennäköisesti olemaan toimintajärjestelmän pääkäyttäjiä. Kyselyn kysymyksiä muokattiin hieman tämän vuoksi. Kokonaisuudessaan kysely käsitteli asioita, joiden voitiin olettaa koskettavan yhtälailla kaikkia toimintajärjestelmän käyttäjiä.

Suunniteltaessa internetsivuja mobiilikäyttöön on tärkeää miettiä, mitä asiakkaat tekevät ja minkälaisissa tilanteissa he ovat käyttäessään sovellusta mobiililaitteella. (Layon 2012, 3; McWherter & Gowell 2012, 16.) Tätä asiaa ei kuitenkaan painotettu kyselyssä millään tavalla, koska mobiilioptimoitu käyttöliittymä haluttiin toistaiseksi pitää salassa kilpailuedun säilyttämiseksi. Ajateltiin myös, että myöhemmin pidettävässä käytettävyystestauksessa on mahdollisuus kysyä osallistujilta heidän näkemyksiään tilanteista, joissa mobiilikäyttö tulee tarpeelliseksi.

Yrityksen sisällä oli aikaisempina vuosina lähetetty käyttäjäkysely asiakkaille, mutta vastauksia ei tuolloin saatu kuin muutamia. Koska aiempi kysely epäonnistui, pyrittiin nyt tehtävässä käyttäjäkyselyssä löytämään parempi tasapaino yksinkertaisuuden ja tulosten saamisen välillä. Kyselyssä painotettiin valintakysymyksiä, joihin on nopea vastata. Vasta näiden jälkeen esitettiin vaihtoehtoisia kysymyksiä, joihin pystyi halutessaan vastaamaan omin sanoin. Myös kysymysten määrä pyrittiin pitämään suhteellisen alhaisena ja kyselyyn valittiin vain tarpeellisimpana nähdyt kysymykset. Kysymyksiä tuli yhteensä 23 kappaletta.

Houkutukseksi kyselyyn lisättiin viisi kappaletta palkintoja. Palkinnonjakoon osallistumisen edellytyksenä oli vastata kaikkiin kysymyksiin ja antaa kyselyn lopussa sähköpostiosoite tai puhelinnumero. Palkintojen lupaamisen vaarana on usein se, että osa vastanneista saattaa vastata kyselyyn vain voittaakseen palkinnon eikä ole

kiinnostunut antamaan osallistumisellaan rakentavaa palautetta. Mainittua ongelmaa on todennäköisesti mahdotonta korjata poistamatta palkinnon jakoa kokonaan, mutta ongelmaa pyrittiin lieventämään palkintojen tyypillä sekä sillä, että osallistujien tulee vastata kaikkiin kysymyksiin osallistuakseen kilpailuun. Palkintoina oli viisi kappaletta ilmaisia osallistumisia yrityksen[11] laadunhallinnan koulutuksiin. Ennen kyselyn tekemistä otettiin selvää palkintojen jakoon liittyvistä lakisääteisistä seikoista.

4.2 Käyttäjäkyselyn analysointi

Käyttäjäkyselyn tuloksia analysoitiin pääasiassa määrällisesti. Saadut vastaukset oli helppo muuntaa raportiksi Webropol-kyselyohjelman työkaluilla. Raportin pohjalta pystyttiin tämän jälkeen analysoimaan saadut vastaukset. Analysoinnissa tutkittiin ensisijaisesti suosituimpia vastauksia ja käyttäjien kirjoittamia kommentteja. Näiden jälkeen pyrittiin miettimään, vaikuttiko jokin asia, kuten vastaajien rooli toimintajärjestelmässä, olennaisesti vastauksiin, ja tarvitseeko tämän takia tuloksia tulkita jollakin tavalla. Lopullisten päätösten osalta mietittiin vielä, onko päätöksiä mahdollista tai kannattavaa toteuttaa ja miten ne eroavat jo mietityistä ratkaisuista. Käyttäjäkyselyn kysymysten perustelu on luettavissa luottamuksellisesta liitteestä 2 Kyselyn kysymysten avaus ja kyselyn tulosten analysointi luottamuksellisesta liitteestä 3 Kyselyn tulokset ja analysointi.

Analysoinnissa jouduttiin jonkin verran tulkitsemaan saatuja vastauksia. Osa kysymyksistä käsitteli ominaisuuksia, jotka on lisätty toimintajärjestelmään vasta vastikään, eivätkä kaikki asiakkaat olleet selvästikään tietoisia niistä. Kuten oletettiin, suurin osa vastaajista mainitsi olevansa yrityksessä toimintajärjestelmän pääkäyttäjiä. Tähän oli osattu onneksi varautua jossakin määrin etukäteen, joten kyselyn tuloksia oli mahdollista tulkita tämän tiedon perusteella. Kyselyllä saatujen tulosten perusteella pystyttiin aloittamaan käyttöliittymän suunnittelu ja priorisoimaan siihen kuuluvia ominaisuuksia.

[11] Koulutukset järjestää IMS Business Solutions Oy:n emoyhtiö Qualitas Fennica

5 Prototyypin vaatimukset ja määrittely

”The requirements specification is the keystone of all software documentation. It is the statement of what the software system is to provide.” (Horch 2003, 212)

Yleensä sovelluskehittäjät saavat vaatimusmäärityksetvalmiina suoraan kehitettävän sovelluksen tilaajalta. Tässä projektissa projektiryhmä kuitenkin vastasi itse määritysten ja vaatimusten laatimisesta sekä niiden hyväksyttämisestä tilaajalla. Vaatimusmääritys perustuu osittain valmiiseen sovellukseen ja osittain projektissa valmistuvaan prototyyppiin. Valmista sovellusta koskeva vaatimusmääritys käsittää tuotteen taustan, tavoitteet ja ympäristön. Prototyyppiä koskeva vaatimusmääritys käsittää automatisoinnin ja käyttötapaukset. Käyttötapaukset on kuvattu myös Cucumber-testiympäristön vaatimalla liiketoimintalogiikkaa edustavalla tyylillä, jonka yksittäiset askeleet määritetään Gherkin-ohjelmointikielellä. Käyttötapaukset ottavat huomioon vain prototyyppiin luotavat toiminnallisuudet. Sovelluksen vaatimukset ja määrittely on kuvattu tarkemmin luottamuksellisissa liitteissä 4 ja 5:
Vaatimusmäärityksmääritysdokumentti osa 1 ja 2.

6 Prototyypin suunnittelu

”It presents what the designers believe to be the correct solution and response to the approved requirements.” (Horch 2003, 214.)

Tässä luvussa esitellään mobiilioptimoidun käyttöliittymän prototyypin suunnittelun eri vaiheet. Luku on jaettu projektin alussa suoritettuun pohdintaan ja taustaselvitykseen, sovelluksen tyypin valitsemiseen sekä alustavan ulkoasun suunnitteluun. IMS-toiminnanhallintajärjestelmän nykyisessä käyttöliittymässä käytetään vanhentunutta HTML Frameset -tekniikkaa (frames), mikä haittaa olennaisesti sivujen muokattavuutta ja sen saattamista mobiilioptimoiduksi. Kehysten poistaminen käyttöliittymästä on todettu yrityksen sisällä jo kauemman aikaa niin työlääksi projektiksi, ettei asiaan aiota puuttua. Tämän perusteella todettiin, että täysin uusi käyttöliittymä on paras keino toteuttaa mobiilioptimointi.

6.1 Sovelluksen tyypin valinta

Ensimmäisiä valintoja projektissa oli sovelluksen tyypin määrittäminen. Varteenotettavia vaihtoehtoja oli kaksi: mobiilisovellus ja selainsovellus. Sekä mobiilisovelluksessa että selainsovelluksessa on omat vahvuutensa ja heikkoutensa. Toiseen näistä päädyttiin hyvinkin pikaisesti. Jo ennen opinnäytetyöprojektin alkua projektiryhmä toteutti pienen mobiilisovellusdemon Android-käyttöjärjestelmälle. Demo rakennettiin nopeasti, noin viikon aikana, ja vähäisen itseopiskelun pohjalta. Demo kommunikoi toimintajärjestelmän kanssa ja sillä pystyi tarkastelemaan järjestelmän sisältöä. Demon rakentamisen myötä saatiin tarkempaa näkemystä mobiilisovelluksen toteuttamisesta ja sen vaatimista resursseista. Yrityksessä ei ollut aikaisempaa kokemusta tämänkaltaisten sovellusten toteuttamisesta, joten demo koettiin erittäin tärkeäksi uuden käyttöliittymätyypin valinnan kannalta. Samalla demo toimi rajaavana tekijänä opinnäytetyön aiheen valinnassa.

Mobiilisovelluksen huonona puolena on, että sovellus pitää toteuttaa kullekin mobiilikäyttöjärjestelmälle erikseen, ja sovelluksen kehitystä varten tarvitaan kunkin

valitun käyttöjärjestelmän osaamista. Tämän ongelman ratkaisisi Phonegap-palvelu, jolla pystyy luomaan mobiilisovelluksia pelkän HTML-, CSS- ja JavaScript-kielien avulla useimmille tunnetuille mobiilialustoille. Yritys ei kuitenkaan halunnut olla riippuvainen kolmannen osapuolen työkalusta, joten palvelun käyttö ratkaisuna jouduttiin hylkäämään.

Jotkut järjestelmät vaativat, että sovellus laitetaan jakeluun käyttöjärjestelmän omalle jakelukanavalle[12]. Näillä jakelukanavilla voi olla alustasta riippuen omat kriteerinsä sovelluksille. Apple on esimerkiksi julkaissut oman *iOS Human Interface Guidelines*[13] -ohjeistuksen. Näiden kriteerien hyvänä puolena on, että ne pakottavat sovelluskehittäjät tekemään sovelluksistaan korkealaatuisia. Toisaalta ne saattavat myös rajoittaa sovelluksen ulkoasua, sisältöä ja lisäksi vaativat ylimääräisiä resursseja kehitysryhmältä.

Jo ennen projektia suoritetun Android-mobiilisovellusdemon rakentamisen yhteydessä voitiin todeta, ettei yrityksellä riitä resursseja rakentamaan tai ylläpitämään lukuisten eri mobiilikäyttöjärjestelmien sovelluksia. Selainsovellus nähtiin ehdottomasti luonnollisempana vaihtoehtona sovelluksen tyypille. Aiemmin mainittujen etujen lisäksi ratkaisevana tekijänä oli, että nykyinen käyttöliittymä on selainsovellus. Tässä muodossa sovellus on helppo yhdistää nykyisen käyttöliittymän rinnalle. Yritykselle on myös tärkeää, että uutta käyttöliittymää voi halutessaan käyttää myös tietokoneella, mikä ei ole mahdollista mobiilisovelluksilla.

6.2 Prototyyppiin suunnitellut toiminnallisuudet

Käyttöliittymässä pyrittiin tarjoamaan käyttäjälle olennainen osa toimintajärjestelmän toiminnoista. Näytön koon pienentyessä vähennetään käyttäjän kerralla näkemää sisältöä. Mukautuva websuunnittelu -tekniikkaa tukevat sivut toisinaan piilottavat

[12] iOS-käyttöliittymällä Apple App Store, Android-käyttöliittymällä Google Play ja Windows Phone -käyttöliittymällä Windows Phone Marketplace
[13] Luettavissa: http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/Introduction/Introduction.html

sisältöä sitä mukaa, kun näytön tai selaimen koko pienenee. Koska kyseessä on kuitenkin sovellus, ei internetsivu, kaikki toiminnot pyritään pitämään käyttäjän saatavilla. Näytön koon pienentyessä toiminnot siirretään omiksi, vähemmän tilaa vieviksi kokonaisuuksikseen. Koska kaikkien toimintojen tulisi olla saatavilla, pyrittiin toimintojen määrä rajaamaan järkevän suuruiseksi ja näyttämään käyttäjälle ensisijaisesti tärkeimmät tai käytetyimmät ominaisuudet.

Toimintojen valinta aloitettiin tekemällä lista kaikista IMS-toimintajärjestelmän Dokumentit-osiossa[14] olevista käyttäjäkohtaisista toiminnallisuuksista. Sovellus on suunnattu järjestelmän katsojakäyttäjille, joten listasta yliviivattiin aluksi sellaiset toiminnallisuudet, jotka eivät koske kyseistä käyttäjäryhmää. Tämän jälkeen priorisoitiin jäljelle jääneet toiminnallisuudet, ja näin saatiin alustava lista prototyyppiin valituista toiminnallisuuksista. Listan järjestystä ja sisältöä muokattiin vielä käyttäjäkyselystä saatujen vastausten perusteella. Listassa esiintyvät toiminnallisuudet toimivat myös pohjana tuotteen kehitysjonon käyttäjätarinoille. Toiminnallisuuksiin kuuluvat muun muassa navigointipuun selaaminen, dokumentin katselu sekä suosikkien hallinnointi.

6.3 Prototyypin alustava ulkoasu

Opinnäytetyöprojektin yksi olennaisimmista osista oli sovelluksen käytettävyyden optimointi, ei keskittyminen ulkoasun hiomiseen. Ulkoasu on kuitenkin olennainen osa käytettävyyttä, joten sen rakennetta suunniteltiin ja se pyrittiin toteuttamaan käytössä olevan ajan puitteissa.

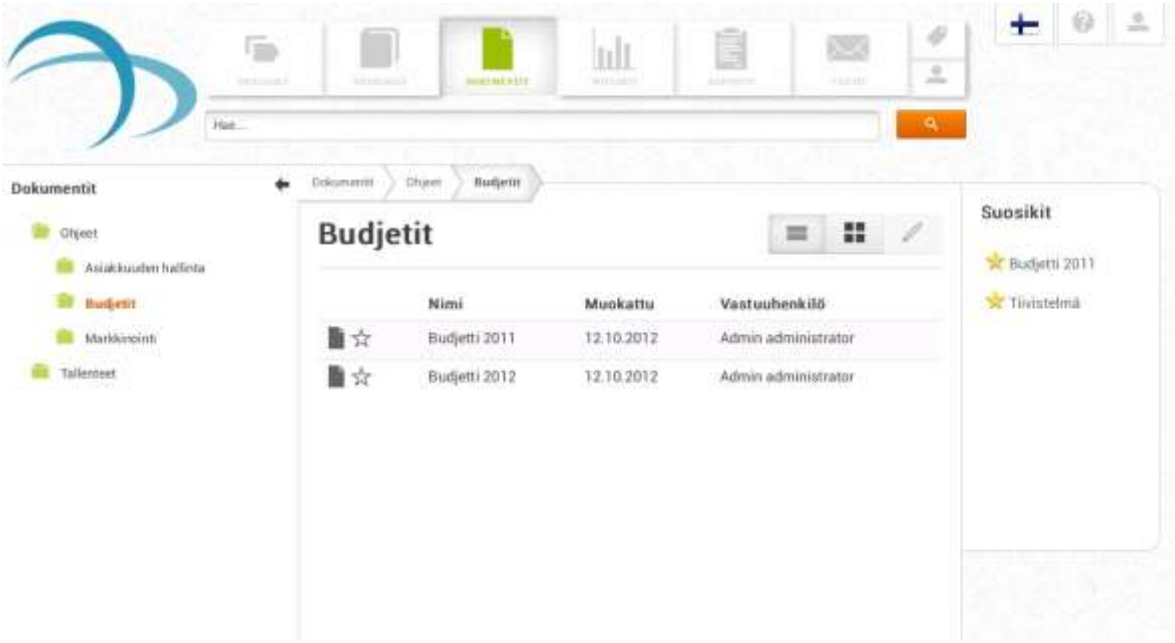
6.4 Grey boxing ulkoasun suunnittelun apuvälineenä

Grey boxing on suunnittelutapa, jossa keskitytään säilyttämään käyttöliittymän ulkoasu mahdollisimman yksinkertaisena ja nimensä mukaisesti harmaana ja valkoisena. Pääpaino on käyttöliittymän rakenteessa ja hierarkiassa. (Santa Maria 2004.) Tämän suunnittelutavan ansiosta on helpompi keskittyä sivujen käytettävyyteen. Se takaa myös

[14] Prototyypin toiminnallisuudet on rajattu koskemaan katsojakäyttäjäryhmää ja toimintajärjestelmän Dokumentit-osiota.

sen, ettei käyttöliittymän ulkoasun suunnittelussa hukata turhan paljon resursseja, jos ulkoasua joudutaan muuttamaan radikaalisti.

Grey boxing -suunnittelutapa helpotti huomattavasti käyttöliittymän suunnittelua opinnäytetyössä. Projektiryhmä koki miellyttäväksi käsitellä käyttöliittymässä olevia elementtejä ja rakenteita harmaansävyisinä komponentteina. Toinen projektiryhmän jäsenistä on värisokea, joten värimaailman huomioonottaminen olisi todennäköisesti kuluttanut ylimääräisiä resursseja.



Kuvio 4. Grey boxing -menetelmän mukainen ulkoasu

6.5 Piirustukset ja rautalankamallit

Ulkoasun ja käytettävyyden suunnittelu vaativat paljon pohjatyötä. Suunnitteluun tarvitaan kattava mielikuva kokonaisuudesta. Alustavaa käyttöliittymän rakennetta suunniteltiin piirtämällä runsaasti kuvia paperille ja tussitaululle. Taululle piirretyissä luonnoksissa pyrittiin tekemään versiointia, eli kuvia mietittiin ja niihin tehtiin pieniä parannuksia.

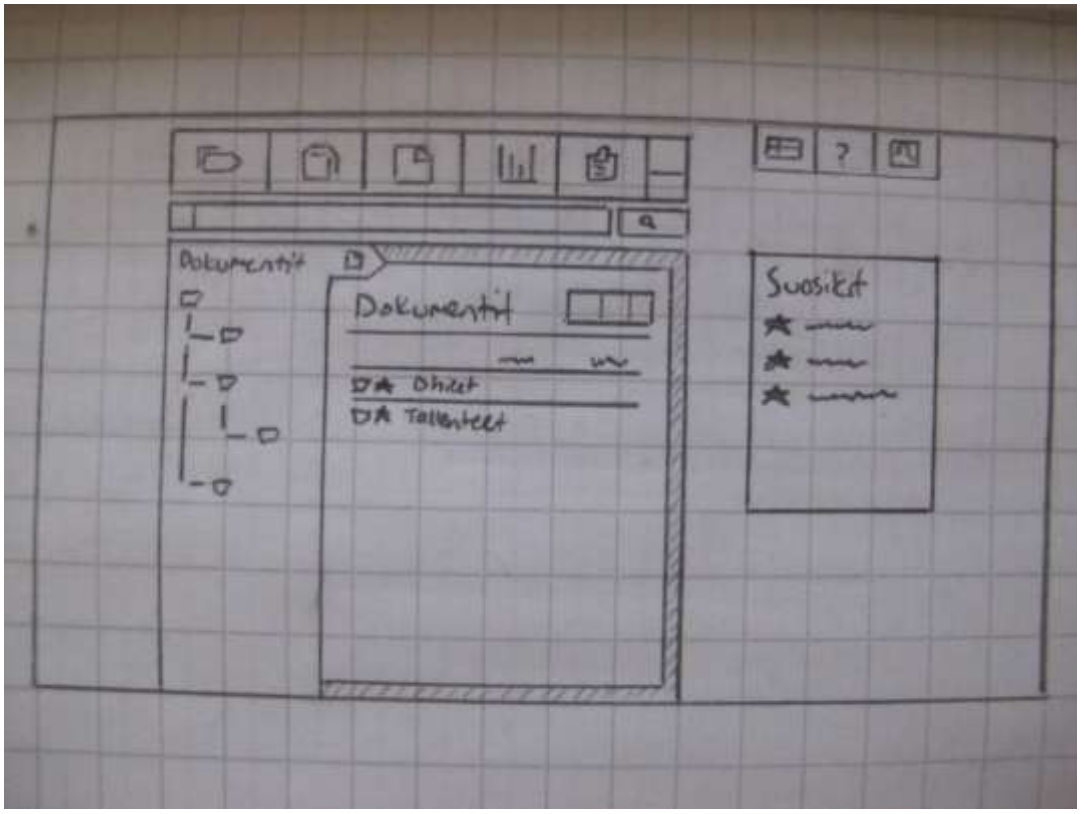
Piirustukset (sketching) ja rautalankamallit (wireframe) ovat kaksi eri asiaa. Piirustukset toimivat pikaisina, karkeina kokeiluina, joista suurin osa hylätään. Kun piirustuksiin

ollaan tyytyväisiä, niiden pohjalta voidaan lähteä tekemään tarkempia rautalankamalleja tai siirtyä suoraan luomaan vuorovaikutteista prototyyppiä. (Boag ym. 2011, 59.)

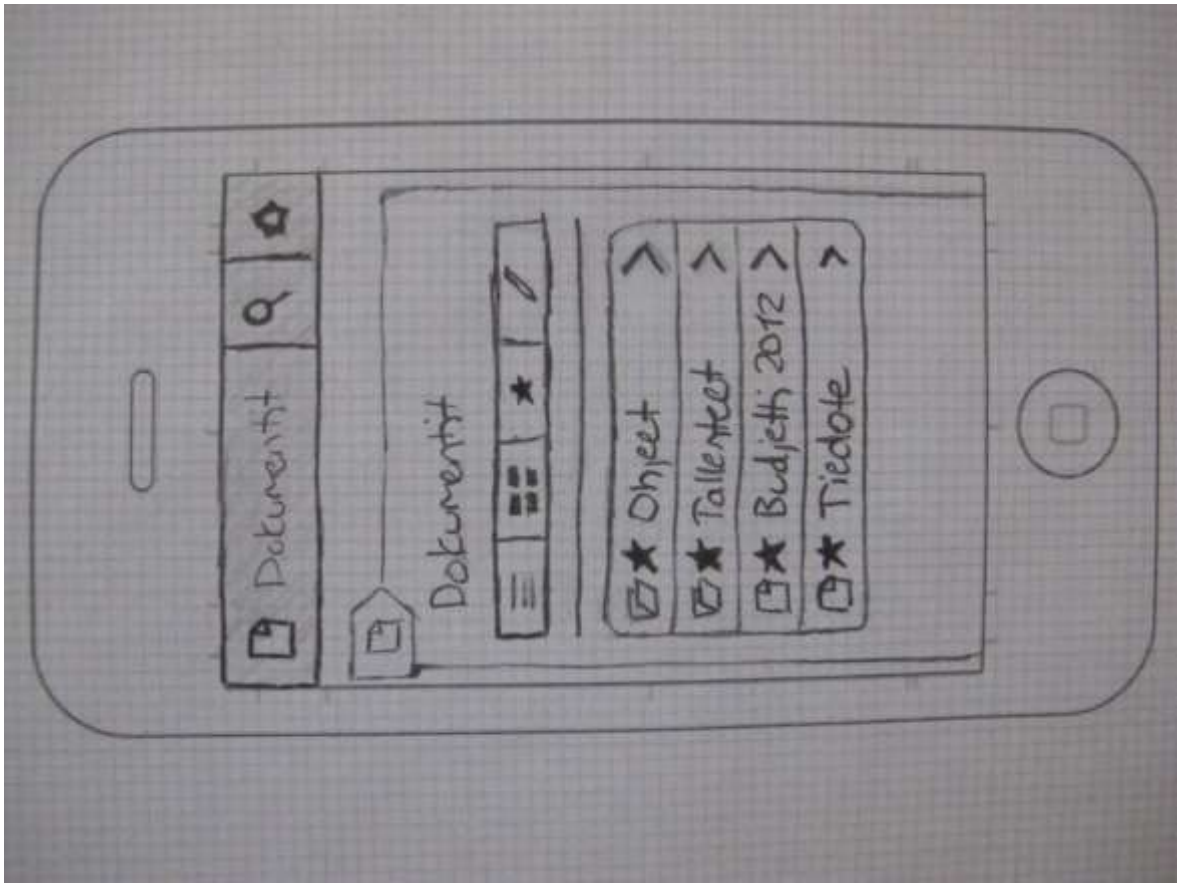
Rautalankamalli on sähköisessä muodossa oleva piirustus. Tarkoituksena on tarkentaa paperille piirrettyjä piirustuksia ja saada ne helpommin muokattavaan muotoon. Piirustuksia ja rautalankamalleja ei ole kuitenkaan tarkoitus päivittää loputtomiin, vaan käytäntönä on siirtyä nopeasti ensimmäisiin prototyyppeihin. (Boag ym. 2011, 59.)

Saatavilla on lukuisia rautalankamallien piirtotyökaluja, kuten Mockflow, Balsamiq ja ProtoShare. Eri ohjelmat ovat erikoistuneet eri tehtäviin. Esimerkiksi ProtoShare-ohjelmaa mainostetaan helppona ja yksinkertaisena mobiili- ja mobiilisovellusrautalankamallien piirtämistyökaluna. Osa ohjelmista on kuitenkin maksullisia, eikä projektiryhmällä ollut kokemusta yhdestäkään näistä ohjelmista.

Rautalankamallien piirtämisessä päätettiin käyttää työkaluna Microsoft Visio -ohjelmaa. Se osoittautui riittävän hyväksi työkaluksi, se oli jo valmiiksi asennettuna projektiryhmän tietokoneille. Projektiryhmällä oli myös kokemusta sen käytöstä. Rautalankamallit saatiin valmiiksi aikataulun mukaisesti projektin alkupuolella ja niiden avulla pystyttiin miettimään ja muokkaamaan käyttöliittymän rakennetta. Käyttöliittymän ulkoasu ja rakenne muuttuivat vielä projektin puolessa välissä käytettävyystestauksen myötä. Rautalankamalleihin päätettiin olla päivittämättä uusia muutoksia, koska nähtiin, että ne olivat toimineet tarvittavana apuna jo projektin alussa.



Kuvio 5. Piirustus käyttöliittymän ulkoasusta tietokoneen näytöllä tai tabletin näytöllä



Kuvio 6. Piirustus käyttöliittymän älypuhelimessa

7 Prototyypin toteutus

”Think how easily an application can be written using that data. That is the power of web services.” (McWherter & Gowell 2012, 38)

Projektin tarkoituksena ei ollut ainoastaan luoda uutta käyttöliittymää toimintajärjestelmälle, vaan samalla uusia yrityksen sovellusarkkitehtonisia ratkaisuja ja käytäntöjä. Käyttöliittymän uudistamisessa hyödynnettiin uusimpia teknologisia ratkaisuja, ja täten myös kokonaisarkkitehtuuri vaati muutoksia.

Projekti oli teknisiltä vaatimuksiltaan ja toteutukseltaan yksi suuri oppimisprosessi. Vain osa projektissa hyödynnetyistä tekniikoista oli projektiryhmälle aikaisemmin tuttua ja tiedossa projektin alussa. Projektin edetessä projektiryhmä sai vähitellen paremman käsityksen toteutettavan sovelluksen arkkitehtonisista vaatimuksista. Projektin aikana tutkittiin aktiivisesti tämän hetken trendejä ja parhaita käytäntöjä arkkitehtoniselle ratkaisulle.

7.1 Demot

Projektin tekninen toteutus aloitettiin tekemällä demoja. Ensimmäistä demoa lähdettiin toteuttamaan hyödyntäen ainoastaan HTML5-, CSS3- ja JavaScript-tekniikoita. Lisäksi näiden tekniikoiden pohjaksi valittiin HTML5 Boilerplate -templaatti. Vaihtoehtoiseksi templaattiratkaisuksi mietittiin muun muassa Twitter Bootstrap ja HTML KickStart -kehyksiä. Nämä kehykset kuitenkin eroavat tarkoitukseltaan paljon HTML5 Boilerplate -templaatista. Projektiryhmä päätti HTML5 Boilerplate -templaatin lisäksi hyödyntää muutamia Twitter Bootstrap -kehyksen sisältämiä ominaisuuksia ja räätälöidä niitä.

Ensimmäinen demo toteutettiin grey boxing -tekniikkaa hyödyntäen, joten sen tarkoituksena ei ollut tutkia ulkoasua vaan käyttöliittymän käytettävyyttä. Lisäksi demolla ei myöskään haluttu toteuttaa täysin toimivaa palvelinpuolta hyödyntävää ratkaisua, vaan tutkia nopeasti suunniteltua käytettävyyttä. Demossa ei keskitytty

havaittuihin pieniin ohjelmointivirheisiin, vaan sen päätarkoituksena oli toimia suuntaa antavana toteutuksena, joka oli joustava mahdollisille muutoksille sekä helposti jatkokehitettävissä.

Ensimmäisen demon luomisen aikana havaittiin tarve CSS-rakenteen ehostamiselle. Mobiilioptimoinnin ja CSS3-ominaisuuksien myötä CSS-rakenne kasvoi huomattavasti ja syntyi tarve sen hallinnoimiseksi. Projektiryhmä päätti hyödyntää CSS-esikäsittelykieltä nimeltä Sass ja kielelle kehitettyä kehystä nimeltään Compass. Niiden avulla kyettiin hallinnoimaan paremmin alati kasvavaa CSS-rakennetta (ks. luku 7.2 CSS3 ja esikäsittely kehittämisen apuna).

Toisen demovaiheen toteutus jatkui siitä, mihin ensimmäisen demon aikana päästiin. Toisin kuin ensimmäisessä demossa, toisessa demossa päätettiin toteuttaa toimiva kokonaisratkaisu. Siihen kuului palvelinpuolen toteuttaminen, joka sisälsi arkkitehtoniset ratkaisut ja REST-rajapinnat (ks. luku 7.5 JavaScript MVC -valintojen maailma). Lisäksi otettiin käyttöön JavaScript MVW[15] -kehys nimeltään AngularJS, joka vastasi sovellusarkkitehtuurin näkymäkerroksesta (ks. luku 7.6 SPA-verkkosovelluksen toteuttaminen AngularJS-kehyksellä). Projektissa tehdyistä demoista rakentui lopullinen prototyyppi.

7.2 CSS3 ja esikäsittely kehittämisen apuna

Projektin alusta alkaen hyödynnettiin CSS3-moduulien tarjoamia uusia ominaisuuksia. Nämä ominaisuudet mahdollistivat interaktiivisen ja visuaalisen käyttöliittymäkokemuksen, joka oli yksi projektin tavoitteista. Tyyliohjeissa hyödynnettiin CSS3-ominaisuuksia aina animaatioista varjostukseen.

CSS3-ominaisuudet auttoivat projektiryhmää kolmella eri osa-alueella: interaktiivisuudessa, tyylissä ja rakenteessa sekä etenkin mobiilioptimoinnissa. Osa aiemmin vain JavaScript-kielen mahdollistamista interaktioista kyettiin toteuttamaan

[15] Model-View-Whatever, joka on Googlen oma termi kehykselle ja tarkoittaa käytännössä samaa kuin MV*

hyödyntäen CSS3-tyylikielen mahdollistamia animaatioita, transitioita ja pseudo-luokkia. Esimerkiksi navigointipuun sulkeutumisanimaatio toteutettiin hyödyntäen CSS3-transitioita.

Interaktion lisäksi tehostettiin käytettävyyttä ja visuaalista ulkoasua muun muassa CSS3:n liukuväreillä, varjoilla ja pseudo-elementeillä. CSS3-ominaisuuksien avulla pystyttiin toteuttamaan näitä visuaalisia tyylitehosteita, jotka oli aiemmin saavutettu vain kuvien avulla. Lisäksi kuvien sijasta hyödynnettiin Icon Font -tekniikkaa, jolla tarvittavat glyyfikuvakkeet (glyphs) muodostettiin fonttien avulla. Fonttipohjaiset kuvakkeet ovat vektorigrafiikkaa, joten niiden kokoa voidaan kasvattaa tai pienentää menettämättä kuvakkeen tarkkuutta. Fonttipohjaisia kuvakkeita voidaan muokata CSS-ominaisuuksien, kuten font-size-määrityksen avulla. Kuvakkeena toimivalle fontille voi myös vaihtaa väriä ja sille voidaan antaa varjostus. Erilaiset hyödynnetyt tyylitehosteet vähentävät kuvien käytön tarvetta ja täten tehostavat sivunlatauksen suorituskykyä.

Puhtaalta pöydältä aloitetun käyttöliittymäsuunnittelun tärkeimpään asemaan nousi kuitenkin mobiilioptimointi. Se saavutettiin Media Queries -tekniikan ja prosentuaalisten kokomäärityksien avulla. Media Queries -tekniikan avulla pystyttiin mukautuva käyttöliittymäsuunnittelu toteuttamaan riippumattomaksi näytön koosta. Media Queries -määritykset reagoivat näytön ominaisuuksiin ja tarvittaessa muokkaavat käyttöliittymää. Käyttöliittymän komponentit skaalautuvat ja paikoin muuttuvat täysin näytön koon mukaan. Käyttöliittymäsuunnittelua ei rajattu olemassa olevien laitteiden näyttöjen resoluutioiden perusteella, vaan projektissa seurattiin mukautuvan käyttöliittymäsuunnittelun periaatteita. Täten käyttöliittymän skaalautuvuudelle asetettiin ylä- ja alarajat. Ylärajaksi määriteltiin resoluutio 1920x1080 pikseliä ja alarajaksi resoluutio 320x480 pikseliä. Mukautuvan käyttöliittymäsuunnittelun mukaisesti käyttöliittymä tukee kaikkia resoluutioita rajamäärityksien välillä. Vaihtoehtoisesti mobiilioptimointi olisi voitu saavuttaa JavaScript-kielen avulla. Se on aiemmin ollut lähes ainoa tapa toteuttaa skaalautumiseen perustuva mobiilioptimointi, ennen Media Queries -tekniikkaa.

Uudet CSS3-ominaisuudet vähensivät JavaScriptin tarvetta ja vastaavasti lisäsivät CSS-tyyliohjeiden käyttöä. Ensimmäistä demoa kehitettäessä todettiin CSS-rakenteen kasvavan suureksi mobiilioptimoinnin ja selainkohtaisten etuliitteiden vuoksi. Tarvittiin parempi tapa hallita jatkuvasti kasvavaa CSS-rakennetta.

7.3 CSS-esikäsittelyn työkaluina Sass ja Compass

Ensimmäisessä demossa havaitun CSS-rakenteen hallinnoimisongelman ratkaisemiseksi lähdettiin tutkimaan viime aikoina suuren suosion saavuttaneita CSS-esikäsittelykieliä. Projektiryhmältä samoin kuin yrityksestä ei löytynyt aiempaa tuntemusta CSS-esikäsittelykielistä, joten kielen valinta jäi projektiryhmän vastuulle. Kieliä löydettiin useampia ja vaihtoehdot rajattiin kolmeen suosituimpaan: LESS-, Sass- ja Stylus-kieleen.

Projektiryhmä valitsi Sass-kielen sen suosion, hyvän tuen sekä erityisesti Compass-kehyksen (ks. luku 7.2.2 Compass) perusteella. Sass-kielen myötä CSS-rakenteesta saatiin järjestelmällinen, kun hyödynnettiin esikäsittelykielistä tuttuja ominaisuuksia, kuten sisäkkäistä syntaksia, muuttujia sekä mixin-laajennuksia. Lisäksi Sass-kielen syntaksin samankaltaisuus CSS-tyylikielen kanssa ja mahdollisuus kirjoittaa normaalia CSS-tyylikieltä olivat tärkeitä perusteita kielen valintaprosessissa. Kielen käyttöönotossa ei siis tarvitsisi opetella paljon uutta asiaa. Tämä katsottiin tärkeäksi, kun lukuisat yrityksen kehitystavat tulevat muuttumaan samanaikaisesti.

Mobiilioptimoinnin kannalta tärkeäksi nousi Media Queries -määrittelyjen käyttö sekä mahdollisuus tiivistää generoidut CSS-tiedostot. Määrittelyjen ei enää tarvinnut olla tyyliohjetiedoston pohjalla, vaan ne voitiin asettaa kunkin CSS-valitsimen sisälle, mikä paransi tyyliohjeen luettavuutta.

```
$blue: #3f76b7;
$red: #d14836;
header {
  background-color: $blue;
  @media(max-width: 320px) {
     background-color: $red;
  }
}
```

```
section[role="main"] {
  @media(max-width: 320px) {
    background-color: $blue;
  }
}
```

Ongelmana oli kuitenkin vielä raja-arvojen toistuminen. Raja-arvon ”320px” ylläpitäminen ei ollut käytännöllistä, sillä se esiintyi useammassa paikassa. Ratkaisuna hyödynnettiin Sass-kielen tarjoamaa mixin-laajennusrakennetta, jonka avulla yleiset raja-arvot saatiin määriteltyä yhteen helposti ylläpidettävään mixin-laajennukseen. Samalla Media Queries -määrityksessä olevat raja-arvot saivat nimien avulla semanttisemman tarkoituksen.

```
$blue: #3f76b7;
$red: #d14836;

@mixin breakpoint($point) {
 @if $point == favorites-hidden {
   @media (max-width: 1100px) { @content; }
 }
 @if $point == phone-view {
   @media (max-width: 320px)  { @content; }
 }
 @else if $point == retina {
   @media (-webkit-min-device-pixel-ration: 2),
          (min-resolution: 192dpi) {
            @content;
          }
 }
}

header {
  background-color: $blue;
  @include breakpoint(phone-view) {
     background-color: $red;
  }
}
section[role="main"] {
  @include breakpoint(phone-view) {
    background-color: $blue;
  }
}
```

Sass erottui muista esikäsittelykielistä suurelta osin Compass-kehyksen ansiosta. Yksi suurimmista syistä esikäsittelykielien käyttöönottoon oli CSS3-ominaisuuksissa esiintyvät selainkohtaiset etuliitteet. Compass-kehyksen sisältämien valmiiden mixin-laajennuksien avulla vältyttiin etuliitteiden toistamiselta eri puolilla CSS-rakennetta.

```
-webkit-transition: opacity 0.2s ease-in-out;
```

```
-webkit-transition-delay: 0;
-moz-transition: opacity 0.2s ease-in-out 0;
-o-transition: opacity 0.2s ease-in-out 0;
transition: opacity 0.2s ease-in-out 0;
```

Yllä oleva määrittely voidaan esittää Compass-kehyksen avulla seuraavassa muodossa:

```
@include transition(opacity 0.2s ease-in-out 0);
```

Compass-kehyksen avulla hallinnoitiin Sass-tiedostojen käännösprosessia. Käännösprosessi suoritettiin automatisoidusti kehyksen sisältämällä watch-komennolla. Lisäksi toteutettiin vaihtoehtoinen käännöstapa maven-liitännäisen avulla, joka kääntää Sass-tiedostoista CSS-tiedostoja sovelluskoodin käännösprosessin yhteydessä. Sass-kielen ja Compass-kehyksen avulla kyettiin tiivistämään CSS-tiedostot mahdollisimman pieniksi. Tiivistämisen avulla nopeutettiin sivun latausaikaa, mikä on tärkeää etenkin mobiilikäytössä.

Sass ja Compass loivat yhdessä kattavan vaihtoehtoisen ratkaisun CSS-rakenteelle. Niiden avulla parannettiin tyyliohjeiden rakennetta ja ylläpidettävyyttä. Lisäksi niissä on matala oppimiskynnys CSS-tyylikieleen tutustuneille, joten tekniikat tuovan lisäarvoa yrityksen tuotekehitykselle. Sass ja Compassin lisäksi demovaiheesta lähtien hyödynnettiin HTML5-standardia.

7.4 HTML5-standardin haasteet ja mahdollisuudet

Koska projektissa oli määritelty korkeat selainvaatimukset, prototyypin kehittämisessä pystyttiin ottamaan käyttöön HTML5-standardi. Projektiryhmä hyödynsi HTML5-standardin mukana tullutta semanttista rakennetta. Semanttiset elementit auttavat rakenteen luettavuudessa, mutta samalla yksinkertaisesta div-elementtirakenteesta tulee monimutkaisempi.

Monimutkaisuutta esiintyi etenkin semanttisten elementtien sisältämien käytäntöjen vuoksi. Projektin aikana tuli eteen tilanteita, jolloin ei voitu nopeasti sanoa, mitä elementtiä pitäisi käyttää. Esimerkiksi HTML5-käytäntöjen mukaan nav-elementtiä ei saisi käyttää kuin päänavigoinneissa (Hogan 2010, 41; Clark 2011). Tällöin oli jatkuvasti

mietittävä sopivinta vaihtoehtoa kuhunkin tilanteeseen. Lisäksi voidaanko taata, että kaikki kehittäjät noudattavat näitä vielä huonosti dokumentoituja sääntöjä, tai että kaikki ymmärtävät ne samalla tavalla? Säännöt eivät ole pakollisia, sillä uudet semanttiset elementit käyttäytyvät samalla tavoin kuin div-elementti. Toisaalta viekö säännöttömyys elementtien semanttisen tarkoituksen? Näitä kysymyksiä jouduttiin pohtimaan projektin aikana. Semanttisuuden todettiin parantavan rakennetta, mutta sääntöjen noudattaminen nosti oppimiskynnystä. Semanttisten elementtien lisäksi projektissa tärkeitä tekijöitä olivat HTML5 Storage ja History -ohjelmointirajapinnat.

Ensimmäisen demokierroksen pohjalta todettiin, että sovellus tulisi hyödyntämään paljon Ajax-tekniikkaa, jotta saavutettaisiin mukautuvan käyttöliittymäsuunnittelun käytännöt. Paljolti Ajax-tekniikkaan perustuvien verkkosovelluksien ongelmana on jo kauan ollut selainhistorian menettäminen. Ajaxin avulla tuotu sivu ei automaattisesti rekisteröidy selaimen muistiin, jolloin menetetään askelpalautin-napin toiminnallisuus (Hogan 2011, 194). Puutteelle oli löydettävä ratkaisu, koska askelpalautin on verkkosovellusten käytetyin nappi. Päätettiin hyödyntää HTML5 History -ohjelmointirajapintaa. Pikaisen tutustumisen ja testauksen jälkeen todettiin History-rajapinnan olevan vielä keskeneräinen. Selaimet käsittelevät selaushistoriaa eri tavoin, joten oli selvästi havaittavissa tarve kolmannen osapuolen kirjastolle, joka yhtenäistäisi History-rajapinnan käytön. Lisäksi selaimen historia oli saatava säilymään myös vanhemmilla selaimilla, jotka eivät tue HTML5 History -rajapintaa. Tutkittiin History-rajapinnan toteuttajia ja löydettiin Pjax- ja History.js-kirjastot. Vähitellen alettiin ymmärtää Ajax-tekniikkaan perustuvan SPA-verkkosovelluksen monipuolista JavaScript-tarvetta ja alettiin tutkia vaihtoehtoja rakenteen hallintaan.

7.5 JavaScript MV* -valintojen maailma

Havaittiin, että uudet HTML5-ominaisuudet, kuten History API ja Storage API, mahdollistavat kattavan SPA-verkkosovelluksen toteuttamisen. JavaScript MV* -kehykset mahdollistivat prototyypissä tarvittavat ominaisuudet, kuten selainhistorian säilyttämisen, templaatit, reitityksen ja integraation REST-rajapintojen kanssa.

Ensimmäisenä ja ehkä koko projektin suurimpana haasteena oli kehyksen valinta lukuisten JavaScript MV* -kehyksien välillä. Projektiryhmällä eikä kenelläkään yrityksen tuotekehityksessä ollut lainkaan aiempaa kokemusta JavaScript MV* -kehyksistä. Avuksi valintaan löydettiin Addy Osmanin[16] luoma TodoMVC-sivusto, jossa voi verrata miten kullakin JavaScript MV* -kehyksellä on luotu toiminnallisuuksiltaan identtinen Todo-sovellus. Sivusto pyrkii olemaan puolueeton ja antamaan yleiskuvan kunkin JavaScript MV* -kehyksen arkkitehtuurista. Koska sivusto on suosittu ja se perustuu täysin avoimeen lähdekoodiin, JavaScript MV* -kehyksien luojat ovat osallistuneet Todo-sovelluksien toteuttamiseen, ja siten sovellukset toimivat myös hyvinä esimerkkeinä itse kehyksistä.

Taustatutkimuksen pohjalta havaittiin, että kaikki JavaScript MV* -kehykset eroavat hyvin paljon toisistaan, joten kehyksen poistaminen tai vaihtaminen tulisi olemaan sekä kallis että haastava operaatio. Tärkeimpiä valintakriteereitä kehyksen valinnassa olivat kehyksen ylläpito, toteuttaja, suosio sekä kuinka aktiivisesti kehystä kehitetään. JavaScript MV* -kehyksien joukosta valittiin kolme kehystä, jotka vastasivat näitä kriteerejä: AngularJS, BackboneJS ja EmberJS. Kehyksien eroavuuksien ja projektiryhmän kokemattomuuden vuoksi näitä kolmea kehystä oli erittäin vaikea verrata toisiinsa. Ryhmän lopullisena valintana oli AngularJS.

7.6 SPA-verkkosovelluksen toteuttaminen AngularJS-kehyksellä

AngularJS oli ainoa kehys, joka oli saavuttanut virallisen rajapyykin eli version 1.0. Se on Googlen kehittämä, joten kehitysyhteisön aktiivisuuden voidaan olettaa olevan erittäin hyvää. AngularJS:n HTML-pohjaiset templaatit, yksikkötestaustuki ja hyvä integraatio REST-rajapintojen kanssa olivat ominaisuuksia, jotka vaikuttivat positiivisesti kehyksen valintaan.

[16] Googlen front-end kehittäjä

AngularJS tai ylipäätänsä JavaScript MV* -kehyksen käyttö oli projektin merkittävin tekninen haaste. Tämän takia JavaScriptiä täytyi katsoa aivan uudesta näkökulmasta. Projektiryhmän oli opiskeltava täysin uusi sovelluskehys sekä ymmärrettävä uusi JavaScript-pohjainen ajattelumalli. JavaScript MV* -kehyksen käyttöönotto nähtiin pakolliseksi ensimmäisen demototeutuksen jälkeen, mikäli haluttiin saavuttaa monipuolinen Ajax-pohjainen sovellusmalli. Lopulta itseopiskelun tuloksena JavaScriptiin saatiin looginen rakenne, joka mahdollisti rikkaan ja interaktiivisen SPA-sovelluksen luomisen.

AngularJS-kehys vastaa käyttäjäpuolen arkkitehtuurista. Kehys toimii ensimmäisenä kerroksena, joka reagoi käyttäjän tekemiin toimiin. Sen ansiosta JavaScript-rakenne saatiin jaettua itsenäisiin osiin samaan tapaan kuin palvelinpuolen arkkitehtuuri. Kehys vastaa suurelta osin sovelluksen interaktiivisuudesta ja näyttämisestä käyttäjälle. Vähitellen käyttäjäpuolen arkkitehtuurista syntyi oma kokonaisuutensa, joka olisi voitu irrottaa täysin palvelinpuolen toteutuksesta omaksi sovelluksekseen. Lopputulos vastaanottaa vain ja ainoastaan tarvittavat resurssit palvelinpuolelle toteutetun REST-rajapinnan kautta. SPA- ja JavaScript MV* -ajattelut mullistivat kokonaisarkkitehtuurin ja vaikuttivat myös palvelinpuolen toteutukseen.

7.7 Arkkitehtuurin elinkaari

Yrityksen sovellusarkkitehtuurin päivittäminen oli mobiilioptimoinnin lisäksi projektin päätavoite. Arkkitehtuurin uusiminen toi syvyyttä opinnäytetyöhön ja se todettiin tarpeelliseksi, mikäli haluttiin hyödyntää uusimpia teknologisia ratkaisuja. Kokonaisarkkitehtuuri ei ollut selvillä projektin alussa, vaan se eli ja sai uusia piirteitä projektin edetessä.

Ensimmäisen demon jälkeen havaittu JavaScript MV* -tarve heijastui kokonaisarkkitehtuuriin. Arkkitehtuuri jaettiin kahteen osaan: palvelinpuolen arkkitehtuuriin ja käyttäjäpuolen arkkitehtuuriin. Palvelinpuoli ei ollut enää vastuussa näkymästä, vaan se toteutettiin palvelukeskeiseksi ja tarjoamaan REST-rajapinta tietovaraston resursseille.

Tähän mennessä yrityksessä[17] ei ole hyödynnetty mitään sovelluskehystä rakenteen hallitsemiseksi. Täten kehitystavat eivät ole olleet yhteneväisiä, ja suurimpana ongelmana on ollut käytäntöjen puuttuminen. Ongelman ratkaisemiseksi arkkitehtuuriin päätettiin lisätä ohjelmistokehys, joka on suunniteltu auttamaan Javalla toteutettujen verkkosovelluksien rakenteen hallintaa. Lopulta päädyttiin valitsemaan Spring Framework ja Play Framework -kehyksien väliltä.

Projektiryhmä päätyi Spring Framework -ohjelmistokehykseen pääosin aikaisempien positiivisten kokemuksiensa myötä. Spring Framework tarjosi erittäin hyvän integroinnin jo olemassa olevaan sovellukseen. Play Framework puolestaan on viimeisimmissä versioissaan keskittynyt enemmän Scala-kieleen, ja se oli täysin tuntematon projektiryhmälle. Projektissa oli otettu käyttöön jo lukuisia uusia teknologioita, joten kokonaan uuden palvelinpuolen ohjelmistokehyksen opettelu nähtiin riskinä.

Ensimmäinen parannus arkkitehtuuriin oli käytäntöjen määrittäminen. Sovellusarkkitehtuurista yleisesti löytyvät tekniset kokonaisuudet päätettiin jakaa omiksi osikseen, mikä toi arkkitehtuuriin luettavuutta, selkeyttä ja ylläpidettävyyttä. Nämä modulaariset osat ovat Repository, Service ja Controller. Kukin niistä sisältää oman toiminnallisen tarkoituksensa ja vastuunsa.

Repository-taso kommunikoi tietokannan kanssa. Se luotiin hyödyntämään Java Persistence (JPA2) -rajapintaa aiemmin yrityksessä käytetyn Hibernate-tietokantakehyksen sijasta. Hibernate kuitenkin päätettiin säilyttää JPA2-rajapinnan toteuttajana. Tällöin toteutus on tehty standardin rajapinnan kautta, eikä se ei ole riippuvainen toteuttajasta. Tietokantakehyksen vaihtaminen onnistuu täten vähäisillä konfiguraatiomuutoksilla. *Service*-tasolla hyödynnetään Spring-kehyksen tarjoamia ominaisuuksia ja toteutetaan sovelluksen liiketoimintalogiikka. Lopuksi *Controller*-taso

[17] Sovelluksen tilaaja IMS Business Solutions Oy

tarjoaa REST-rajapinnan avulla tarvittavat resurssit asiakaspuolen arkkitehtuurille JSON-muodossa. Luotu rajapinta mahdollistaa myös integraation muihin sovelluksiin (ks. Kuvio 2).

Arkkitehtuurissa on tärkeää, että jokainen osa suorittaa vain ja ainoastaan oman vastuualueensa toimintoja. Luotu rakenne ei silti vielä poistanut perusoperaatioiden toistoa, joten avuksi luotiin abstraktit luokat, jotka toteuttavat CRUD-operaatioiden käsittelyn (liite 1). Abstraktien luokkien CRUD-operaatioista saatiin luotua täysin geneeriset hyödyntämällä geneeristä Javaa ja Reflection-tekniikkaa. Abstraktien luokkien ansiosta perivät luokat omaavat automaattisesti CRUD-operaatiot. Kolmitasoarkkitehtuurin mukaiset palvelu- ja tietovarastokerrokset perivät abstraktit CRUD-luokat, ja lopputuloksena vältytään toistamasta samaa rakennetta. Lisäksi abstraktit toteutukset voidaan ylikirjoittaa tapauskohtaisesti, mikäli sille nähdään tarvetta.

Palvelinpuolen jaetut kerrokset ovat yhteydessä toisiinsa hyödyntämällä Spring-kehyksen tarjoamaa IoC-säiliötä ja Dependency Injection -tekniikkaa. Spring hallinnoi täysin omaa IoC-säiliötään, joten projektin aikana oli ratkaistava, kuinka vanhasta kehyksettömästä sovelluskoodista pystyttiin kutsumaan IoC-säiliössä olevia Spring-papuja. Spring-pavut pohjautuvat täysin IoC-säiliöön, eikä niitä voi kutsua normaaliin tapaan Javan new-avainsanalla. Lopulta kuitenkin havaittiin, että Facade-suunnittelumallin avulla kyettiin luomaan yhteys vanhan ja uuden sovellusarkkitehtuurin välille.

Toteutettu palvelinpuolen arkkitehtoninen ratkaisu haaroitettiin projektin aikana Git-versiohallinnan avulla muihin yrityksen projekteihin. Lopullinen palvelinpuolen toteutus noudatti kolmitasoarkkitehtuurin periaatteita, jakaen arkkitehtuurin itsenäisiin osiin ja samalla se toimi kokonaisuudessaan yksittäisenä palveluna (ks. Kuvio 2).

8 Prototyypin demojen käytettävyystestaus

”If you want a great site, you’ve got to test.” (Krug 2006, 133)

Tässä luvussa käsitellään opinnäytetyöprojektissa suoritettua käytettävyystestausta. Testausta päädyttiin tekemään kahdenlaista: käytettävyystestausta (usability testing) ja automatisoitua hyväksymistestausta (automated acceptance testing). Näistä edellinen päätettiin tehdä videoituina testitilanteina, joissa kommunikoitiin käyttäjän kanssa. Jälkimmäisessä käytettiin Cucumber-testaustyökalua, jonka avulla saatiin samalla kirjattua käyttötapaukset.

Käytettävyystestaus nähtiin luonnollisena ja olennaisena osana projektia, ja sen tarkoituksena oli sovelluksen käytettävyyden testaaminen. Yrityksessä ei ole aiemmin toteutettu käytettävyystestausta, joten se toimi samalla tärkeänä kokeiluna, mikäli testaus halutaan tulevaisuudessa ottaa vakituiseen käyttöön. Testaus kokonaisuudessaan suoritettiin suurin piirtein projektin puolessa välissä, jolloin sovelluksen ulkoasun suunnittelu oli jo edennyt pitkälle, mutta projektissa oli vielä hyvin aikaa tehdä tarvittavia muutoksia testituloksien pohjalta.

Käytettävyystestauksessa päätettiin käyttää Steve Krugin ”Lost-our-lease”-menetelmää. Testauksen tarkoituksena ei ollut luoda tarkkaa, pitkälle mietittyä ja analysoitua testausta, vaan saada kiireellisessä aikataulussa kohtuullisen paljon tietoa käyttöliittymän käytettävyydestä mahdollisimman pienellä budjetilla. Tarkoituksena oli myös ottaa kevyt ensiaskel käytettävyystestaukseen, sillä projektiryhmällä eikä yrityksen tuotekehityksellä ollut juurikaan kokemusta käytettävyystestauksesta. Näihin tarkoitusperiin ”Lost-our-lease”-menetelmä vaikutti sopivan erinomaisesti.

Testaukseen osallistuville ei esitetty muita kriteereitä kuin, että heidän tulisi allekirjoittaa projektiryhmän laatima salassapitosopimus ja olla halukkaita osallistumaan testaukseen. Osallistujilta ei siis vaadittu kokemusta IMS-toimintajärjestelmästä, mobiililaitteista tai ylipäätänsä tietotekniikasta. Lähes kaikilta osallistujilta kuitenkin löytyi tuntemusta näistä aiheista.

Testaus päätettiin suorittaa kahdessa vaiheessa (testauskierroksessa). Kumpikin testauskierroksista omistettiin eri näkymille: tablettinäkymälle ja älypuhelinnäkymälle. Näin ryhmä pystyi keskittymään yhden sovellusnäkymän rakentamiseen kerrallaan. Sovellusta voi käyttää myös tietokoneella, mutta nämä kaksi näkymää todettiin tärkeämmiksi, koska käyttöliittymä halutaan ensisijaisesti optimoida mobiililaitteille.

Tietokoneissa, tableteissa ja älypuhelimissa on erikokoisia näyttöjä. Resurssien puitteissa testaus toteutettiin vain yhdellä tablettikoolla ja yhdellä älypuhelinkoolla. Niiden perusteella pyrittiin myös arvioimaan käyttöliittymän soveltuvuutta muun kokoisiin näyttöihin. Soveltuvuuden arviointia helpottivat mukautuvan websuunnittelun periaatteet.

Molemmilla testauskierroksilla päätettiin suorittaa testi viidelle henkilölle. Näin saatiin sekä Krugin että Nielsenin perusteella tarpeeksi hyviä tuloksia. Jokainen testi tallennettiin videokameralla ja olennaisimmat asiat purettiin tämän jälkeen tekstiksi, jota analysointiin tulosten saamiseksi.

Molemmat testauskierrokset koostuivat seuraavista vaiheista:

- Suunniteltiin testauskysymykset ja -tehtävät.
- Toteutettiin demo ja hankittiin testauksen vaatimat materiaalit.
- Kokeiltiin demoa ja testaustapauksia pikaisesti yrityksen sisällä.
- Pyydettiin henkilöitä ottamaan osaa testaukseen.
- Testaus suoritettiin viidelle henkilölle.
- Videoidut testitilanteet purettiin pääpiirteittäin ja analysoitiin.
- Luotiin analysoinnin pohjalta lista muutosehdotuksista.

Ensimmäisen testauskierroksen demo matki varsinaisen sovelluksen käytettävyyttä ja toiminnallisuutta. Demon käyttö ei vaatinut internetyhteyttä ja se oli mahdollista viedä asiakkaan luo tabletilla testaussessioita varten. Ensimmäisellä testauskierroksella testeihin pyydettiin yrityksen lähistöllä toimivia asiakkaita. Yhteydenoton yhteydessä

ilmoitettiin lyhyesti käytettävyystestauksen tarkoitus, arvioitu kesto, käytettävä tallennuskeino (videokuvaaminen) sekä mainittiin pienestä osallistumispalkkiosta.

Toisella kierroksella demo toteutettiin toimivana selainsovelluksena. Sen etuina olivat helposti muodostettava yhteys projektin tilaajan tiloissa langattoman verkon välityksellä ja sovelluksen ketterä parantelu. Demo päätettiin toteuttaa siten, ettei siihen ollut mahdollista ottaa yhteyttä IMS Business Solutions Oy:n toimitilojen ulkopuolelta, ja siten testaus tehtiin yrityksen sisällä. Toisena syynä oli, ettei ryhmä saanut testausta varten omaa älypuhelinta, vaan testitapaamisissa piti joka kerta lainata jonkun yrityksen työntekijän älypuhelinta. Testaukseen osallistujat eivät olleet tuotekehityksen, vaan myynnin, markkinoinnin ja koulutuksen puolelta.

8.1 Käytettävyystestien tarkoitus

Käytettävyystesteissä videokuvattiin käyttöliittymää, mutta niissä pyrittiin huomioimaan myös se, miten osallistujat suhtautuivat käyttöliittymään ja vaikuttivatko he turhautuneilta vai eivät. Testauksessa pyrittiin kartoittamaan navigointia, demojen ulkoasua sekä muutaman yleisen toiminnallisuuden ja kuvakkeiden toimivuutta ja ymmärrettävyyttä. Mittareina käytettiin sitä, miten hyvin käyttäjät onnistuivat heille annetuissa tehtävissä, kuinka hyvin he tunnistivat näkemiään asioita, kuinka helposti he turhautuivat, ja ymmärsivätkö he, mitä tapahtui heidän painaessaan mitäkin nappia.

Navigoinnin osalta haluttiin tutkia sen helppoutta: kuinka helppoa nappeihin on osua, ymmärtääkö käyttäjä, mitä ikonit tarkoittavat ja mitä käyttöliittymässä tapahtuu, kun hän painaa tiettyä nappia. Sovelluksen ulkoasu pidettiin yksinkertaisena grey boxing -menetelmän mukaisesti. Osallistujilta kuitenkin kyseltiin ulkoasun miellyttävyydestä siltä varalta, että se auttaisi paljastamaan oleellisia puutteita tai hyviä puolia ulkoasusta. Useat teokset suosittelevat hankkimaan tietoa siitä, miten ja missä mobiilisovellusta käytetään ja mitä sillä tehdään. Tätä tutkittiin käytettävyystestauksessa kysymällä osallistujilta, miten he uskoisivat käyttävänsä sovellusta. Poikkeuksena oli yksi osallistujista, joka ei ollut koskaan käyttänyt toimintajärjestelmiä.

8.2 Testaustilanteet

Kaikki testaus tapahtui testaukseen osallistuvan henkilön yrityksen tiloissa. Tilana oli yleensä neuvotteluhuone tai osallistujan oma työhuone. Tällä pyrittiin vähentämään osallistujan testiin kuluttamaa aikaa ja resursseja sekä vähentämään tilanteen luomaa jännitystä. Jokainen testaustilanne aloitettiin lyhyellä esittäytymisellä, muutamalla käyttöliittymää koskemattomalla kysymyksellä ja muutaman minuutin vapaalla keskustelulla. Tämän vaiheen tarkoituksena oli tutustua osallistujaan ja laukaista hänen tuntemaa mahdollista jännitystä ennen testauksen alkamista. IMS Business Solutions Oy:n ulkopuolisilta osallistujilta pyydettiin myös allekirjoitus salassapitosopimukseen. Kysymysten jälkeen osallistujalle painotettiin, että tapaamisessa oli tarkoitus testata ja kuvata käyttöliittymää eikä häntä. Tällä pyrittiin vakuuttamaan osallistujalle, ettei kyseessä ole testi, jossa häntä arvioitaisiin.

Testaustilanteissa testin pitäjä pyrki olemaan mahdollisimman rauhallinen ja neutraali, jottei hänen olemuksensa eikä kommunikointinsa vaikuttaisi testin osallistujan suoriutumiseen. Tilanteissa pyrittiin myös siihen, ettei osallistujaa autettaisi ongelmatilanteissa, jotta pystyttäisiin näkemään ongelmien todellinen vakavuus. Tästä jouduttiin poikkeamaan kerran, kun testissä käytetty selain alkoi oikutella. Osallistujia pyydettiin myös ajattelemaan ääneen, jotta saataisiin enemmän tietoa siitä, minkälaisia ajatuksia käyttöliittymä heissä herättää. Kovin moni osallistujista ei kuitenkaan noudattanut tätä ohjetta toivotusti, joten testin pitäjä joutui välillä muistuttamaan asiasta. Tästä huolimatta osallistujista pystyi yllättävän helposti huomaamaan, jos he vaikuttivat turhautuneilta tai hämmentyneiltä. Testaustilanteiden välillä pyrittiin pitämään vähintään sen pituinen väli, että aiemman testin videomateriaali kerittiin tällä välillä purkaa tekstiksi. Tällä pyrittiin välttämään sitä, että aiempi testaustilanne vaikuttaisi seuraavan testaustilanteen tulkintaan.

8.3 Käytettävyystestien analysointi ja tulokset

Aineisto purettiin pääpiirteittäin sen perusteella, miten osallistujat toimivat eri tehtävissä, onnistuivatko he pääsemään haluttuun lopputulokseen, minkälaisia ongelmia he kohtasivat ja mitä muita olennaisia kommentteja he antoivat testien aikana.

Käytettävyystestauksen aineistoa analysoitiin osittain määrällisesti ja osittain laadullisesti. Tuloksista ei piirretty kaavioita, vaan ne kirjoitettiin auki tekstinä. Analysoinnissa käytettiin Steve Krugin ohjeita sen sijaan, että tuloksia olisi tutkittu kylmästi onnistumisprosenttien tai muiden mittarien kautta. Tulokset ja analysointi löytyvät Käytettävyystestauksen analysointi tablettinäkymässä raportista (luottamuksellinen liite 6), ja Käytettävyystestauksen analysointi älypuhelinnäkymässä raportista (luottamuksellinen liite 7).

9 Prototyyppiin päätetyt parannukset

Käytettävyystestausten tulosten pohjalta kirjattiin ylös muutosehdotukset, joiden perusteella havaitut ongelmat todennäköisesti saataisiin korjattua. Lista jaettiin ensin kolmeen osaan: muutokset tablettinäkymään, muutokset älypuhelinnäkymään ja muutokset molempiin näkymiin. Tämän jälkeen listasta eroteltiin vielä parannukset, jotka pyritään tekemään opinnäytetyöprojektin puitteissa sekä parannukset, jotka jätetään jatkokehitykseen. Tarkka lista parannuksista löytyy raportista Tehtävät muutokset ja parannukset (luottamuksellinen liite 8).

Esimerkkinä tehdyistä havainnoista joitakin kuvakkeita muutettiin paremmin vastaamaan niiden tarkoitusta ja joitakin käyttöliittymän elementtejä suurennettiin tai siirrettiin sopivampiin paikkoihin. Esimerkiksi käyttöliittymän **Näytä suosikit** -nappia ei osattu etsiä puhelinnäkymässä, joten se siirrettiin paikkaan, josta se löytyisi huomattavasti helpommin.

10 Prototyypin toimivuuden osoittaminen

Sovellustestauksessa nähdään yleisesti neljä tasoa: yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. Yksikkötestaus on suositeltavaa tehdä JavaScript MV* -kehyksiä käytettäessä, koska JavaScript on dynaamista, eikä se sisällä kääntäjää, joka ilmoittaisi virheellisestä koodisyntaksista. Projektissa kuitenkin päätettiin olla suorittamatta yksikkö-, integraatio- ja järjestelmätestausta, koska ensisijainen tarve oli käyttää resurssit käyttöliittymän ja sen käytettävyyden testaukseen. Yksikkötestaukseen ei myöskään osattu varautua projektin alussa, koska JavaScript MV* -kehyksen käyttöönotosta ei tuolloin ollut vielä tietoa.

Sovellustestausta ei kuitenkaan haluttu jättää kokonaan pois projektista, joten tutustuttiin hyväksymistestaukseen ja muutamaan automatisoituun testausympäristöön. Hyväksymistestauksen avulla pystytään toteamaan, että sovellus pystyy suoriutumaan sille määritellyistä vaatimuksista. Tämä nähtiin tärkeänä asiana. Tutustumisen kohteina oli kaksi selaintestausympäristöä: Cucumber ja Selenium. Niistä valittiin pelkästään Cucumber-testausympäristö. Selenium-ympäristöön verrattuna se menee askeleen pitemmälle liiketoimintalogiikkaan perustuvan tasonsa ansiosta. Selenium-ympäristön etuna on puolestaan se, että Firefox-lisäosan ja ohjelmointiympäristön avulla testejä pystytään luomaan helpommin ja nopeammin.

Näiden kahden ympäristön välillä ei siis ole mullistavia eroavaisuuksia. Cucumber valittiin projektiin testausympäristöksi, koska se perustuu osittain liiketoimintalogiikkaan eli sitä on helppo verrata käyttötapauksiin, ja ryhmällä oli lisäksi kokemusta sen käytöstä. Cucumber on jo käytössä IMS-toimintajärjestelmässä, joten uuden, lähes samanlaisen, testausympäristön tuomista kehitysympäristöön ei nähty järkeväksi. Cucumber on myös valmiiksi yhdistetty tuotekehityksen Hudson-integraatiotyökaluun.

Hyväksymistestit tehdään yleensä käyttäjätarinoiden pohjalta. Jo projektin alussa Cucumber-testit päätettiin kuitenkin tehdä käyttötapauksien pohjalta käyttäjätarinoiden sijaan. Vaatimusmäärityksessä kuvatut käyttötapaukset kirjoitettiin alusta alkaen sillä

oletuksella, että niiden perusteella pystytään myöhemmin kirjoittamaan kattavat testitapaukset prototyyppiin rakennetuille ominaisuuksille. Testit saatiin rakennettua ripeästi jo projektin alussa kirjoitettujen käyttötapausten pohjalta ja ne saatiin kattamaan kaikki suunniteltujen käyttötapausten askeleet. Projektiryhmällä oli jo kokemusta Cucumber-testien tekemisestä. Kirjoitetut Cucumber-testit löytyvät kokonaisuudessaan tietosysteemikansiosta.

11 Tulokset

Projektin tuloksena syntyi suunnitelmien mukaisesti vuoden 2012 loppupuolella mobiilioptimoidun käyttöliittymän prototyyppi sekä käyttäjäkyselyn ja kahden testauskierroksen dokumentaatio ja analyysi. Prototyyppi syntyi projektin aikana toteutettujen demojen lopputuloksena. Sen kehitysprosessi kesti suunnittelusta testaukseen ja sen toteuttaminen sisälsi tässä opinnäytetyössä esitetyt vaiheet. Prototyyppi ei sellaisenaan ole valmis käyttöönotettavaksi, mikä oli tilaajan ja koko projektiorganisaation tiedossa jo projektin alussa.

Käyttäjäkyselyllä saaduista vastauksista näkyi selvästi eri toimintojen ja ominaisuuksien suosio vastaajien keskuudessa. Kysely pohjusti toteutusta, ja saatujen tulosten avulla saatiin parempi kuva siitä, mistä sovelluksen kehittäminen tulisi aloittaa ja mitä siihen kannattaisi sisällyttää. Kyselyn avulla saadut vastaukset eivät vaikuttaneet ainoastaan prototyypin kehitykseen, vaan niistä saatiin jatkokehitysideoita prototyypille ja yleistietoa toimintajärjestelmän nykyisestä käytöstä.

Opinnäytetyö toi uusia menetelmiä yrityksen tuotekehityksen työnkulkuun. Sen tuloksena päivitettiin yrityksen sovellusarkkitehtuuria ja saatiin kattava näkemys uusista vaihtoehtoisista teknologisista ratkaisuista. Uudet käyttöönotetut teknologiset ratkaisut saatiin integroitua nykyiseen sovellukseen ilman suurempia ongelmia. Lisäksi palvelinpuolen arkkitehtoniset ratkaisut otettiin käyttöön yrityksen muissa projekteissa jo opinnäytetyöprosessin aikana. Valitut teknologiat auttoivat projektiryhmää pääsemään lopputulokseen ja saavuttamaan käyttöliittymän mobiilioptimoinnin. Toisaalta vaihtoehtoisiin tekniikoihin tutustuminen ja niiden käyttöönotto olivat jo itsessään osa projektin tulosta.

Projektissa mobiililaitteilla suoritettu käytettävyystestaus pyrki selvittämään prototyypin käytettävyyttä painottuen mobiilikäyttöön. Käytettävyystestien tuloksien pohjalta pystyttiin parantamaan prototyypin käytettävyyttä ja ulkoasua. Testauksen avulla saatiin selville enemmän asioita, kuin mitä lähdettiin selvittämään. Käsiteltyjen mittarien lisäksi osallistujilta saatiin parannusehdotuksia sekä tietoa siitä, mihin ja missä tilanteissa he

haluaisivat käyttää käyttöliittymää. Testien perusteella saatiin myös ehdotuksia jatkotoimenpiteiksi. Hyväksymistestit rakennettiin käyttötapausten pohjalta. Niiden perusteella saatiin todettua, että prototyypille asetetut toiminnalliset vaatimukset ja tavoitteet täyttyivät. Yhteenveto prototyypin toteutuksesta löytyy Loppuraportista (luottamuksellinen liite 9).

12 Arviointi

Opinnäytetyön teoriaosuuteen oli paikoitellen vaikea löytää lähteitä, sillä monet projektissa käytetyistä teknologioista ovat aivan uusia, eikä niistä ole saatavilla vielä kirjallisuutta. Toisena ongelmana oli, että lähdemateriaalia, joka käsittelee nimenomaan selainsovelluksia, eikä vain internetsivuja tai mobiilisovelluksia, oli hyvin vaikea löytää. Mobiilisovelluksia ja mobiilioptimoituja internetsivuja käsittelevissä teoksissa oletetaan lähes poikkeuksetta, että lukija on rakentamassa tuotetta, joka on yleisesti saatavilla selaimella tai ladattavissa jostakin palvelusta. Suurin osa hankitusta tiedosta perustuu siis internetsivujen ja osa mobiilisovellusten käytettävyyteen. Tätä tietoa tulkittiin ja sovellettiin selainsovelluksen tarkoituksiin. Tulkinnassa on täytynyt erityisesti ottaa huomioon kaksi asiaa. Ensiksikin kyseessä ei ole vapaasti esillä oleva internetsivu, jolle kuka tahansa saattaisi eksyä. Toiseksi sovelluksen käytettävyydessä tulisi ensisijaisesti ottaa huomioon, että se on jatkuvasti käytössä oleva työväline, ja toissijaisesti että se on ensinä kemältä helposti ymmärrettävissä. Näiden kahden asian pohjalta tulkinta oli loppujen lopuksi varsin helppoa, sillä suurin osa tulkittavasta tiedosta keskittyi käytettävyyteen.

Lähteinä pyrittiin käyttämään pääasiassa alalla tunnettujen tekijöiden laatimaa kirjallisuutta ja internetsivuja. Opinnäytetyön edetessä samojen alalla tunnettujen henkilöiden ja heidän teostensa nimet tulivat vastaan lähes jatkuvasti. Tätä voitaneen pitää merkkinä siitä, että kyseiset henkilöt ja teokset ovat vähintäänkin arvostettuja alallaan. Muissa lähteissä keskityttiin teoksiin, jotka otsikoltaan ja johdannoltaan vaikuttivat käsittelevän opinnäytetyölle olennaisia asioita mahdollisimman monipuolisesti.

Termille ”mobiilioptimointi” ei löytynyt vielä virallista määritelmää. IT-yhteisöllä vaikuttaa kuitenkin olevan tarpeeksi yhtenäinen näkemys termin tarkoituksesta. Projektissa syntynyttä prototyyppiä voidaan tämän näkemyksen perusteella kutsua mobiilioptimoiduksi.

Parannusehdotuslistasta pystyi näkemään, että käyttöliittymän suunnittelussa oli otettu oikea suunta ja saatu luotua pohja, jota oli hyvä lähteä kehittämään eteenpäin. Lähes kaikki listan muutosehdotukset toivat pieniä parannuksia prototyypin käytettävyyteen tai ominaisuuksia, joita oli jo mietitty, mutta joita ei ollut vielä ehditty toteuttaa.

Koko opinnäytetyö oli itsessään oppimisprosessi ja sellaisena se oli hyvin toimiva ja opettava. Projektiryhmälle annettiin hyvin vapaat kädet käytettyjen menetelmien ja tekniikoiden valitsemiseksi. Koko opinnäytetyön ajan ryhmä sai tutustua uusiin teknologioihin, joilla toteuttaa sovellus ja siihen kuuluvat osat.

12.1 Projektinhallinnan arviointi

Projekti eteni projektin alussa aikataulun mukaisesti. Puolesta välistä eteenpäin aikataulussa pysyttiin noin yhtä sprinttiä edellä, jonka ansiosta projektin lopussa ryhmä sai tuoteomistajan luvalla keskittyä muutaman viikon ajan lopputyön dokumentaation viimeistelemiseen.

Projektissa hyödynnettiin ketteriä menetelmiä (Scrum, XP, Kanban) ja työ toteutettiin yrityksen tiloissa ja sen omistamilla laitteilla. Menetelmistä Scrum oli ehdottomasti tärkein. Ryhmällä oli jo kokemusta Scrum- ja XP-menetelmien mukaisesta työskentelystä. Monet XP-menetelmän opit eivät kuitenkaan soveltuneet projektin luonteeseen. Esimerkiksi pariohjelmointi ei tuntunut luontevalta, koska ryhmän jäsenet keskittyivät erilaisiin tehtäviin. XP-menetelmän suosimaa yksikkötestausta ei myöskään otettu käyttöön, koska testaamisessa haluttiin keskittyä käyttöliittymän käytettävyyteen.

Projektinhallinta toteutettiin lähes samalla tavoin, kuin miten ryhmä on aiemminkin työskennellyt. Tarkoituksena oli kuitenkin ottaa yksi lisäys, jotta ryhmä pystyy projektinhallintaa riskeeraamatta kokeilemaan ja oppimaan uusia asioita. Mukaan otettiin Kanban-menetelmän käyttämä fyysinen seurantataulu käyttäjätarinoiden hallinnoimiseksi. Taulu eroaa jonkin verran Scrum-menetelmässä käytetystä taulusta. Kanban-taulun avulla ryhmä pysyi paremmin selvillä prototyypille tärkeistä

ominaisuuksista ja sen kehittämisessä käytetyistä periaatteista[18], mutta muutoin taulun käyttö ei erityisemmin vaikuttanut projektiin.

Kokonaisuudessaan projektinhallinta sujui luontevasti ja ongelmitta. Projektinhallinnolliset raportit saatiin valmiiksi ajoissa. Ketterät menetelmät todettiin hyväksi valinnaksi ja tuttujen menetelmien vuoksi projektinhallintaan ei jouduttu käyttämään ylimääräisiä resursseja. Scrumin periaatteista onnistuttiin myös pitämään tiukasti kiinni.

12.2 Oman oppimisen arviointi

Projektiryhmän osaaminen on karttunut huomattavasti opinnäytetyön aikana. Kyselyjen laatimisesta tai käytettävyystestauksesta ei ollut juurikaan kokemusta ennen opinnäytetyötä. Omaa oppimista karttui myös etenkin opinnäytetyössä hyödynnetyissä teknologisissa ratkaisuissa, joista suurin osa oli projektiryhmälle ennestään tuntemattomia. Ryhmän aikaisemmissa projekteissa on ollut tapana tutkia ja ottaa käyttöön uusia teknologisia ratkaisuja. Tältä pohjalta ryhmä ei nähnyt uusiin teknologioihin tutustumista suurena riskinä, vaan pikemminkin hyväksi todettuna käytäntönä. Projektiryhmän onnistui tutustua uusiin teknologisiin ratkaisuihin ja toteuttaa ne opinnäytetyössä. Lisäksi ryhmä tutustui ja oppi lukuisia erilaisia vaihtoehtoisia ratkaisutapoja, joita ei otettu käyttöön prototyyppiin. Ryhmän jäsenet tekivät paljon yhteistyötä ja keskustelivat asioista, joten molempien ryhmän jäsenten tietotaito ja osaaminen kasvoi myös niissä asioissa, jotka eivät kuuluneet heidän vastuualueisiinsa.

12.3 Käyttäjäkyselyn arviointi

Käyttäjäkyselyllä ei onnistuttu saavuttamaan tarkkaa kohderyhmää. Tähän oli kuitenkin varauduttu etukäteen miettimällä kysymysten sisältöä. Saavutettu ryhmä ei myöskään

[18] Asiakaslähtöisyys, nopeus, helppous ja käytettävyys

ollut kaukana haetusta kohderyhmästä, joten tulkintaa ei jouduttu kovin paljoa tekemään.

Webropol-kyselytyökalu paljastui monipuoliseksi ja toimivaksi. Tulosten kerääminen tapahtui ohjelmalla automaattisesti, mikä vähensi työmäärä huomattavasti verrattuna siihen, jos kysely olisi käsitelty papereilla. Vaihtoehtoisiin kyselytyökaluihin ei ehditty tutustua. Edellä mainituin perustein voidaan todeta, että käyttäjäkysely onnistui tarvittavan hyvin sekä yrityksen että mobiilioptimoidun käyttöliittymän kehityksen osalta.

12.4 Valittujen teknologioiden arviointi

Projektin aikana projektiryhmä teki useita valintoja teknologisissa ratkaisuissa. Tärkeää oli, että niitä käytettäisiin parantamaan tilaajalle toteutettavaa tuotetta, ei haittaamaan sitä. Prototyypin toteutus on moniosainen ja se sisältää useita eri tarkoitukseen keskittyviä tekniikoita ja kehyksiä. Jokainen valittu teknologinen ratkaisu oli valintaprosessi. Sovelluskehityksessä on yleistä, että samalle sovelluskehityshaasteelle on useita toteutustapoja. Tämä johti projektin aikana jatkuvaan arvioimiseen ja päätöksen tekoon.

Käytettyjen teknologioiden valinta oli suunnittelutyön tulosta. Projektiryhmällä oli projektin alussa alustava mielikuva hyödynnettävistä teknologioista. Nämä alustavat ratkaisut esiteltiin projektiorganisaatiolle projektin aloituskokouksessa, ja lopputulos eroaa siten, että Apache Tiles -templaattikehyksen sijasta päätettiin hyödyntää JavaScript MV* -kehyksen tarjoamaa templaattirakennetta. Lisäksi alustavasti ei tiedetty mitään CSS-esikäsittelykielistä. Nämä vaihtoehdot muuttivat arkkitehtuuria merkittävästi, mutta alustavasti suunniteltua Spring-kehystä ei tarvinnut vaihtaa. Mobiilikäytössä pieneksi heikkoudeksi nousi mobiiliselainten JavaScript-suorituskyky. Toisaalta sovellus päivittää vain tarvittavan osan sivusta ja täten datasiirto mahdollisesti hitaan 3G-yhteyden läpi pienenee.

Hyödynnetyt kielet ja kehykset olivat projektiryhmän kannalta perusteltuja ja ne sopivat hyvin toteutettuun prototyyppiin. Teknologisissa ratkaisuissa mietittiin prototyypin elinkaaren tulevaisuutta ja muutama asia jäi vielä avonaiseksi. Teknologisten ratkaisujen joukkoon olisi voitu lisätä vielä muutamia. Näitä arvioidaan luvussa 13 Jatkokehitysajatukset.

12.5 Käytettävyystestauksen arviointi

Krugin suosittelema testausmenetelmä todettiin toimivaksi. On todennäköistä, että paremmin suunniteltuna ja analysoituna käytettävyystestauksesta olisi mahdollista saada parempia tuloksia. Paremmin suunniteltuun testaamiseen verrattuna ”Lost-our-lease”-menetelmä ei kuitenkaan kuluta juurikaan resursseja ja aiheuttaa huomattavasti vähemmän stressiä testien laatijoille. Menetelmä tarjosi kattavan kokonaisuuden, joka sopi hyvin ensi kertaa käytettävyystestaukseen tutustuvalle.

Käytettävyystestauksessa suositellaan testauksen jatkuvaa toistamista tietyin aikavälein, jotta saadaan tietoa siitä, ovatko ongelmat poistuneet tehtyjen parannusten myötä ja mitä uusia ongelmia ilmenee näiden ongelmien poistuessa. Testauksen toistamista ei lyhyessä projektissa kuitenkaan pystytty toteuttamaan, koska suoritetut kaksi testauskierrosta omistettiin eri näkymille.

13 Prototyypin jatkokehitysajatukset

13.1 JavaScript MV* -kehyksen uudelleenharkinta

Projektissa tutustuttiin suosituimpiin JavaScript MV* -kehyksiin ja niistä päätettiin ottaa käyttöön AngularJS. Projektiryhmä kuitenkin totesi, että olisi ideaalisinta tutustua tarkemmin ainakin kahteen AngularJS-kehyksen kanssa kilpailevaan kehykseen ennen kuin voidaan sanoa, mikä niistä kannattaa ottaa käyttöön. Jatkokehityksessä tulisi ensimmäisenä kokeilla näitä vaihtoehtoja, ennen kuin kehitetään muita ominaisuuksia, koska JavaScript MV* -kehyksen vaihto on myöhemmin kallis ja haastava operaatio.

Kehyksen valinnassa tärkeässä roolissa tulee olemaan i18n-lokaalisaatio ja Tomcat-spesifinen context path, jotka jäivät opinnäytetyön aikana vielä avoimiksi. Toisaalta JavaScript MV* -kehyksen hyödyntäminen nähtiin parhaaksi vaihtoehdoksi interaktiivisten SPA-sovelluksien toteuttamiseksi. Havaittuja ongelmia tutkittiin opinnäytetyön aikana ja todettiin, että ne ovat ratkaistavissa, eivätkä ne aiheuttaneet opinnäytetyöprosessiin muutoksia.

13.2 JavaScript rakenteen hallinnointi

Projektiryhmä ei hyödyntänyt viime aikoina suuren suosion saavuttaneita JavaScript-esikäsittelykieliä, kuten CoffeeScript-kieltä. Projektissa hyödynnettiin jo lukuisia yrityksen tuotekehitykselle uusia tekniikoita, joten esikäsittelykielen hyödyntäminen nähtiin liian suurena oppimiskynnyksenä yrityksen tuotekehitykselle. Lisäksi projektiryhmä piti tärkeänä JavaScript-kielen osaamista. CoffeeScript kuitenkin tuo ylläpidettävyyttä JavaScript-rakenteeseen, joten sen hyödyntämistä on harkittava jatkossa. Esikäsittelykielen lisäksi on suositeltavaa hyödyntää AMD (Asynchronous Module Definition) -formaattia, jolla hallinnoidaan JavaScript-rakennetta ja rakenteellisia riippuvuuksia.

13.3 Muokkaamisominaisuudet mobiililaitteilla

Mobiilioptimoitu käyttöliittymäsovelluksen prototyyppi on suunnattu katsojakäyttäjille, koska mobiilialustojen selaimet eivät tue nykyisessä IMS-toimintajärjestelmässä käytettyjä vanhentuneita teknologisia ratkaisuja, kuten Java-appletteja. Osa IMS-toimintajärjestelmän muokkausominaisuuksista täytyy siis suunnitella ja rakentaa uudestaan mobiiliselaimille ystävällisillä menetelmillä, kuten HTML5 Canvas -menetelmällä. Tämä puolestaan vaatii huomattavan paljon resursseja. Jatkokehityksen yhteydessä tulisi siis miettiä, onko muokkausmahdollisuuksia ylipäätänsä järkevä lähteä tekemään, ja missä vaiheessa ja miten ne kannattaisi toteuttaa.

13.4 Tuotteen ulkoasu

Projektissa keskityttiin sovelluksen käytettävyyteen ja ulkoasuun, mutta sovelluksen tuotteistamisen yhteydessä esimerkiksi sovelluksen värimaailmalle saatetaan asettaa uusia kriteerejä. Grey boxing -tekniikan avulla sivuston rakenne saatiin hyvälle mallille ja käytettävyystestauksen aikana pystyttiin toteamaan, että rakenne näyttää ja tuntuu hyvältä. Suurin työ ulkoasun viimeistelyssä on todennäköisesti värimaailman toteuttaminen ja kuvakkeiden viimeisteleminen.

13.5 Räätälöinnin mahdollistaminen

Projektin aikana ei ehditty toteuttaa sovellukseen juurikaan mahdollisuutta käyttäjäkohtaiseen räätälöintiin. Tämä asia pidettiin kuitenkin mielessä sovelluksen jatkokehitystä ja tulevaisuutta ajatellen. Räätälöinti tulisi mahdollistaa sekä käyttäjä- että yrityskohtaisesti. Toimintajärjestelmää käyttävällä yrityksellä voi olla tarve käyttää omaa teemaa, jossa käyttöliittymän värit vastaavat yrityksen värejä, ja yrityksen logo näkyy käyttöliittymässä.

Yksittäisellä käyttäjällä puolestaan tulisi olla mahdollisuus räätälöidä sivulla näkyviä elementtejä tarpeen mukaan. Esimerkiksi käytettävyystestauksessa paljastui, että käyttäjät toivoisivat mahdollisuutta keskittyä tekstin lukemiseen ilman, että käyttöliittymän reunoilla olevat elementit häiritsevät heitä.

13.6 Käytettävyystestauksen tulokset

Käytettävyystestauksen tulosten perusteella päätettiin tehdä muutoksia käyttöliittymään. Osa näistä tehtiin jo projektin aikana ja osa päätettiin jättää jatkokehitysehdotuksiin. Kaikki ehdotetut muutokset on järkevää toteuttaa prototyypin jatkokehityksessä. Sekä projektissa suoritetut että jatkokehitykseen jätetyt ehdotukset löytyvät luottamuksellisesta liitteestä 8 Tehtävät muutokset ja parannukset.

14 Pohdinta ja johtopäätökset

Mobiilioptimoitu käyttöliittymä on ajankohtainen yhä lisääntyvän mobiililaitteiden käytön myötä, ja yrityksen kannattaisi panostaa sen kehitykseen. Toteutettu prototyyppi antaa yritykselle paremman kokonaiskuvan siitä, mihin toimintajärjestelmän pitäisi tulevaisuudessa suuntautua, jotta se palvelisi paremmin alati kasvavaa mobiilitarvetta. Prototyyppi tuo mukanaan lukuisia uusia teknologisia ratkaisuja ja samalla se toimii hyvänä pohjana jatkokehitykselle.

Yritys tulee hyötymään uusista teknologisista ratkaisuista ja suurilta osin hyödyntääkin niitä jo. Opinnäytetyöprojektissa hyödynnetyt lukuisat tekniset ratkaisut vaikuttavat tuotekehityksen kehitystapoihin. Projektiryhmän vapaus valita hyödynnetyt teknologiset ratkaisut korostivat etenkin projektiryhmän taitoja, ja täten myös yritys hyötyi toteutuksesta sekä opitusta. Projektissa toteutetut vaiheet osoittivat, että valitut teknologiset ratkaisut ovat toimivia, käyttöliittymä on käytettävä ja projektiryhmä on kartuttanut huomattavasti osaamistaan opinnäytetyöprojektin aikana.

Opinnäytetyössä tehdyt käyttäjäkysely ja käytettävyystestaus avasivat yrityksen ovet asiakaslähtöisempään tuotekehitykseen. Kiinnostus uusien kyselyjen ja testauksen tekemiseen ja niiden pohjalta saatuihin tuloksiin näkyi selvästi yrityksen sisällä. Asiakaslähtöisyys vaikuttaa suurelta osin toteutettuihin ominaisuuksiin, käytettävyyteen ja täten myös hyödynnettyihin ratkaisuihin. Sillä pystytään tulevaisuudessakin parantamaan kaikkia yrityksen tarjoamia tuotteita ja palveluita. Jotta kyselyjen tekemistä on mahdollista jatkaa, on suositeltavaa hankkia yritykselle sähköinen kyselyjen tekemis- ja lähettämisohjelmisto. Käytettävyystestausta varten yrityksen olisi hyvä hankkia vähintään videokamera tai mahdollisesti varmistaa oma testaukseen soveltuva tila yrityksen tiloissa.

Uuden käyttöliittymän toteuttaminen tuo yhden olennaisen ongelman yritykselle. Kun toimintajärjestelmään luodaan uusia ominaisuuksia, osa niistä täytyy toteuttaa erikseen sekä vanhaan että uuteen käyttöliittymään. Uuden käyttöliittymän kehittäminen nähtiin joka tapauksessa paremmaksi vaihtoehdoksi, kuin lähteä korjaamaan vanhaa

käyttöliittymää. Korjaaminen olisi vaatinut huomattavasti enemmän resursseja ja mahdollisesti pysäyttänyt koko yrityksen tuotekehitystiimin työskentelyn pitemmäksi ajaksi.

Mikäli yritys päättää prototyypin pohjalta panostaa vahvasti mobiilitoteutukseen, on sen hankittava erilaisia mobiililaitteita testausta varten. Lisäksi olisi hyvä hyödyntää BrowserStack-palvelua, joka tarjoaa erilaisia testiympäristöjä verkkosovelluksen testaukselle. Kokonaisuudessaan opinnäytetyöprojekti vietiin loppuun onnistuneesti, ja saavutettiin hyvin sille asetetut tavoitteet. Lopputuloksena syntyi jopa odotukset ylittänyt mobiilioptimoidun käyttöliittymän prototyyppi.

Lähteet

201Proof 2012. Mobile-Friendly Web Design and Mobile-Optimized Web Design Are Not the Same. Luettavissa: http://www.201proof.com/blog/mobile-optimized-web-design-vs-mobile-friendly-web-design. Luettu: 22.11.2012.

AngularJS 2012a. Understanding the Controller Component. Luettavissa: http://docs.angularjs.org/guide/dev_guide.mvc.understanding_controller. Luettu: 20.11.2012.

AngularJS 2012b. Homepage. Luettavissa: http://angularjs.org/. Luettu: 20.12.2012.

AngularJS 2012c. Directive. Luettavissa: http://docs.angularjs.org/guide/directive. Luettu: 20.12.2012.

AngularJS 2012d. Overview. Luettavissa: http://docs.angularjs.org/guide/overview. Luettu: 20.12.2012.

AngularJS 2012e. Unit Testing. Luettavissa: http://docs.angularjs.org/guide/dev_guide.unit-testing. Luettu: 20.12.2012.

Atkins Jr 2012. A Word About CSS4. Luettavissa: http://www.xanthir.com/b4Ko0. Luettu: 25.10.2012.

Barnum, Carol M. 2010. Usability Testing Essentials : Ready, Set...Test. Morgan Kaufmann. San Francisco.

Bhaskaran, V. & LeClaire, J. 2010. Online Surveys for Dummies. Wiley Publishing. Indiana.

Bibeault, B. & Katz, Y. 2008. jQuery in Action. Manning. Greenwich.

Blodget, H. & Cocotas, A. 2012. THE FUTURE OF DIGITAL [SLIDE DECK]. Luettavissa: http://www.businessinsider.com/future-of-digital-slides-2012-11?op=1. Luettu: 13.12.2012.

Boag, P., Friedman, V., Hodge, S., Inchauste, F., Jovanovic, J., Heilmann, C., Chapman, C., Bowen, R., Follett, A. & Snell, S. 2011. Smashing Magazine. Professional Web Design. The Best of Smashing Magazine. John Wiley & Sons. Chichester.

Bos, B. 2011. The CSS Standardization Process. Luettavissa: http://www.w3.org/Style/2011/CSS-process. Luettu: 5.12.2012.

Brinck, T., Gergle, D. & Wood, S. 2002. Usability for the Web. Morgan Kaufmann. San Francisco.

Burke, B. 2009. RESTful Java with JAX-RS. O'Reilly Media. Sebastopol.

Butcher, S. 2012. The Reality of Enterprise Mobility and BYOD. Avanade Blog. Luettavissa: http://69.64.69.185/mobility-2/the-reality-of-enterprise-mobility-and-byod/. Luettu: 5.12.2012.

Carver, M. 20120. The Responsive Web. MEAP. Manning. Greenwich.

Compass. Homepage. Luettavissa: http://compass-style.org/. Luettu: 28.12.2012.
Can I use... 2012. HTML5. Luettavissa: http://caniuse.com/#cats=HTML5. Luettu: 30.9.2012.

Colyer A. 2012. The New Application Architectures. Katsottavissa: http://www.youtube.com/watch?v=8WCSfTE-X38. Luettavissa: http://springone2012.s3-us-west-1.amazonaws.com/SpringOne2GX_2012_Keynote-Day-2_Final.pdf. Katsottu: 19.10.2012.

Coyier, C. 2012a. Poll Results: Popularity of CSS Preprocessors. Luettavissa: http://css-tricks.com/poll-results-popularity-of-css-preprocessors/. Luettu: 28.12.2012.

Coyier, C. 2012b. Musings on Preprocessing. Luettavissa: http://css-tricks.com/musings-on-preprocessing/. Luettu: 28.12.2012.

Cucumber 2012. Kotisivu. Luettavissa: http://cukes.info/. Luettu: 6.12.2012.

Dashorst, M. & Hillenius, E. 2009. Wicket in Action. Manning. Greenwich.

De Leeuw, E., Hox, J. & Dillman, D. 2008. International Handbook of Survey Methodology. Lawrence Erlbaum Associates. New York.

EmberJS 2012. EmberJS MVC. Luettavissa: http://emberjs.com/guides/ember_mvc/. Luettu: 10.11.2012.

Extreme Programming 1999. Acceptance Tests. Luettavissa: http://www.extremeprogramming.org/rules/functionaltests.html. Luettu: 30.12.2012.

Frain, B. 2012. Responsive Web Design with HTML5 and CSS3. Packt Publishing. Birmingham.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1994. Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley. Boston.

Gartner 2011. Gartner Says Worldwide Enterprise IT Spending to Reach $2.7 Trillion in 2012. Luettavissa: http://www.gartner.com/it/page.jsp?id=1824919. Luettu: 30.11.2012.

Hogan, B. 2010. HTML5 and CSS3. Develop with Tomorrow's Standards Today. The Pragmatic Programmers. Raleigh / Dallas.

Horch, J. 2003. Practical Guide to Software Quality Management. 2. painos. Artech House Books. Norwood.

Indvik, L. 2012. How Important Is a Mobile-Optimized Site for Your Business? Luettavissa: http://mashable.com/2012/10/10/mobile-site-small-business/. Luettu: 5.12.2012.

Ingram, M. 2010. Mary Meeker: Mobile Internet Will Soon Overtake Fixed Internet. Luettavissa: http://gigaom.com/2010/04/12/mary-meeker-mobile-internet-will-soon-overtake-fixed-internet/. Luettu: 30.11.2012.

Keith, M. & Schincariol, M. 2009. Pro JPA 2. Mastering the Java Persistence API. Apress. New York.

Kennedy, A. & de León, I. 2011. Pro CSS for High Traffic Websites. Apress. New York.

Koskinen, J. 2005. Käytettävyystestaus. Luettavissa: http://www.cs.uta.fi/usabsem/luvut/13-Koskinen.pdf. Luettu: 13.12.2012.

Krug, S. 2006. Don't Make Me Think! 2. painos. Peachpit. Berkeley.

La Counte, S. 2012. Build your own app. Huron Street Press. Chicago.

Lakso, J. 2012. Opinnäytetyö. Mobiilioptimoidun käyttöliittymän prototyypin vaatimukset ja määritys.

Layon, K. 2012. Mobilizing Web Sites – Develop and Design. Peachpit Press. Berkeley.

LESS. Homepage. Luettavissa: http://lesscss.org/. Luettu: 28.12.2012.

McWherter, J. & Gowell, S. 2012. Professional Mobile Application Development. John Wiley & Sons. Chichester.

Mikowski, M. & Powell, J. 2012. Single Page Web Applications. Manning. Greenwich.

Mozilla 2012a. @media. CSS Reference. Luettavissa: https://developer.mozilla.org/en-US/docs/CSS/@media. Luettu: 6.12.2012.

Mozilla 2012b. JavaScript Overview. Luettavissa: https://developer.mozilla.org/en-US/docs/JavaScript/Guide/JavaScript_Overview. Luettu: 5.1.2012.

Netherland, W., Weizenbaum, N., Eppstein, C. & Mathis, B. 2012. Sass and Compass in Action. MEAP. Manning. Greenwich.

Nielsen, J. 2000. Jakob Nielsen's Alertbox, March 19, 2000. Luettavissa: http://www.useit.com/alertbox/20000319.html. Luettu: 15.11.2012.

Olson, M. 2012. Optimized vs Responsive Design: Learning More About Your Mobile Website. Luettavissa: http://biznik.com/articles/mobile-friendly-vs-mobile-optimized-vs-responsive-design-learning-more-about-your-mobile-website. Luettu: 23.11.2012.

Osmani, A. 2012a. Journey Through The JavaScript MVC Jungle. Luettavissa: http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/. Luettu: 5.11.2012.

Osmani, A. 2012b. Learning JavaScript Design Patterns. O'Reilly Media. Sebastopol. (Luettavissa: http://addyosmani.com/resources/essentialjsdesignpatterns/book/. Luettu: 11.12.2012.)

Pilgrim, M. 2010. HTML5. Up and Running. O'Reilly Media. Sebastopol.

Richardson, L. & Ruby, S. 2007. RESTful Web Services. O'Reilly Media. Sebastopol.

Rubin, J. & Chisnell, D. 2008. Handbook of Usability Testing. 2. painos. Wiley. Indianapolis.

Santa Maria, J., 2004. Grey Box Methodology. Luettavissa: http://v3.jasonsantamaria.com/archive//2004/05/24/grey_box_method.php. Luettu: 10.12.2012.

SeleniumHQ 2012a. Kotisivu. Luettavissa: http://seleniumhq.org/. Luettu: 6.12.2012.

SeleniumHQ 2012b. Selenium IDE. Luettavissa: http://seleniumhq.org/projects/ide/. Luettu: 6.12.2012.

Singh, Y. 2012. Software Testing. Cambridge. New York.

Stylus. Homepage. Luettavissa: http://learnboost.github.com/stylus/. Luettu: 28.12.2012.

Taloussanomat 2012. Älypuhelinten myynti kasvanut 57 prosenttia. Luettavissa: http://www.taloussanomat.fi/kauppa/2012/11/02/alypuhelinten-myynti-kasvanut-57-prosenttia/201241238/12. Luettu: 5.12.2012.

Terrill, B. 2012. Nine Ways to Improve User Experience in Mobile Design. Luettavissa: http://css-tricks.com/nine-ways-to-improve-user-experience-in-mobile-design/. Luettu: 14.12.2012.

Terrill, B. & Sherrett, J. 2012. 50 Ways to Please Your Customers: A guide to mobile web design best practices. Mobify.

TNS Gallup 2011. Joka neljännellä suomalaisella käytössään älypuhelin. Luettavissa: http://www.tns-gallup.fi/index.php?k=14714. Luettu: 29.11.2012.

Tullis, T. & Albert, W. 2008. Measuring the User Experience. Morgan Kaufmann. San Francisco.

Walls, C. 2011. Spring in Action. 3. painos. Manning. Greenwich.

Walton T. 2011. CSS3 vs. CSS: A Speed Benchmark. Luettavissa: http://coding.smashingmagazine.com/2011/04/21/css3-vs-css-a-speed-benchmark/. Luettu: 5.12.2012.

Wroblewski, L. 2010. Touch Target Sizes.
Luettavissa: http://www.lukew.com/ff/entry.asp?1085. Luettu: 14.12.2012.

Luottamukselliset liitteet

Liite 1. Esimerkkipätkiä geneerisestä abstract CRUD -toteutuksesta.

Liite 2. Kyselyn kysymysten avaus

Liite 3. Kyselyn tulokset ja analysointi

Liite 4. Vaatimusmääritysdokumentti osa 1

Liite 5. Vaatimusmääritysdokumentti osa 2

Liite 6. Käytettävyystestauksen analysointi tablettinäkymässä

Liite 7. Käytettävyystestauksen analysointi älypuhelinnäkymässä

Liite 8. Tehtävät muutokset ja parannukset

Liite 9. Loppuraportti