Kalle Markkanen

Haasteet intranet portaalin käyttöliittymän luomisessa

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikka
Insinöörityö
14.11.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Kalle Markkanen Haasteet Intranet-portaalin käyttöliittymän luomisessa 50 sivua 14.11.2010
Tutkinto	Insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaaja	Lehtori Kari Aaltonen Sovellusarkkitehti Markus Aukeala

Insinöörityön tavoitteena oli tutkia haasteita käyttöliittymän rakentamisessa intranet-portaaliin ja muodostaa uudelleenkäytettäviä komponentteja ja hyviä käytäntöjä tulevia asiakasyrityksen projekteja varten. Hyvien käytäntöjen ja uudelleenkäytettävien komponenttien oli tarkoitus tehostaa kehitysprosessia asiakasyrityksen tulevissa projekteissa. Sovelluskehitysalustana toimi Microsoft Sharepoint 2010. Käyttöliittymän rakentamiseen käytetyt tekniikat olivat ASP.NET-alustasta tuttuja perustyylisivuja, sivunasetteluita, JavaScript-komponentteja ja CSS-määrityksiä.

Projektitiimillä ei ollut aikaisempaa kokemusta ohjelmistoprojekteista Sharepoint-2010 alustalla, sillä tuote oli tullut vastikään markkinoille. Projektin aikana kävi ilmi, että käyttöliittymän muokkaaminen Sharepoint-projekteissa saattaa olla hyvin työläs prosessi. Sovelluksen tukemien internet-selaimien määrää jouduttiin karsimaan, koska käyttöliittymän saattaminen yhtenäiseksi jo kolmella internet-selaimella osoittautui haastavaksi.

Insinöörityön lopputuloksena syntyi uudelleenkäytettäviä perustyylisivuja, CSS-määrityksiä sivunasetteluita ja JavaScript komponentteja. CSS-määrityksistä muodostui tietynlainen kehys, jonka avulla käyttöliittymän ulkonäön muokkaaminen sujuu vaivattomasti. Uudelleenkäytettävien käyttöliittymäkomponenttien ansiosta asiakasyritys voi selviytyä tulevaisuuden projekteista pienemmillä kustannuksilla käyttöliittymän osalta. Projektitiimin osalta kehittäjät saivat runsaasti uusia kokemuksia ja taitoja ohjelmistokehityksestä Sharepoint-2010 alustalle. Projekti myös osoitti, että raskas räätäöinti Sharepoint-alustalle asiakkaan toiveiden mukaan asettaa projektitiimille suuria haasteita.

Avainsanat	Käyttöliittymä, ASP.NET, Sharepoint

Metropolia

Abstract

Author Title Number of Pages Date	Kalle Markkanen Challenges in creating user interface for intranet portal 50 pages 14.11.2011
Degree	Bachelor of Engineering
Degree Programme	Engineer
Specialisation option	digital media
Instructor	Kari Aaltonen Principal Lecturer Markus Aukeala Software Architect

Aim of this thesis was to explore the challenges of user interface building of an Intranet-portal, and a re-usable components and best practices for future projects for the client company. The best practices and reusable components are designed to enhance the company's customer development process for future projects. The platform for application development was Microsoft Sharepoint 2010. User interface construction techniques used were familiar from the ASP.NET platform such as master pages, page layouts, JavaScript and CSS.

The project team had no previous experience in software projects in Sharepoint 2010 platform, because the product had come to the market recently. During the project it became clear that the user interface editing for Sharepoint projects may be a very laborious process. We had to cut the down the number of web browsers supported by the application, because the user interface customization for just three web browsers turned out to be a challenge.

The result of engineer thesis emerged as master pages, CSS files, page layouts, JavaScript-components. CSS styles formed a certain kind of framework which enables the user interface appearance editing is effortless way. Reusable user interface components, enables the customer to the company can survive for future projects at a lower cost for the user interface. The project team of developers got a lot of new experiences and skills in software development for Sharepoint 2010 platform. The project also showed that heavy customizing for Sharepoint platform sets big challenges for the project team.

Keywords	User interface, ASP.NET, Sharepoint

Sisällys

Lyhenteitä ja käsitteitä

Ajax — Asynchronous javascript and xml on joukko tekniikoita, joita hyödyntämällä saadaan aikaan interaktiivisempi web-sovellus.

C# — Microsoftin .NET -ympäristöä varten vuonna 2000 lanseerattu ohjelmointikieli.

Codebehind

ASP.NET -teknologian käyttämä tiedosto, jossa määritellään kontrollien toiminnallisuus.

Http-käsittelijä

ASP.NET-tekniikka, joka palauttaa dataa http-pyynnön mukaisesti, tavanomaisesti kyselymerkkijonon perusteella.

Käyttäjäkontrolli

ASP-NET-tekniikan kontrolli, joka mahdollistaa useiden palvelin-kontrollien ryhmittelyn yhdeksi kontrolliksi ja niiden toiminnallisuuden muokkaamisen.

Web-osa — on ASP.NET -kontrolli, jota käytetään yleensä sisällön esittämiseen. Web-osa antaa käyttäjän muokata web-osan ominaisuuksia, kuten ulkonäköä ja sisältöä.

XSL — XML-pohjainen kieli, jolla voidaan määritellä ja renderöidä ulkoasu XML-dokumenteille

1 Johdanto

Insinöörityöni tavoitteena on kartoittaa käyttöliittymäsuunnitteluun ja sen tekniseen toteutukseen liittyviä haasteita ja etsiä niille ratkaisumalleja, joita asiakasyritys voi hyödyntää tulevissa Sharepoint -projekteissa. Insinöörityöni yhteydessä käsitellään asiakasyritykselle tuotettua intranet-portaalia. Intranet-portaali on yrityksen sisäinen työkalu, joka auttaa informaation välitystä, tiedonhallintaa ja tiedonhakua. Intranet toteutetaan Microsoft Sharepoint 2010 alustaa hyväksikäyttäen. Lopputuloksen tulisi vastata käyttöliittymän osalta mahdollisimman lähelle graafikon luomaa ohjeistusta. Graafinen ohjeistus pitää sisällään kuvat sivujen -asetteluista sekä portaalin typografian. Työkaluina käyttöliittymän toteutukseen toimii Microsoft Visual Studio 2010 ja Sharepoint Designer. Käytettyjä tekniikoita ovat aspx-sivut, html ja css. Ohjelmointikielinä toimivat C# ja JavaScript. JavaScriptin osalta käytettiin jQuery-kirjastoa.

Sharepoint 2010 on Microsoftin vuonna 2010 julkaisema web-alusta. Se on suosittu isokokoisten yritysten intranet- ja ekstranet -ratkaisuihin sekä dokumenttien hallintaan. Se pitää sisällään esimerkiksi web-sivujen julkaisujärjestelmän, sisällönhallinan, hakuominaisuudet ja sosiaaliset verkostoitumistyökalut. Sharepointin erityisominaisuuksia ovat lukuisat etukäteen määritellyt web-sivut, edistykselliset hakuominaisuudet, keskitetty tiedostonhallinta ja integroituminen Microsoft Office 2010 -tuotteiden kanssa.

Työn tilasi suomalainen ohjelmistoyritys Digia oy. Digia on toiminut vuodesta 1997 alkaen, ja se työllistää yli tuhat henkilöä Suomessa, Ruotsissa, Norjassa, Venäjällä, Kiinassa ja USA:ssa. Koska Sharepoint 2010 on melko uusi tuote, ei yrityksellä ollut paljon kokemusta vastaavista projekteista. Vähäinen kokemus näkyi käytännössä siten, että ongelmatilanteissa käyttöliittymän suhteen ei ollut montaa henkilöä joiden, puoleen kääntyä. Projektin toteutusvaiheessa pidettiin päivittäin kokous, jossa käsiteltiin tilannetta ja havaittuja ongelmakohtia. Projektin loppuvaiheessa järjestettiin katselmus, johon osallistui kehittäjät ja asiakkaan edustajat. Katselmuksen jälkeen käsiteltiin asiakkaan määrittämät muutostoiveet.

2 Käyttöliittymäsuunnittelu

2.1 Käytettävyys

Käyttöliittymän suunnittelu on hyvin pitkälle viety, jopa tieteenä määritelty osa-alue. Käyttöliittymäsuunnittelu ei rajoitu pelkästään web-sovelluksiin, vaan sitä harjoitetaan kaikessa, missä vallitsee ihmisen ja koneen välinen vuorovaikutus. Haasteita web-sovelluksen käyttöliittymän suunnittelussa asettavat monet tekijät. Monet haasteista riippuvat täysin sovelluksen lähtökohdista, kuten informaation määrästä, mutta ylivoimaisesti suurimman haasteen asettaa ihmisluonnolle ominainen tapa hakea tietoa internet-sivustolta. Ihmisluonteelle ominainen tapa hakea tietoa web-sivulta voidaan kiteyttää lausahdukseen ”älä pakota minua ajattelemaan” [1, s. 11].

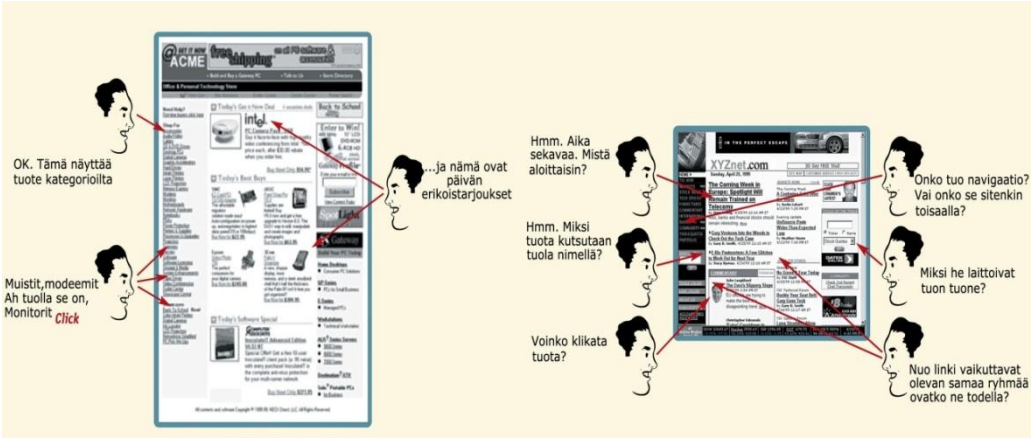
Käyttöliittymän pääasiallinen tavoite on hyvä käytettävyys [1, s. 5]. Hyvän käytettävyyden luominen asettaa samalla haasteen, jonka ratkaiseminen ei suinkaan ole helppoa. Luotaessa käyttöliittymää web-sovellukseen tulee ensisijaisesti ottaa huomioon ihmisluonne. Mikäli sovelluksen käytettävyys on heikko, käyttäjä todennäköisesti turhautuu ja etsii parempaa ratkaisua. Mikäli käyttäjä ei intranet-portaalin tapauksessa löydä helposti etsimäänsä tietoa, hän todennäköisesti vain lähettäisi sähköpostia esimiehelleen tai muulle henkilölle jolta haluttu tieto saattaisi löytyä. Koska Intranet-portaalia käytetään työajalla, käyttäjillä harvemmin on aikaa tutkia läpi navigaation syövereitä halutun tiedon löytämiseksi. Hyvän käytettävyyden avaintekijöitä on antaa käyttöliittymän mukailla ihmiselle ominaista tapaa silmäillä sisältöä, sen sijaan että ihminen lukisi systemaattisesti kaiken sivulla tarjotun tiedon.

Etusivulla on hyvin tärkeä merkitys web-sovelluksessa. Etusivu antaa ensivaikutelman sovelluksesta ja sen tulee tarjota navigaation kautta selkeät hierarkiat saatavilla olevaan informaatioon. Haasteeksi etusivun toteuttamisessa Intranet-portaalissa osoittautuu informaation määrä. Navigaatio tulee pitää selkeänä, eikä käyttäjälle saa aiheuttaa hämmennystä navigaatiosyövereillä. Laaja määrä informaatiota täytyy pystyä jaottelemaan hierarkkisesti selkeisiin pääkategorioihin, jotta tiedon hierarkia on helposti hahmotettavissa. Web-sovellusten käyttöliittymien etusivut ovatkin alkaneet omaksua sanomalehdistä tuttua tapaa jakaa tietoa palstoihin ja kokonaisuuksiin. Sanomalehdet ovat erinomainen esimerkki vuosisatojen mittaan hioutuneesta käyttöliittymästä. Informaation pääkategorioita on korostettu ja lukijan on helppo hahmottaa tarjolla ole-

van tiedon hierarkia. Lukija on myös jatkuvasti tietoinen siitä, missä osiossa sanomalehteä hän kulloinkin on.

Kuten aiemmin tuli todettua, käytettävyys on olennaisessa osassa käyttöliittymän suunnittelua ja samalla yksi suurimmista haasteista. Jotta voidaan luoda hyvän käytettävyyden omaava web-sovellus, täytyy ensin tutustua ihmisen tapaan etsiä tietoa internet-sivulta. Käyttöliittymäsuunnittelussa tavoite on se, että sovelluksen käyttäjän ei tarvitse kuluttaa aikaa ajattelemiseen. Web-sovelluksen näkökulmasta tämä tarkoittaa sitä, että käyttäjä on jatkuvasti tietoinen, mikä on hänen sijaintinsa, mistä hän voi aloittaa, mistä hän löytää haluamansa ja miksi linkit on nimetty niin kuin on.

Yleispätevää ratkaisua käytettävyyden ongelmaan ei ole, mutta hyviä lähestymistapoja on. Jotta sovelluksen käyttäjä olisi jatkuvasti tietoinen sijainnistaan, tulee navigaatiossa korostaa sijaintia rohkeasti. Vaikka sijainnin korostus erottuisi räikeästi sivustolta, on se parempi vaihtoehto kuin se, että käyttäjällä on epäselvyyttä omasta sijainnistaan sovelluksen syövereissä. Web-sovelluksen etusivulle tultaessa tulisi välittömästi vallita tietoisuus siitä, mille sivustolle on saavuttu. Yleisin ja hyväksi havaittu tapa toteuttaa tämä on sijoittaa esimerkiksi yrityksen logo tai tunnuslause sivuston vasempaan tai oikeaan ylänurkkaan. Sivuston omistajan korostus tulisi olla ensimmäisiä asioita, jotka osuvat käyttäjän silmään. Näin sovelluksen käyttäjälle syntyy heti tietoisuus siitä mihin hän on saapunut.



Kuva 1. Kaksi etusivua [1, s. 12].

Kun käyttäjä on saapunut sovelluksen etusivulle, alkaa hän todennäköisesti etsiä tietoa. Kuvassa 1 on esimerkki kahdesta internet-sivusta, toinen on helposti hahmotettavissa

ja toinen ei. Kuvassa vasemmalla puolella olevassa internet-sivussa on selkeästi erottuva visuaalinen hierarkia ja navigaatio on helposti hahmotettavissa. Kuvan oikealla puolella olevassa internet-sivussa tilanne on päinvastainen. Tiedon etsimistä varten informaatio jaetaan navigointiin ja hakukenttiä käytetään apuna tiedon löytämiseen. Tiedon löytämistä varten tulee informaatio jaotella palstoihin sanomalehden tapaan. Lisäksi navigaation tulee mallintaa informaation hierarkiaa niin, että kokonaisuudet ovat helposti hahmotettavissa.

Intranet-portaalin tapauksessa etusivun informaatio jaettiin omiin palstoihin, kuten uutisiin, blogeihin, sosiaalisiin status-päivityksiin. Kun informaatio on jaettu palstoihin ja navigaatio hierarkkisiin kokonaisuuksiin, tulee erotella selkeästi mitä käyttöliittymässä on tarkoitus pystyä napsauttamaan ja mitä ei [1. s. 14]. Hyvä tapa erotella napsautettavat elementit on esimerkiksi alleviivata linkit ja tehdä esimerkiksi napsautettavista kuvista nappien näköisiä. Käyttöliittymän elementtien nimeäminen on tärkeää toteuttaa selkeästi, sillä epäselvät nimet elementeissä aiheuttavat vain hämmennystä loppukäyttäjälle.

Kuva 2. Nappi käyttöliittymässä [1, s. 14].

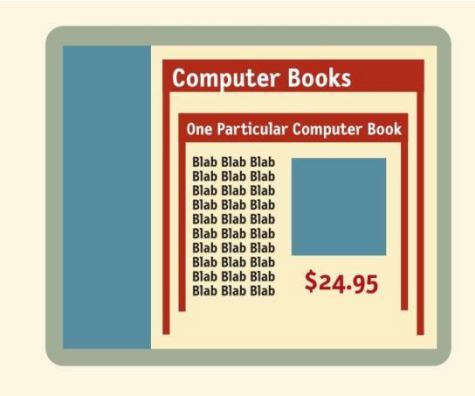
Näennäisesti pienillä asioilla on iso vaikutus käyttöliittymän käytettävyyteen. Kuvassa 2 on havainnollistettu, kuinka käyttöliittymän elementtien ulkoasulla voidaan vaikuttaa käytettävyyteen. Esimerkiksi esitäytetty lomake, lomakkeiden syötteiden reaaliaikainen tarkistus ja hakukenttien hakualueet parantavat sovelluksen käytettävyyttä huomattavasti. Automaattisella syötteiden tarkistuksella tarkoitetaan toimintoa, joka antaa välittömästi palautetta siitä, oliko annettu syöte oikeassa muodossa vai ei. Näin ei tarvitse painaa lomakkeen lähetys nappia todetakseen, että jotkin tiedot eivät olleet syötetty ohjelman edellyttämällä tavalla. Hyvä esimerkki huonosta käytettävyydestä ovat perin-

teiset html- lomakkeet, jotka eivät ole esitäytetty tai eivät sisällä reaaliaikaista syötteiden tarkistusta.

2.2 Visuaaliset elementit

Ihmiselle ominainen tapa silmäillä internet-sivustoja ja hakea tietoa asettaa haasteita käyttöliittymän visuaaliselle suunnittelulle. Toisaalta, kun tiedostetaan, miten ihminen hakee tietoa ja silmäilee internet-sivuja, voidaan sitä käyttää hyväksi visuaalisessa suunnittelussa. On olemassa tiettyjä ihmisten olennaisia ominaisuuksia huomioivia nyrkkisääntöjä, jotka auttavat visuaalisessa suunnittelussa. Näitä sääntöjä ovat visuaalinen hierarkia, napsautettavien elementtien erottuvuus, ”kohinan” minimointi ja sivujen hajottaminen selvästi erottuviin osioihin [1. s. 31].

Visuaalisessa hierarkiassa on pohjimmiltaan sama idea kuin informaatioarkkitehtuurin tavalla jakaa tietoa hierarkkisesti. Visuaalisen hierarkian luominen auttaa ihmistä hahmottamaan sivulla esiintyvien elementtien suhteet toisiinsa, eli mikä on tärkeää ja mikä vähemmän tärkeää. Sanomalehdessä on hyvin erottuva visuaalinen hierarkia. Otsikot on isolla kirjaimilla kirjoitettu ja teksti jaettu palstoihin artikkelien mukaan. Hyvin tehtyjen internet-sivujen visuaalinen hierarkia saattaakin muistuttaa hyvin läheisesti sanomalehden visuaalista hierarkiaa.



Kuva 3. Käyttöliittymän visuaalinen hierarkia [1, s. 32]

Kuvassa 3 näkyy hyviä visuaalisen hierarkian käytäntöjä. Kyseiset käytännöt ovat tuttuja sanomalehtien etusivuilta. Parhaiten visuaalisen hierarkian tärkeys erottuu, kun katsoo internet-sivua, joka on toteutettu lukijalle tuntemattomalla kielellä. Mikäli vieraalla

kielellä toteutetusta sivusta pystyy päättelemään esimerkiksi päänavigaation, sekundaarisen navigaation, osiot ja elementtien hierarkian, on visuaalinen hierarkia toteutettu hyvin.

Ihmisten etsiessä informaatiota internet-sivuilta lähes poikkeuksetta on kyse linkkien takana olevasta informaatiosta. Ihmiset toisin sanoen etsivät linkkejä, joiden he arvelevat johtavan etsimänsä tiedon lähteille. Mikäli internet-sivulla ei ole selkeästi eroteltu, mikä on napsautettavaa ja mikä ei, se johtaa nopeasti käyttäjän turhautumiseen. Normaalin tekstiosion tulisi erottua selkeästi linkeistä, esimerkiksi värin avulla. Toinen hyvä tapa erottaa napsautettava teksti on alleviivaus. Napsautettavista kuvista tulisi tehdä mahdollisimman paljon napin tai näppäimen näköisiä. Tällöin ihminen tekee luonnostaan nopeasti johtopäätöksen, että kyseistä kuvaa voi napsauttaa.

Taustakohinan minimointi on tärkeää huomioida visuaalisessa suunnittelussa. Taustakohinalla tarkoitetaan kaikkea, mikä häiritsee ihmisen tiedon etsintää. Taustakohinaa internet-sivuilla ovat esimerkiksi välkkyvät mainokset ja informaation lukemista vaikeuttavien graafisten elementtien käyttö. Informaation lukemista häiritseviä elementtejä ovat esimerkiksi liian paksut ja tummat väliviivat listaelementtien väleissä.

2.3 Informaatioarkkitehtuuri

Informaation määrä web-sovelluksissa kasvaa jatkuvasti. Intranet-portaaleissa tiedon määrä on perinteisesti suuri. Yritys pyrkii tuomaan työntekijöilleen kaiken tarpeelliseksi katsomansa informaation työntekijöiden saataville keskitetysti. Suuri tiedon määrä asettaa haasteen käyttöliittymän informaatioarkkitehtuurille. Informaatioarkkitehtuurilla tarkoitetaan tiedon jäsentämistä palvelun käyttötarkoitusta tukevalla tavalla [2, s. 51]. Informaatioarkkitehtuurin tehtävänä on jakaa informaatiosisältö siten, että se on helposti hahmotettavissa ja ymmärrettävissä. Parhaimmillaan informaatioarkkitehtuuri on silloin, kun sovelluksesta löytää helposti etsimänsä.

Laaja informaation määrä asettaa informaatioarkkitehtuurisen haasteen. Ihmiset eivät kuitenkaan hahmota tietoa yhdenmukaisella tavalla, joten myös ihmisten eroavaisuudet tuovat oman lisänsä informaatioarkkitehtuurin määrittelyyn. Esimerkiksi ihmisen ikä, kulttuuri ja koulutus vaikuttavat ihmisen tapaan hahmottaa informaatiota [2, s. 51].

Ihmisillä kuitenkin on tiettyjä informaation hahmottamiseen liittyviä yhteisiä ominaisuuksia. Ihminen hahmottaa maailmaa luonnollisesti luokittelemalla tietoa eri kategorioihin [2, s. 51]. Informaatioarkkitehtuurin yksi tärkeimmistä ominaisuuksista web-sovelluksissa onkin tukea ihmisen luontaista tapaa hahmottaa tietoa.

Informaatioarkkitehtuurin takana on tietomalli, jossa on kuvattuna tiedon rakenteet ja suhteet [3, s. 123]. Tietomallina web-sovelluksissa toimii tavanomaisesti relaatiotietokanta. Päällepäin informaatioarkkitehtuuri on näkymätön, ellei siihen kiinnitä erityistä huomiota. Se konkretisoituu tiedon hierarkian, näkymien ja hakuominaisuuksien kautta. Tiedon hierarkia näkyy siten, että käyttäjä tiedostaa, mitä osaa tietosisällöstä hän kulloinkin katselee. Hyvin toteutettu tiedon hierarkia auttaa tätä kautta tiedon omaksumisessa. Tiedon hakua koskien informaatioarkkitehtuuri jakaa tietoa käyttäjäryhmäkohtaisesti käyttötarkoitukset huomioiden. Käytännössä tämä tarkoittaisi sitä, että tietoa jaettaisiin eri tavalla käyttäjäryhmästä riippuen. Esimerkiksi yrityksen talouden hallinta -ryhmään kuuluvat ihmiset saisivat erilaisia näkymiä ja toimintoja käyttöliittymän kautta kuin tuotantopuolella toimivat ihmiset.

2.4 Asiakasnäkökulma

Web-sovellusten toteuttaminen asiakastyytyväisyys huomioiden asettaa melkoisia haasteita. Sovelluksen loppukäyttäjänä toimivat luonnollisesti ihmiset, joten käyttöliittymään ja toiminnallisuuteen kohdistuvat muutoshalut ovat hyvin tavanomaisia, vaikka sovellus olisikin täysin spesifikaation mukainen. Muutostarpeet saattavat olla pieniä tai isoja. Muutostarpeiden luonteeseen vaikuttavat monet seikat loppukäyttäjien koulutuksesta itse sovelluksen ominaisuuksiin [4]. Intranet-portaali-projektissa muutostarpeita tuli lukuisia liittyen käyttöliittymän yksityiskohdista toiminnallisuuteen.

Sovelluksen toimintalogiikka on hyvin yleinen tekijä, johon muutostarpeet kohdistuvat. Sovellus voi vastata täysin spesifikaatiota, vaikka muutostarpeita ilmenee. Sovelluksen ollessa tuotannossa loppukäyttäjät tulevat ajatelleeksi, voisiko jonkin toiminnon tehdä kenties hieman yksinkertaisemmin tai paremmin tarkoitustaan palvelevaksi. Tällaisten tarpeiden huomioiminen on hyvin hankalaa sovelluksen suunnitteluvaiheessa ja siitä syystä ne ilmenevätkin tavallisesti vasta sovellusta käytettäessä. Sovelluskehittäjän näkökulmasta ohjelman toimintalogiikka saattaa vaikuttaa hyvin toimivalta, mutta lop-

pukäyttäjillä ei välttämättä ole yhtä vahvaa osaamista tietotekniikan osa-alueella, joten näkemykset eriävät.

Käyttöliittymään saattaa kohdistua paljon muutostarpeita jälkikäteen. Asiakas saattaa haluta mitä erikoisimpia muutoksia käyttöliittymään, vaikka käyttöliittymä vastaisi spesifikaatiota täydellisesti [4]. Muutosten taustalla saattaa olla monia tekijöitä. Mikäli sopimuksessa sanotaan, että asiakas saa sellaisen käyttöliittymän kuin haluaa, saattaa muutostarpeiden määrä olla suuri. Muutostarpeiden taustalla näkyy selvästi, millainen asiakasyritys on kyseessä. Mikäli asiakas on esimerkiksi tietotekniikan alalta, projektin johtoryhmä koostuu todennäköisesti tietotekniikasta jokseenkin perillä olevista ihmisistä. Tällöin on epätodennäköistä, että käyttöliittymään kohdistetaan suuria muutoksia pelkästään ”kosmeettisista syistä”, mikäli sovellus vastaa spesifikaatiota. Vastaavasti mikäli asiakkaan johtoryhmä koostuu vähemmän teknisistä ihmisistä, muutosehdotuksia tulee helposti laidasta laitaan [4]. Ohjelmiston toimituksesta sopimusta laadittaessa on järkevää ottaa jo etukäteen huomioon, miten muutostarpeita käsitellään. Mikäli sovellus on verifioitu asiakkaan puolesta ja todettu toimivaksi, kaikista muutoksista tulisi laskuttaa erikseen, mikäli halutaan tehdä taloudellisesti kannattavaa ohjelmistokehitystä [4].

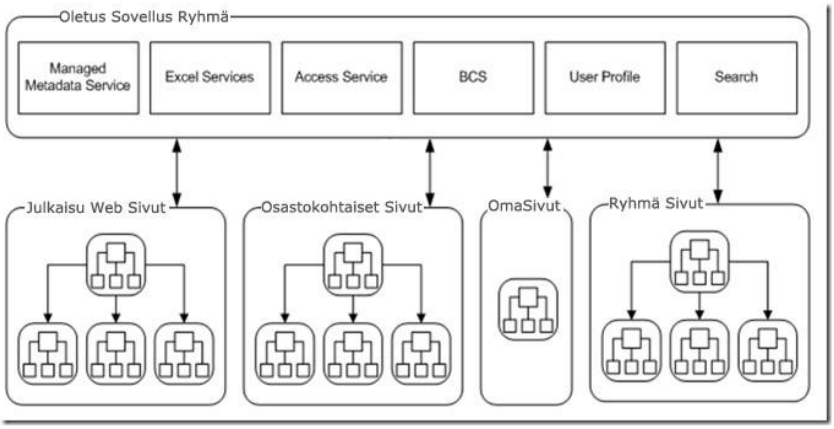
3 Johdatus Sharepoint 2010 -teknologiaan

3.1 Julkaisujärjestelmä sovelluskehityksen alustana

Sharepoint on suunniteltu yrityksen julkaisu- ja tiedonhallintajärjestelmäksi. Se sisältää valmiiksi rakennettuna lukuisia ominaisuuksia, kuten käyttäjäryhmien hallinnan, käyttäjäoikeudet, intranet- ja eksranet-sivujen julkaisujärjestelmän, dokumenttienhallinnan, tiedostonhallinnan, hakuominaisuudet ja työtilat. Sharepointista on hyötyä erityisesti yrityksille, jotka käyttävät ennestään Microsoftin Office -tuoteperheestä tuttuja tuotteita, kuten Exceliä. Microsoft Officen tuoteperheen tuotteilla tuotettuja tiedostoja, kuten Excel- ja Word -dokumentteja sekä Access-tietokantoja, pystyy hyödyntämään sellaisenaan Sharepointissa [5, s. 4].

Sharepoint tarjoaa myös lukuisia ennaltamääriteltyjä web-sivuja intranet- ja ekstranet portaalien hyödynnettäväksi. Näitä web-sivuja ovat esimerkiksi blogit, tiimityötilat ja

oma sivu. Datalähteenä toimivat niin sanotut listat. Listat ajavat saman asian kuin perinteiset tietokantataulut web-sovelluksissa. Datalähteiden hyödyntäminen ei rajoitu pelkästään listoihin. Business Connectivity Serviceä (BCS) hyödyntämällä voidaan muodostaa yhteys muihin tietokantoihin, kuten Microsoft SQL-Server-tietokantoihin [6, s. 20] Myös listoja on määritelty ennalta lukuisia määriä, kuten tiedotteet ja dokumentit. Sisäänrakennettuja ominaisuuksia pystyy muokkaamaan sovelluskehitystyökaluja hyväksikäyttäen.



Kuva 4. Sharepoint-arkkitehtuuri [7].

Sharepoint-sovellus on jaettu sivustokokoelmiin. Kuvassa 4 sivustokokoelmat on rajattu nuolten alapuolelle. Kuvassa on myös esimerkki siitä, miten sisältöä voidaan jakaa sivustokokoelmiin. Kuvassa esiintyvä sivustojako voisi hyvin sopia ison yrityksen tarpeisiin. Jokaisella yrityksen osastolle luotaisiin sivusto ”Osastot” -sivustokokoelmaan. Jokaiselle työntekijälle puolestaan luotaisiin oma sivusto ”Omat sivustot” sivustokokoelmaan. Kuvan yläosassa nuolten yläpuolella on yleisimmin käytettyjä palveluita. Palveluita on lukuisia moneen eri tarkoitukseen ja ne tuovat lukuisia ominaisuuksia toiminnallisuuteen. Esimerkiksi Excel -palvelu mahdollistaa Excel -dokumenttien hyödyntämisen sellaisenaan Sharepoint -sivustoilla. Vastaavasti Business Connectivity Services -palvelu mahdollistaa tiedon hyödyntämisen ulkopuolisista datalähteistä, kuten tietokannoista [8]. Näin ollen ei ole enää tarvetta käyttää aikaa tietokantakohtaisten ratkaisujen luomiseen.

Sivustokokoelmat sisältävät alisivustoja, jotka voivat puolestaan sisältää alisivustoja. Sivustokokoelmat sisältävät myös listat, tyylitiedostot, kuvatiedostot ja JavaScript- tiedostot. Sovelluksen ylläpitäjät voivat luoda uusia sivustokokoelmia itsenäisesti. Yritys

voi tällä tavoin esimerkiksi hajauttaa tietoa sivustokokoelmiin. Käyttäjät jaetaan käyttäjäryhmiin ja niille voidaan antaa oikeuksia. Sivustokokoelmiin voi rajata pääsyoikeudet vain tietyille käyttäjäryhmille. Kaikki ominaisuudet huomioiden se tarjoaa huomattavan määrän ennalta määriteltyä toiminnallisuutta web-sovellukseksi.
Uudelleenkäytettävän käyttöliittymän luominen on olennainen osa sovelluskehitysprojektia web-sovellusten osalta. Jotta voidaan luoda uudelleenkäytettävä käyttöliittymä, on hyvä tietää kuinka sovelluskehitys Sharepoint -alustalle tapahtuu.

Uudelleenkäytettävyydellä tarkoitetaan sitä, että jo kertaalleen luotuja käyttöliittymäelementtejä voidaan käyttää uudelleen tulevissa projekteissa. Ei riitä, että osaa toteuttaa sovelluksen perustyylisivun, sivunasettelut ja tyylitiedostot. Jotta tuote saadaan toimitettua asiakkaan ympäristöön, tulee siitä tehdä niin sanottu wsp, joka asentuu ja päivittyy helposti asiakkaan testi- ja tuotantoympäristöön. Ennen paketin muodostamista tulee käyttöliittymän elementtejä varten luoda omat moduulit ja ominaisuudet (featuret), jotta käyttöliittymän elementit saadaan asennettua korrektilla tavalla. Myös web-mallit (web-template) tulee luoda, jotta voidaan määritellä, mitä perustyylisivua, sivunasetteluita ja web-osia sivut tulevat käyttämään. Web-malleilla on lisäksi olennainen rooli sivustohierarkian luomisessa.

Perustyylisivut, sivunasettelut ja tyylitiedostot ovat käyttöliittymän perusta. Perustyylisivut ja sivunasettelut ovat peräisin ASP.NET-teknologiasta. Ne on suunniteltu yhtenäistämään käyttöliittymän ulkoasua [9, s. 720] Niiden muokkaaminen sujuu parhaiten käyttämällä erillistä ohjelmaa nimeltä Sharepoint Designer. Sharepoint Designer mahdollistaa perustyylisivuissa, sivunasetteluissa tai tyylitiedostoissa tehtyjen muutosten näkemisen välittömästi web-sivun päivittämisellä. Sharepoint Designer on huomattavasti intuitiivisempi tapa muokata käyttöliittymää verrattuna Visual Studioon. Kun tarvittavat muutokset käyttöliittymän elementteihin on tehty, ne voidaan kopioida lopulta Visual Studioon. Projektista tehdään Visual Studiossa wsp-paketti, joka sisältää kaikki käyttöliittymän ja toiminnallisuuden komponentit.

Datalähteet ovat olennainen osa web-sovellusten toiminnallisuutta. Datalähteinä tavanomaisesti toimivat tietokannat. Sharepointin datalähteinä toimivat listat. Listat perustuvat tietokantoihin, mutta ne toimivat hieman eri tavalla kuin perinteiset tietokannat. Relaatiotietokantoihin verrattuna yksi lista vastaa tietokantataulua. Listat eivät

kuitenkaan tarjoa yhtä monipuolisia informaation suunnittelumalleja kuin relaatiotietokannat. Esimerkiksi datan kyselyssä käytettävistä liitoslausekkeista listat eivät tue kaikkia relaatiotietokannoista tuttuja liitoslausekkeita [10, s. 122] Listaan voi syöttää dataa ennalta määriteltyihin sarakkeisiin. Listoja voi luoda käyttöliittymästä käsin, Sharepoint-designerilla tai Visual Studiossa. Täydellinen kontrolli listojen luontiin saadaan vain Visual Studiota hyväksikäyttäen. Listoista voi hakea dataa niin sanottuja CAML-kyselyjä tai LINQ-lausekkeita hyödyntäen. Listoista tuotua dataa on myös mahdollista tyylitellä XSL-kieltä hyväksikäyttäen. Intranet- projektissa XSL-tekniikkaa käytettiin tyylittelemään uutislistauksia.

3.2 Moduulit ja ominaisuudet

Tiedostojen toimitukseen tarvitaan moduuleja. Moduulit ovat keino määritellä, mitä elementtejä provisioidaan sivustokokoelmiin omaisuuksien avulla [11]. Moduulien elements.xml-tiedostoissa määritellään, mitä tiedostoja moduuli sisältää. Moduulin sisältävät tiedostot voivat olla esimerkiksi web-osia, sivunasetteluita, tyylitiedostoja tai perustyylisivuja. Moduulien elements.xml-tiedostoissa voidaan myös määritellä esimerkiksi moduulissa tulevien web-osien omaisuuksia. Tyypillisesti erityyppisiä elementtejä varten luodaan omat moduulit. Intranet-projektissa käyttöliittymän elementtejä varten luotiin omat moduulit perustyylisivuja, sivunasetteluita ja tyylitiedostoja varten. JavaScriptit ja käyttäjä-kontrollit (user-control) eivät tarvinneet omaa moduulia, sillä ne toimitettiin käyttämällä hyväksi ns. kartoitettuja kansioita (mapped folders). Kartoitetuilla kansiolla tarkoitetaan tekniikkaa, joilla voidaan toimittaa tiettyjä elementtejä, kuten kuvia ja JavaScript-tiedostoja, ennalta määriteltyyn tiedostopolkuun palvelimen Sharepoint tiedostosysteemissä.

Moduulit ja ominaisuudet ovat hyvin läheisessä vuorovaikutuksissa keskenään. Moduulit sellaisenaan eivät pysty tuomaan sisältämiään elementtejä sivustokokoelmiin. Moduulien sisällön tuomiseksi sivustokokoelmiin tarvitaan ominaisuuksia. Ominaisuudet ovat tapa toimittaa moduulien sisältämät tiedostot sovelluksen käytettäväksi sivustokokoelmissa. Ominaisuuksilla on myös paljon muitakin rooleja kuin toimia moduulien sisällön toimittajana. Ominaisuudet mahdollistavat helpon toiminnallisuuden päivittämisen, toiminnallisuuden versioinnin, toiminnallisuuden aktivoinnin rajatulla sektorilla ja

paljon muuta [12]. Intranet- projektissa ominaisuuksia käytettiin pääasiassa sovelluksen toiminnallisuuden toimittamiseen moduulien avulla ja sovelluksen muokkaamiseen. Ominaisuudet mahdollistavat sovelluksen muokkaamisen ominaisuuksien vastaanottajien kautta (feature receiver). Ominaisuuksien vastaanottajat mahdollistavat koodin ajamisen ennaltamääriteltyjen tapahtumien yhteydessä. Ennaltamäärättyjä tapahtumia ovat esimerkiksi ominaisuuden aktivointi ja deaktivointi. Ominaisuuksien vastaanottajia käytettiin Intranet-portaalissa esimerkiksi poistamaan moduuleissa tuodut tiedostot sovelluksen käytöstä ominaisuuksien deaktivoinnin yhteydessä. Mikäli ominaisuutta halutaan käyttää vain sovelluksen muokkaamiseen ominaisuuksien vastaanottajien kautta, ei tällöin ominaisuuden mukaan tarvitse liittää moduuleja tai muita tiedostoja.

3.3 Tapahtumien vastaanottajat ja ominaisuuksien nitojat

Tapahtumien vastaanottajilla (event receivers) oli olennainen rooli perustyylisivujen yhtenäistämisessä Intranet-portaalissa luoduille uusille sivustoille. Intranet-portaali mahdollistaa käyttäjien luoda uusia sharepointin omia sivuja, kuten blogeja. Kaikilla intranetin sivuilla tulee kuitenkin olla yhtenäinen ulkoasu.

```
17          var web = properties.Web;
18          if (web.WebTemplate.Equals("STS") || web.WebTemplate.Equals("MPS") || web.WebTemplate.Equals("WIKI") |
19              || web.WebTemplate.Equals("ENTERWIKI") || web.WebTemplate.Equals("BDR") || web.WebTemplate.Equals(
20               || web.WebTemplate.Equals("BDR#0") || web.WebTemplate.Equals("OFFILE") || web.WebTemplate.Equals(
21          {
22              web.MasterUrl = "/_catalogs/masterpage/MTVMasterPage/MTVMediaSPTemplate.master";
23              web.CustomMasterUrl = "/_catalogs/masterpage/MTVMasterPage/MTVMediaSPTemplate.master";
24              web.Navigation.UseShared = true;
25              web.Update();
26              //if the created web is a blog, break role inhertiance(to enable commenting for "peruskäyttäjät" g
27              if (web.WebTemplate.Equals("BLOG"))
28              {
29                  foreach (SPGroup group in web.Groups)
30                  {
31                      if (group.Name == "MTV MEDIA peruskäyttäjät")
32                      {
33                          var list = web.Lists.TryGetList("Kommentit");
34                          var readDef = web.RoleDefinitions["Osallistuja"];
35                          var roleAssigment = new SPRoleAssignment(group);
36                          roleAssigment.RoleDefinitionBindings.Add(readDef);
37                          if (!list.HasUniqueRoleAssignments)
38                              list.BreakRoleInheritance(true);
39                          var AssigmentForGroup = list.RoleAssignments.GetAssignmentByPrincipal(group);
40                          AssigmentForGroup.RoleDefinitionBindings.Add(readDef);
41                          AssigmentForGroup.Update();
42
43                      }
44                  }
45          }
```

Koodiesimerkki 1. Perustyylisivun vaihtaminen tapahtuman vastaanottajassa.

Koodiesimerkissä 1 näkyy tapahtuman vastaanottajan toiminta perustyylisivun vaihtamiseksi. Koodiesimerkissä katsotaan, mikä on luodun sivuston sivumalli. Mikäli sivumalli on jokin Sharepointin omista sivuista, tapahtuu perustyylisivun vaihto. Tapahtumien vastaanottajien toimintaperiaate on hyvin yksinkertainen. Tapahtumien vastaanottajat

omaavat lukuisia ennalta määriteltyjä tapahtumia, joiden yhteydessä voidaan ajaa haluttuja toimintoja. Intranet-portaalin käyttöliittymän osalta tapahtumien käsittelijöissä käytettiin tapahtumia, joita kutsutaan aina, kun uusi sivu luodaan sivustokokoelmaan. Tapahtumien käsittelijät tarjoavat myös lukuisia muita tapahtumia monitoroitavaksi.

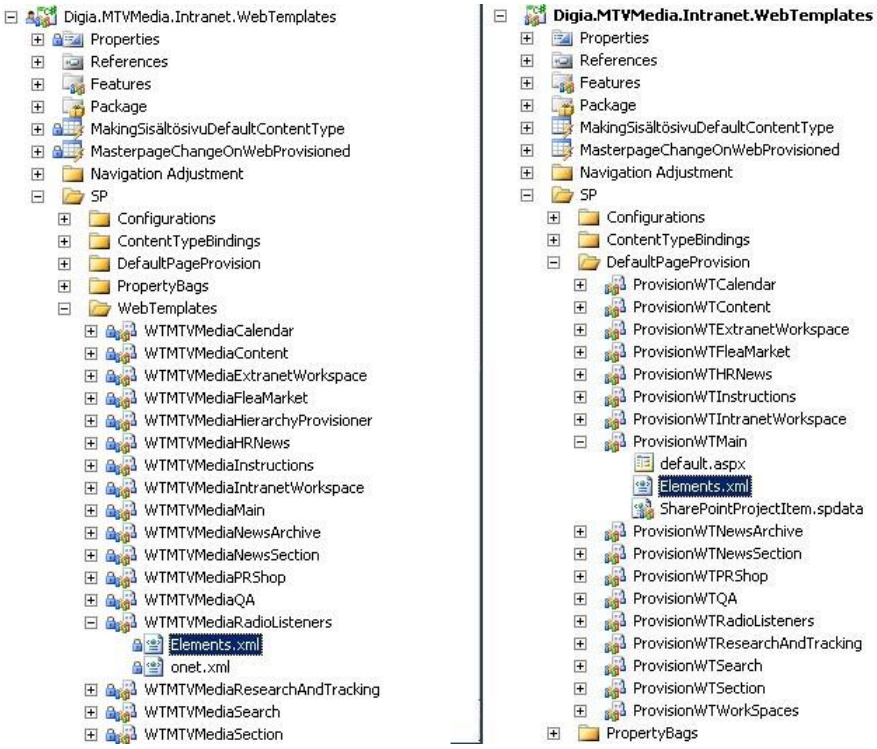
Tapahtumien vastaanottajat eivät kuitenkaan havaitse, mikäli uusi sivu luodaan toiseen sivustokokoelmaan, kuin missä tapahtuman vastaanottaja sijaitsee. Tämänkaltaisia tapahtumia esiintyy esimerkiksi silloin, kun intranetin käyttäjä luo itselleen ”oma sivu” sivuston. Oletuksena omalla sivulla on sitä varten suunniteltu perustyylisivu, joka eroaa huomattavasti intranet-portaalia varten luoduista perustyylisivuista. Jotta perustyylisivu saataisiin vaihdettua omalle sivustolle, tarvitaan ominaisuuksien nitojan (feature stapler) apua. Toisin kuin tapahtuman vastaanottaja, ominaisuuksien nitoja havaitsee, kun sivuja luodaan toisiin sivustokokoelmiin. Ominaisuuksien nitoja toimii siten, että se monitoroi määriteltyjä sivustomalleja. Kun jokin ennalta määritellyistä sivustomalleista luodaan web-sovelluksessa, jossa ominaisuuksien nitoja sijaitsee, kutsutaan automaattisesti ominaisuutta, joka on linkitetty monitoroituun sivustomalliin.

Tapahtuman vastaanottajia ja ominaisuuksien nitojia voidaan hyödyntää myös huomattavasti laajemmin kuin käyttöliittymän perustyylisivujen vaihtamiseen. Intranet-portaalissa tapahtuman vastaanottajia käytettiin myös esimerkiksi käyttäjäoikeuksien muokkaamiseen. Oletuksena blogeja pystyi kommentoimaan vain peruskäyttäjästä korkeampaan käyttäjäryhmään kuuluvat jäsenet. Määrittelyjen mukaan peruskäyttäjilläkin tuli olla oikeus kommentoida blogeja. Ratkaisu löytyi antamalla peruskäyttäjä-käyttäjäryhmälle oikeudet blogin ”kommentit”-listaan uuden blogin luonnin yhteydessä. Tapahtuman vastaanottajaa käytettiin ajamaan käyttäjäoikeuksia muuttavaa koodia blogisivun luonnin yhteydessä.

3.4 Web-mallit

Toiminnallisuuden ja käyttöliittymäelementtien ollessa valmiit tulee olla jokin tapa, miten saadaan kopioitua sovellus muihin ympäristöihin. Muilla ympäristöillä tarkoitetaan esimerkiksi asiakkaan tuotanto- ja testiympäristöä. Web-mallien avulla sovellus saadaan rakennettua uudestaan muissa ympäristöissä. Käytännössä web-mallit mahdollistavat sivustohierarkian määrittämisen, web-osien sijoittamisen paikoilleen sivunasette-

luihin ja ominaisuuksien toimittamisen sivustokokoelmiin ja sivuihin [13]. Manuaalinen lähestymistapa edellyttäisi sivustohierarkian luomisen, web-osien lisäämisen ja ominaisuuksien aktivoimisen käyttöliittymästä käsin tai powershell-skriptejä hyväksikäyttäen.



Kuva 5. Web-templates -projektin sisältö.

Kuvassa 4 näkyy web-mallit -projekti intranet-portaalin osalta. Kuvasta näkyvät web-mallien käyttöliittymää koskevat tiedostot. Web-mallit sisältävät myös ominaisuudet, jotka tuovat moduulien sisällön sivustokokoelmiin ominaisuuksien aktivoinnin yhteydessä. Web-osien sijoittelut sivunasetteluissa, käytettävissä olevat sivunasettelut ja perustyylisivu määritellään xml-tiedostossa. Käytettävissä olevat sivunasettelut näkyvät parhaiten siten, että luotaessa uusi kustomoitu sivu Intranet-portaalissa käyttäjällä on ennalta määriteltyjä vaihtoehtoja erilaisista sivunasetteluista.

Web-mallit mahdollistavat toiminnallisuuden päivittämisen ominaisuuksien kautta. Vaikka sovellus olisikin valmis, tavanomaisesti ohjelmistoprojekteissa kehitetään toiminnallisuutta myös sovelluksen ylläpitovaiheessa. Ominaisuuksien päivityskehyksen avulla toiminnallisuutta voidaan päivittää uuteen versioon. Käytännössä ominaisuuksien päivitys tapahtuu xml-tiedostoja hyväksikäyttäen tai koodin kautta hyödyntämällä objektimallia. Sharepointista myös julkaistaan päivityspaketteja, jotka edellyttävät toimiakseen tiettyjä kriteereitä. Esimerkiksi sivustojen mallien tulee vastata alkuperäistä

versiotaan melko tarkasti, jotta päivitys onnistuu. Web-mallit tukevat näitä kriteereitä ominaisuuksien avulla ja mahdollistavat uusien korjaus -pakettien ajamisen ongelmitta.

4 Käyttöliittymäsuunnittelu

4.1 Sharepoint käyttöliittymäalustana

Käyttöliittymän luominen Sharepoint-alustalla eroaa joiltakin osin tavanomaisesta web-sovelluksen käyttöliittymän kehittämisestä. Tavanomainen tapa luoda käyttöliittymä web-sovellukseen toimii niin, että sovelluskehittäjät luovat sovelluksen toimintalogiikan ja käyttöliittymän toteuttaja määrittelee ulkoasun, eli sen miltä sovellus näyttää internet-selaimessa. Pääperiaate käyttöliittymän luomisessa eroaa tavanomaisesta mallista siten, olemassa olevaa käyttöliittymää muokataan sen sijasta, että luotaisiin uusi käyttöliittymä. Sharepoint tuo mukanaan kattavan määrän valmiita perustyylisivuja, sivunasetteluita ja tyylitiedostoja. Näitä valmiita tiedostoja käytetään lähtökohtana uutta käyttöliittymää luotaessa. Tyhjältä pöydältä aloittaminen esimerkiksi perustyylisivujen luomisen suhteen ei ole suositeltavaa eikä järkevää. Perustyylisivulla tarkoitetaan tiedostoa, jossa määritetään web-sivun elementtien, kuten navigaation -asemointi.

Valmiita perustyylisivuja, sivunasetteluita tai tyylitiedostoja ei saa muokata, vaan niistä tulee tehdä kopiot. Perustyylisivuja ja sivunasetteluita koskien on olemassa tiettyjä rajoituksia, joita tulee noudattaa. Esimerkiksi perustyylisivuista ei saa poistaa tiettyjä elementtejä, kuten navigaatioita. Elementtien poistamisen sijaan elementtejä voi kuitenkin piilottaa ja tyylitellä tyylitiedostoja hyväksikäyttäen melko vapaasti. Käyttöliittymää ei kuitenkaan kannata lähteä muuttamaan radikaalisti alkuperäisestä asettelustaan. Radikaalit käyttöliittymän muutokset saattavat johtaa käyttöliittymän hajoamiseen esimerkiksi siirryttäessä web-sivun muokkaustilaan.

Virallisesti Sharepoint tukee täysin vain yhtä internet-selainta, Microsoft Internet Exploreria versiosta 7.0 lähtien [14]. Puutteellisesti tuetaan esimerkiksi Mozilla Firefoxia ja Apple Safaria. Internetselain rajoitukset huomioiden Sharepoint-projektia ei kannata lähteä toteuttamaan monelle internetselaimelle, mikäli käyttöliittymää räätälöidään huomattavasti. Intranet-portaalissa pelkästään käyttöliittymän yhtenäistäminen Inter-

net Explorerille ja Firefoxille vei kuukausia, tosin käyttöliittymää muokattiin huomattavasti alkuperäiseen nähden.

4.2 Vaatimusten analysointi

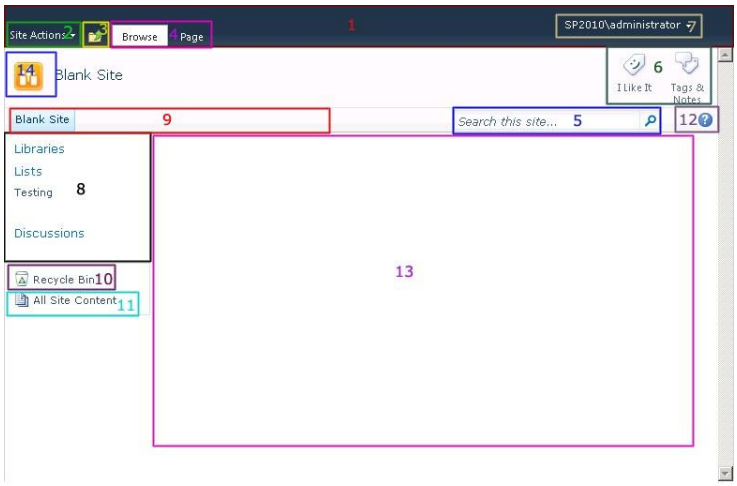
Käyttöliittymän luomisen lähtökohtana on käyttöliittymän määritelmät. Määrittelyssä kerrotaan mitä käyttäjän tulee pystyä tekemään käyttöliittymästä käsin. Määrittelydokumentaatiota seuraa graafiset ohjeistukset. Graafinen ohjeistus on graafikon määrittelemä kokonaisuus käyttöliittymän ulkonäköä koskevista asioista. Käytännössä graafinen ohjeistus ilmenee käyttöliittymää kuvaavina kuvatiedostoina ja typografian määritelmillä.

Käyttöliittymää Sharepointille suunniteltaessa graafiseen ohjeistukseen tulee suhtautua kriittisesti. Graafikoilla harvemmin on teknistä tietämystä siitä, mitä käyttöliittymän takana olevalla sovelluksella on valmiiksi tarjottavana. Esimerkiksi tietyissä käyttöliittymän elementeissä on valmiiksi toteutettu valikko, jonka ulkonäköä voi muokata. Valmiita komponentteja ei suinkaan kannata lähteä keksimään uudestaan, vaan on järkevämpää hyödyntää olemassa olevaa. Intranet-portaalin tapauksessa joitakin kohtia hylättiin alkuperäisestä graafisesta ohjeistuksesta, sillä yksinkertaisempi ratkaisu oli jo olemassa valmiiksi rakennettuna.

Sharepoint-alustalle voi luoda käyttöliittymää kahdella tekniikalla. Tekniikoista käytetään termejä yksinkertainen kustomointi ja edistynyt kustomointi [15, s. 101]. Yksinkertainen tekniikka tarkoittaa olemassa olevan käyttöliittymän elementtien ulkoasun muokkaamista tyylitiedostojen avulla, mutta perustyylisivuihin ei kohdisteta muutoksia. Yksinkertaisella tekniikalla voidaan lähinnä muuttaa elementtien värejä, typografiaa ja piilottaa elementtejä tarpeen mukaan. Edistyneessä tekniikassa myös perustyylisivuihin ja sivunasetteluihin tehdään muutoksia. Perustyylisivujen ja sivunasetteluiden muokkaaminen mahdollistaa hyvin erilaisen ulkoasun toteuttamisen kuin pelkästään yksinkertaista tekniikkaa käytettäessä. Vastaavasti myös käyttöliittymästä tulee virhealttiimpi kuin yksinkertaista tekniikkaa käytettäessä. Intranet-portaalissa käytettiin edistynyttä tekniikkaa käyttöliittymän toteuttamiseen. Yksinkertaisella tekniikalla ei olisi pystytty tuottamaan graafista ohjeistusta vastaavia perustyylisivuja ja sivunasetteluita.

4.3 Valmiit elementit

Oletus-perustyylisivu sisältää lukuisia kontrolleja. Vaatimusmäärittelyiden pohjalta määritellään mitä kontrolleja tullaan käyttämään. Tietyt kontrollit ovat kuitenkin välttämättömiä sovelluksen toiminnallisuuden takaamiseksi. Käyttöliittymän kontrollien sijaintia perustyylisivussa voi muuttaa muokkaamalla perustyylisivua. Kontrollien ulkonäköä voi muuttaa tyylitiedostoja ja JavaScriptia hyödyntäen.



Kuva 6. V4-perustyylisivu, kontrollit numeroituna.

Kuvassa 6 näkyy yksi valmiista perustyylisivuista, v4.master. Kaikki kuvassa numeroidut elementit tulevat valmiina oletus -perustyylisivun mukana. Käyttöliittymän graafisesta suunnittelusta vastaavan henkilön olisi hyvä ensiksi tutustua siihen, miltä käyttöliittymä näyttää oletusarvoisesti. Kun graafikko on huomioinut käyttöliittymän valmiiden kontrollien lähtökohdat, uusi käyttöliittymä on kenties helpommin toteutettavissa. Intranet-portaalin graafisesta suunnittelusta vastaava henkilö ei ollut nähnyt, miltä oletus -käyttöliittymä näyttää. Tämä johti siihen, että graafinen ohjeistus oli joiltakin osin niin vaikeasti toteutettavissa, että siitä oli pakko poiketa.

Taulukko 1. Kuvan 5 toiminnot

1	Nauha	Nauhakontrollia käytetään aktiivisen sivun tai aktiviteetin muokkaukseen.
2	Sivun toiminnot	Pudotusvalikko, jota käyttää pääasiassa autentikoituneet henkilöt navigoidessaan eri toimintoihin.
3	Globaali linkkipolku	Näyttää linkkipolun aktiiviseen sivuun ponnahdusikkunana, hierarkiassa ylimmästä sivusta alkaen.

4	Nauhan kontekstuaaliset paneelit	Paljastaa eri toiminnallisuuksia nauha kontrollista.
5	Haku-alue	Mahdollistaa tiedon hakemisen.
6	Sosiaaliset napit	Mahdollistaa sosiaaliset toiminnot, kuten tykkäyksen ja kommentoinnin.
7	Tervetuloa valikko	Näyttää kirjautuneen käyttäjän nimen ja linkin omalle sivulle, profiiliin ym.
8	Reunanavigaatio	Vertikaalinen navigaatio, joka näyttää linkit sivuihin, jotka ovat hierarkisesti valitun sivun alapuolella.
9	Ylänavigaatio	Ensisijainen horisontaalinen navigaatio sivustolle.
10	Roskakori	Linkki hiljattain poistettujen elementtien näkymään.
11	Kaikki sivuston sisältö	Linkki kaikki sivuston elementit näyttävään näkymään.
12	Apua -ikoni	Linkittyy apu dokumentaatioon.
13	Sisältöalue	Säiliö sivun sisältöä varten.

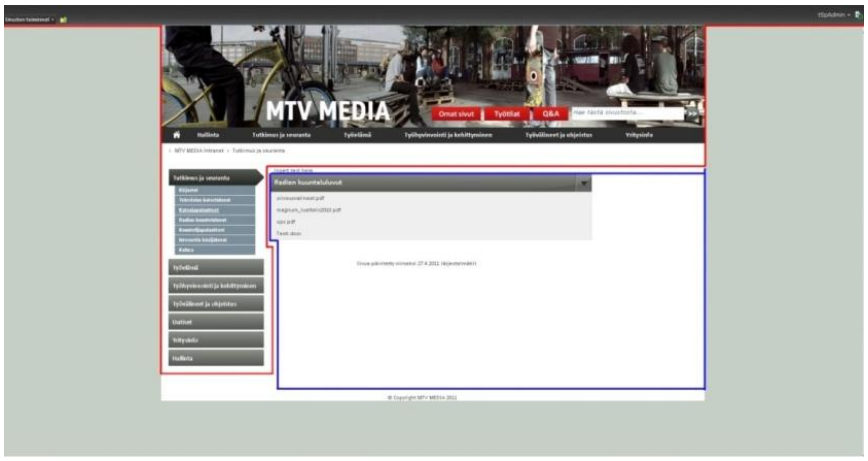
Taulukossa 1 on eriteltynä kuvan 5 numerojen mukaiset käyttöliittymän valmiit elementit. Kaikki taulukossa esiintyvät elementit ovat muokattavissa ulkonäöllisesti ja toiminnallisesti. Ulkonäön muokkaaminen tapahtuu tyylitiedostoilla ja JavaScriptilla, toiminnallisuutta voi muokata käyttöliittymästä käsin tai ohjelmallisesti. Intranet-portaali –projektissa ei käytetty kaikkia taulukossa esiintyviä elementtejä. Piilotimme käyttöliittymästä taulukossa esiintyvät elementit 10 ja 11. Lähes kaikkien käyttöliittymän elementtien ulkonäköä tai toimintaa muokattiin.

Ulkonäöllisesti haasteellisin muokattu elementti oli reunanavigaatio. Reunanavigaation rakenne koostuu html standardin mukaisesta listasta. Internet-selaimien väliset erot aiheuttivat listan näkymisen eri tavalla tyylimääritysten muodostamisen jälkeen. Toiminnallisesti haastavin listan 1 elementti oli ylänavigaatio. Ylänavigaatio näyttää oletusarvoisesti vain samaan sivustokokoelmaan kuuluvat sivut. Määritysten mukaisesti ylänavigaation tuli kuitenkin näyttää samat linkit riippumatta siitä, missä sivustoko-

koelmassa aktiivinen sivu sijaitsee. Ongelman ratkaisemiseksi luotiin lopulta käyttäjäkontrolli ja http-käsittelijä, jotta navigaatio saatiin yhtenäistettyä kaikkien sivustokokoelmien osalta.

4.4 Toteutukseen käytetyt tekniikat

Sharepointin käyttöliittymän toteutukseen käytettävät tekniikat ovat hyvin samankaltaisia verrattuna Microsoftin ASP.NET kehykseen. Käyttöliittymän kokonaisvaltaisesta ulkonäöstä vastaavat perustyylisivut. Perustyylien tarkoitus on yhdenmukaistaa ulkoasu käyttöliittymissä [16, s. 381]. Perustyylisivuissa määritellään pääelementtien, kuten navigaation ja ylätunnisteen sijoittelu. Perustyylisivuja voi olla useita, mikäli se on tarpeen. Web-sovelluksen varsinaisen sisällön asettelu määritellään sivunasetteluissa. Sivunasetteluissa määritellään esimerkiksi palstat web-osia varten. Sivunasettelulla pystyy vaikuttamaan merkittävästi web-sivulla esitettävän sisällön ulkonäköön. Esimerkiksi sivunasettelu, jossa on määritelty neljä vyöhykettä sisältöä varten, näyttää varsin erilaiselta kuin kahden palstan sivunasettelu.



Kuva 7. Perustyylisivu ja sivunasettelu.

Pääperiaate on, että sivunasettelun esittämä sisältö on muuttuvaa perustyylisivujen sisällön ollessa staattinen. Kuvassa 7 nähdään perustyylisivun elementit punaisella rajattuna ja sivunasettelun elementit sinisellä rajattuna. Perustyylisivut eivät vaikuta varsinaisesti siihen, miltä web-sivu tulee näyttämään. Perustyylisivut määrittelevät pääosin, mitä elementtejä perustyylisivulla sijaitsee, elementtien ulkoasu määritellään tyylitiedostoilla. Tämä tarkoittaa perustyylisivujen osalta esimerkiksi, mitä navigaatio elementtejä sivulle sijoitetaan. Perustyylisivut ja sivunasettelut pohjautuvat ASP.NET

teknologiasta tuttuihin aspx-sivuihin. Sivunasettelut käyttävät .aspx-päätteisiä tiedostoja ja perustyylisivut .master-päätteisiä tiedostoja. Aspx-sivut muistuttavat hyvin paljon html-dokumenttia. Varsinaisen sisällön kannalta olennaisimpana erona html-dokumentin ja aspx-sivun välillä on ASP.NET-tekniikkaan pohjautuvat kontrollit ja sivunasetteluiden paikat määrittävät tagit.

```
16  <asp:Content ID="Content1" ContentPlaceHolderId="PlaceHolderMain" runat="server">
17      <SharePoint:CssRegistration ID="MTVCustomWpStyles" runat="server" Name="<% $SPUrl:~sitecollection/Style Library/Styles/MTVCustomWpStyles.css %>" />
18      <script type="text/javascript">
19      $(document).ready(function(){
20          $(".MainLeftZone > table TR:eq(2) .ms-WPBody").css("border", "none");
21          });
22      </script>
23      <table width="100%" ID="mainWebpartPanel">
24          <tr valign="top">
25              <td width="540">
26                  <div class="MainLeftZone">
27                   <WebPartPages:WebPartZone runat="server" FrameType="TitleBarOnly" ID="LeftZone" Title="LeftZone" />
28                  </div>
29              </td>
30              <td width="650">
31                  <div class="MainRightZone">
32                      <WebPartPages:WebPartZone runat="server" FrameType="TitleBarOnly" ID="RightTopZone" Title="RightTopZone" />
33                  </div>
```

Koodiesimerkki 2. Aspx-sivu.

Koodiesimerkissä 2 on näyte aspx-sivun rakenteesta. Aspx-sivuilla käytetään myös täysin standardeja html-tageja, kuten kuvassa näkyviä ”<table>” -tageja. Aspx-sivujen toiminnallisuuden takana on niin sanotut ”code-behind” -tiedostot. Nämä toiminnallisuuden määrittelevät tiedostot toteutetaan jollakin .NET kehyksen tukemista ohjelmointikielistä, kuten C#:llä tai Visual Basicillä. Toiminnallisuutta luotaessa aspx-sivuille kehittäjän tulee olla selvillä aspx-sivun elinkaaresta. Elinkaaren päävaiheita ovat sivun kysely, kyselyyn vastaaminen, alustaminen, lataus, post-komennon käsittely, renderöinti ja purkaminen [17, s. 132].

Taulukko 2. Aspx-sivun elinkaaren vaiheet.

Elinkaaren vaiheen nimi	Selitys
Sivun kysely	Prosessi, jossa määritellään millä tavalla sivun pyyntöön vastataan. Tapa, jolla sivun pyyntöön vastataan riippuu esimerkiksi siitä, onko sivun pyydetyn sivun dataa välimuistissa, vai tuleeko pyydetty sivu renderöidä uudestaan alusta alkaen.
Aloitus	Kyselyä koskevien omaisuuksien määrittäminen, käyttöliittymän kielen määrittäminen
Alustaminen	Sivulla sijaitsevat kontrollit asetetaan manipuloitaviksi ja perustyylisivut teemat mukaan lukien määritetään.
Lataus	Mikäli sivun pyyntö-metodi on post-mallinen, kontrolleja koskeva informaatio ja omaisuudet määritetään.
Post-komennon tapahtumien käsittely	Mikäli kysely on post-tyyppinen ja sisältää tapahtumia, tapahtumien hallitsijoita kutsutaan. Post-datan oikeellisuus (validation) tarkistetaan.
Renderöinti	Kontrollien tila tallennetaan ja kontrollit renderöidään.
Purkaminen	Sivun pyynnön käsitelleet resurssit vapautetaan.

Taulukossa 2 näkyy aspx-sivun elinkaaren vaiheet selityksineen. Käyttöliittymäsuunnittelun osalta aspx-sivun elinkaaren vaiheita hyödynnettiin navigaation yhtenäistämisen toteuttavan käyttäjäkontrollin ja http-käsittelijän yhteydessä. Sharepoint- tai ASP.NET-projekteissa käyttöliittymästä vastaavat kehittäjät ovat harvemmin tekemisissä aspx-sivun elinkaarten vaiheiden kanssa. Elinkaaren vaiheilla on enemmän merkitystä kehittäjille, jotka kehittävät toiminnallisuuden web-sovelluksen taustalla.

Varsinainen sisältö sivunasetteluissa, kuten teksti ja kuvat, tuodaan web-osia tai html-kenttiä hyväksikäyttäen. Web-osa on sisältöä esittävä komponentti, joka voidaan asettaa sivunasetteluun. Web-osan datalähteenä voi toimia esimerkiksi listat. Web-osa an-

taa loppukäyttäjälle mahdollisuuden muokata web-osan ulkonäköä ja toimintoja. Web-osia löytyy valmiiksi määriteltyinä erilaisiin tarkoituksiin. Web-osia voi myös luoda manuaalisesti ohjelmoimalla. Valmiiksi määriteltyjen web-osien toiminnallisuus ei välttämättä riitä vastaamaan sovelluksen määrittelyitä, tällöin web-osa voi olla tarpeen ohjelmoida itse. Sivunasetteluihin voi myös lisätä staattista sisältöä näyttävän rikkaan html-kentän. Web-osien avulla tuotettu sisältö muuttuu tietokannan sisällön mukaisesti, mutta html-kenttien sisältö pysyy sellaisenaan, ellei sitä muuteta manuaalisesti.

Tyylitiedostot ovat merkittävä osa web-sovelluksen käyttöliittymän toteutusta. Sharepoint ei ole tässä suhteessa poikkeus. Tyylitiedostoilla voidaan vaikuttaa web-sivun ulkonäköön hyvin paljon. Perustyylisivujen elementtien osalta tyylitiedostoilla voidaan määritellä elementtien lopullinen sijainti sivulla, mittasuhteet, värimaailma, typografia ja paljon muuta. Tyylitiedostoja voi määritellä sivustokohtaisesti web-sovellusta varten. Esimerkiksi jokaiselle perustyylisivulle voidaan määritellä oma tyylitiedosto. Tyylitiedostojen koko saattaa helposti paisua satojen rivien mittaisiksi, joten on helpompaa tehdä erillisiä tyylitiedostoja eri tarkoituksia varten. Tyylitiedostojen hajauttaminen helpottaa myös sovelluksen kehittämistä ja ylläpitämistä. Hajautetuilla tyylitiedostoilla kehittäjien ei tarvitse etsiä tiettyjä tyylimäärityksiä yhdestä satojen rivien pituisesta tyylitiedostosta.

```
/*----  General Layout  ----*/

P {
   margin: 10px;
   background-color: Yellow;
   color: Green;
   font-family: helvetica,arial;
   font-size: 12px;
   line-height: 18px;
}

div.logo {
    float: left;
    height: 50px;
    width: 200px;
}
```

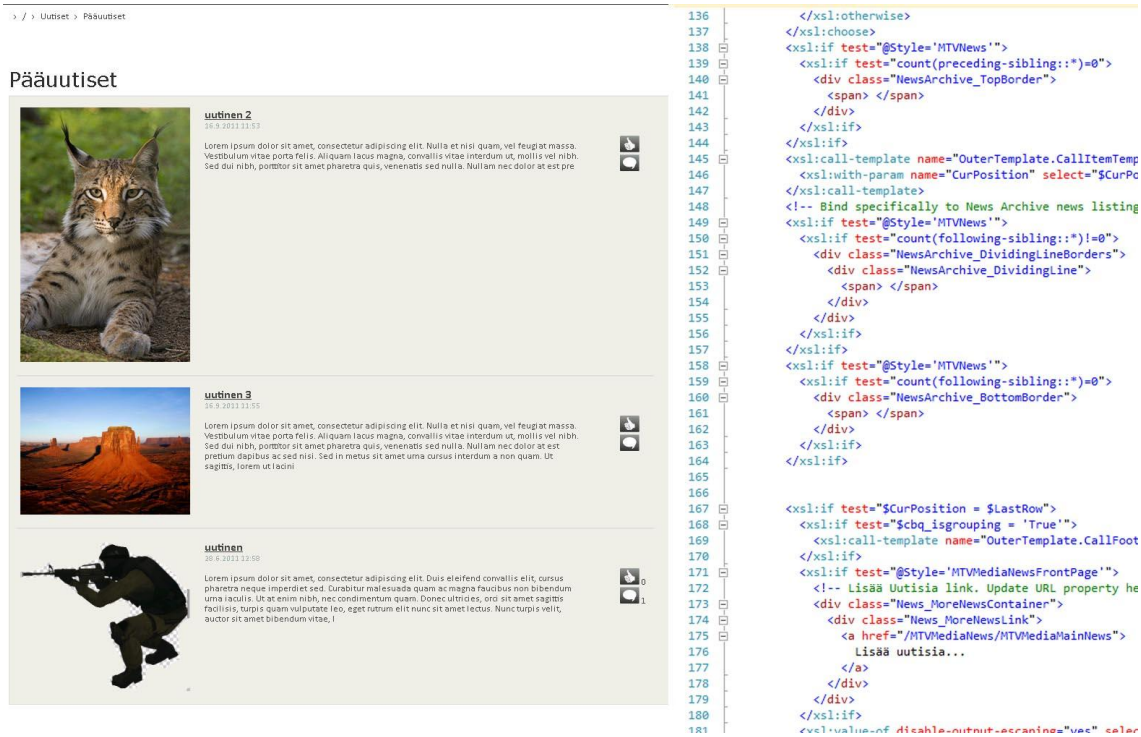
Koodiesimerkki 3. Tyylimäärittelyjä

Koodiesimerkissä 3 nähdään esimerkki CSS-tyylimäärittelyistä. CSS-tyylimääritykset ovat syntaksiltaan helposti omaksuttavia. CSS-tyylimäärityksiä on olemassa valtava määrä. Niillä on mahdollista tehdä globaaleja, kaikkia html-elementtejä koskevia määrityksiä tai vain tiettyihin html-elementteihin kohdistettuja määrityksiä esimerkiksi html-elementin id:n perusteella.

JavaScript on yhä isommassa merkityksessä nykyaikaisia web-sovelluksia. JavaScriptilla voidaan lisätä web-sivun interaktiivisuutta monella tapaa. JavaScriptilla voidaan toteuttaa esimerkiksi käyttöjärjestelmistä tuttuja pehmeästi avautuvia valikoita, leijumisefektejä ja nappiefektejä. JavaScriptilla oli olennainen rooli myös intranet-portaalin käyttöliittymän luomisessa. Tiettyjä elementtejä perustyylisivuissa ei pysty muokkaamaan tyylitiedostoilla tarpeeksi, joten JavaScript auttaa tässäkin asiassa. Esimerkiksi tietyt käyttöliittymäelementit sisältävät kuvatiedostoja, ja mikäli kuvatiedosto halutaan vaihtaa, tarvitaan kuvatiedoston polun uudelleenmäärittelyyn JavaScriptia.

4.5 Listanäkymät

Listadatan hyödyntämiseen käytetyt tekniikat käyttöliittymän osalta eroavat huomattavasti muista käyttöliittymän muodostamiseen käytetyistä tekniikoista, kuten tyylitiedostoista. Kun listojen sisältämän datan esitysmuotoa käyttöliittymässä halutaan muokata, puhutaan lista -näkymistä. Listoista on mahdollista luoda erilaisia näkymiä web-osilla, jotka hyödyntävät listojen dataa. Listadataa hyödyntäviä web-osia ovat esimerkiksi sisältökysely-web-osa ja listanäkymä-web-osa. Samasta listadatasta voi luoda hyvinkin erilaisia näkymiä XSL-tekniikkaa hyödyntämällä. Käytännössä listadatan esitysmuotojen muokkaaminen tapahtuu olemassa olevia XSL-tiedostoja muokkaamalla [18]. Listoja hyödyntävät web-osat hyödyntävät oletusarvoisesti valmiiksi määriteltyjä XSL -määrittelyjä, samaan tapaan kuin perustyylisivut hyödyntävät oletuksena Sharepointin omia CSS-tyylitiedostoja.



Kuva 8. Uutislistaus.

Kuvassa 8 näkyy XSL-tekniikalla muokattu uutislistaus. Kuvassa 7 on näkyy myös osa XSL-koodista, jota käytettiin tekemään tyylimäärityksiä uutisnostoille. Samaa tekniikkaa käytettiin myös antamaan erilaisia vaihtoehtoja etusivun uutisnostoille. Intranet-portaalissa XSL-tekniikalla määriteltiin uutisille kokonaisuudessaan viisi eri esitystapaa. XSL-tekniikalla määriteltiin elementtien koot ja sijainnit toisiinsa nähden. Kun elementit olivat oikeilla paikoillaan oikean kokoisina, käytettiin CSS-tekniikkaa tyylittelemään elementtien ulkoasu. Käytännössä tämä tapahtui siten, että XSL-koodissa määriteltiin elementeille CSS-luokat, joita hyödynnettiin tyylitiedostoissa.

5 Käyttöliittymän toteutus

5.1 Toteutusprosessi ja perustyylisivut

Intranet-portaalia varten toteutettiin useita perustyylisivuja, sivunasetteluita, tyylitiedostoja ja JavaScript-tiedostoja. Käyttöliittymän saattaminen valmiiksi oli useita kuukausia kestänyt prosessi. Käyttöliittymää testattiin aktiivisesti sen ensiaskeleista lähtien. Kaikkia käyttöliittymässä tehtyjä muutoksia tuli testata kolmella internet- selaimella. Internet-selaimet olivat Internet Explorer 7, Internet Explorer 8 ja Mozilla Firefox.

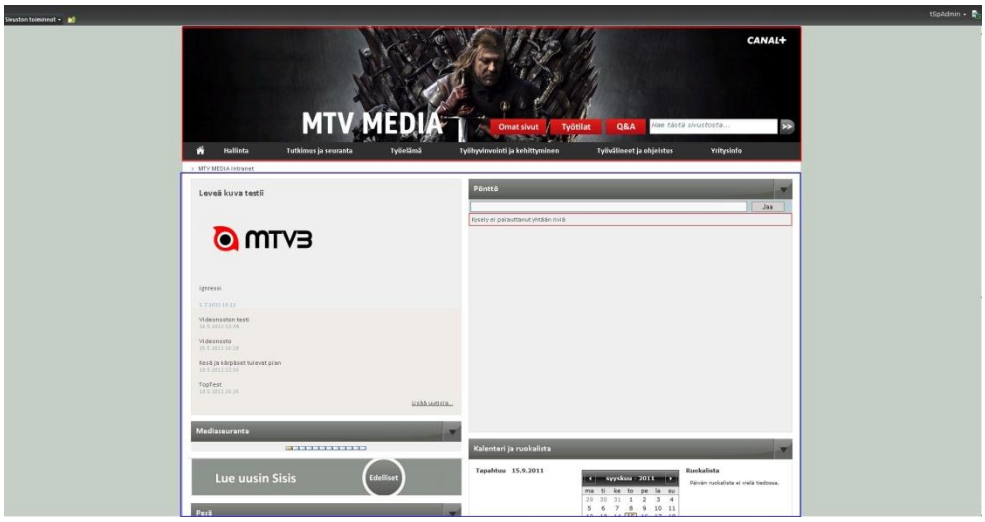
Helpoimmaksi vaiheeksi käyttöliittymän rakentamisessa ilmeni perustyylisivujen ja sivunasetteluiden toteuttaminen. Perustyylisivuille ja sivunasetteluille ei tarvinnut tehdä suuria muutoksia ensimmäisten versioiden jälkeen. Haastavimmaksi osuudeksi muodostui tyylitiedostojen määrittäminen ja ylänavigaation yhtenäistäminen sivustokokoelmien välillä. Tyylitiedostojen määrittely olisi ollut hyvin suoraviivainen toimenpide, mikäli tuettavia internet-selaimia olisi ollut vain yksi. Lopputuloksen näyttäminen samalta kaikissa tuetuissa selaimissa osoittautui melkoiseksi haasteeksi. Yhtenäisen navigaation toteuttaminen edellytti hyvin pitkälle räätälöityä ratkaisua.
JavaScriptien määrittely käyttöliittymää varten sujui suoraviivaisesti. Ainoastaan otsake-kuvia vaihtava kuvakaruselli vaati astetta enemmän syventymistä. Kustomoidun JavaScriptin rooli intranet-portaalissa oli lähinnä kuvakarusellin toteuttaminen ja käyttöliittymän muokkaus niiltä osin, kun se ei ollut mahdollista tyylimäärityksiä hyväksikäyttäen. JavaScriptilla toteutettiin nappiefektit, otsakkeen kuvakaruselli ja käyttöliittymäelementtien muokkausta niiltä osin, kun se ei ollut tyylitiedostoilla mahdollista.

Perustyylisivujen toteutus aloitettiin muokkaamalla kopiota yhdestä Sharepointin omista perustyylisivuista. Perustyylisivujen muokkaus oli käytännössä perinteistä div-taittoa Sharepointin erityisominaisuudet huomioiden. Erityisominaisuus, johon perustyylisivuja muokatessa tuli kiinnittää huomiota oli se, että tiettyjä elementtejä ei- saanut poistaa perustyylisivuista, vaikka niitä ei käyttöliittymään haluaisi. Korrekti tapa päästä eroon ei halutuista elementeistä, oli elementtien piilottaminen tyylitiedoston kautta.

Taulukko 3. Perustyylisivut ja niiden merkitykset.

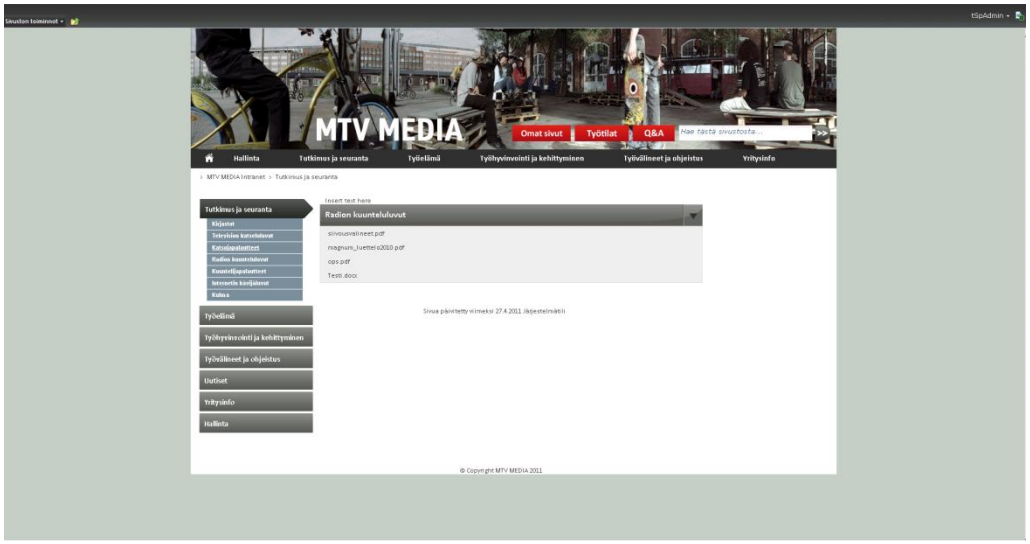
Perustyylisivun nimi	Perustyylisivun tarkoitus
etusivu	perustyylisivu etusivua varten
osio	perustyylisivu osio sivustoja varten
oma sivu	perustyylisivu oma-sivusto sivuja varten
systeemi	perustyylisivu niitä sivuja varten, joissa tulee olla sama system/custom –perustyylisivu
Sharepoint sivu	perustyylisivu Sharepointin omia sivuja, kuten blogeja varten
Työtilat	perustyylisivu työtiloja varten

Taulukossa 3 on nähtävillä Intranet-portaalia varten tehdyt perustyylisivut tarkoituksineen. Kaikkiaan perustyylisivuja muodostettiin kuusi kappaletta. Intranet-portaalin määrittelyjen mukaan kaikki normaalille käyttäjälle portaalissa näkyvät sivut tulisi noudattaa asiakasyrityksen hyväksymää graafista ilmettä ja typografiaa. Tämä tarkoitti käytännössä sitä, että mitkään muut kuin hallintasivustot eivät saisi näyttää Sharepointin omilta muokkaamattomilta sivuilta. Muokkaamattomilla sivuilla on valmiiksi määritelty graafinen ilme, jota haluttiin välttää.



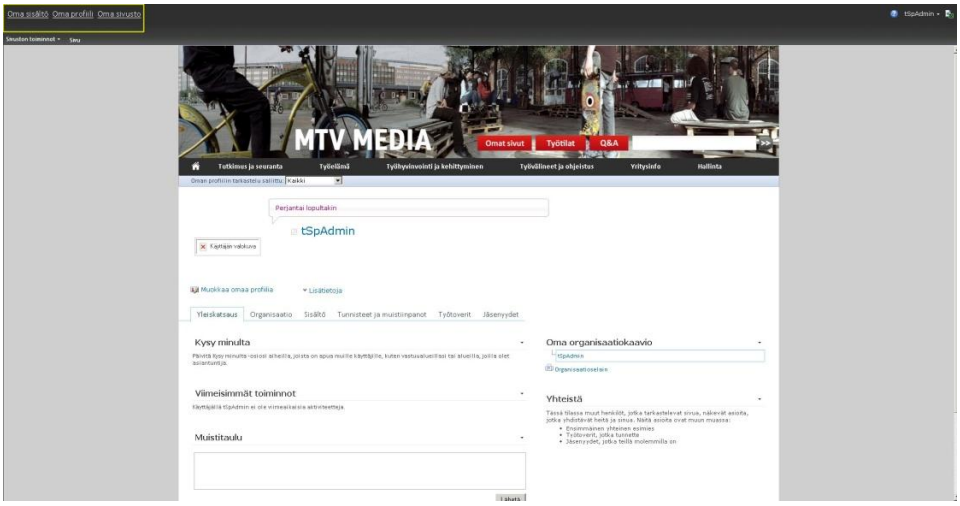
Kuva 9. Intranet-portaalin etusivu.

Kuvassa 9 näkyvässä Intranet-portaalin etusivussa on nähtävillä etusivua varten määritelty perustyylisivu. Perustyylisivussa määriteltiin kaikki kuvan 9 sinisellä rajattujen web-osien ympärillä olevat elementit. Portaalin etusivusta oli tarkoitus saada näyttävän näköinen, mutta myös käytännöllinen. Kuvassa 9 punaisella rajattu otsake- navigaatio kopioitiin kaikkiin perustyylisivuihin. Lisäksi otsaketta varten luotiin niin sanottu kuvakaruselli. Kuvakarusellin tarkoitus on luoda interaktiivisuuden tunnetta sivustolle otsakkeen vaihtuvien taustakuvien avulla. Etusivulla on web-sovelluksissa hyvin tärkeä merkitys. Intranet-portaalin tapauksessa etusivun merkitys korostuu entisestään. Etusivu antaa käyttäjälle ensivaikutelman sovelluksesta. Etusivulta tulee pystyä navigoimaan käyttäjän etsimään paikkaan ja löytämään tarvittavaa tietoa. Intranet-portaalin etusivu näyttää web-osien avulla käyttäjien status-päivityksiä kommentteineen, uusimpia uutisia, kalenterin tapahtumineen, linkkejä uusimpiin sivustoihin ja paljon muuta. Käyttäjien status-päivitykset ovat sosiaalisesta mediasta tuttu sovellus, jossa käyttäjät voivat esimerkiksi jakaa mietteitään, tiedotteita ja käydä keskustelua toistensa kanssa.



Kuva 10. Osiosivu.

Kuvassa 10 on nähtävillä osio-perustyylisivu. Olennainen ero etusivun perustyyliin nähden on kuvan vasemmassa reunassa näkyvä navigointipaneeli. Navigointipaneelin ulkoasu kustomoitiin täysin tyylimäärityksiä hyväksikäyttäen. Navigointipaneelin tyylimääritykset vaativat huomattavan määrän aikaa toimiakseen täydellisesti kaikilla tuetuilla internet-selaimilla. Navigaatiopaneelin toteuttamiseen käytetyistä tyylimäärityksistä muodostettiin myöhemmin CSS-kehys, joka mahdollistaa navigaatiopaneelin ulkoasun muokkaamisen pienellä vaivalla. Osiosivun tarkoitus on näyttää sisältöä. Osiosivuilta tulee pystyä helposti navigoimaan informaatiohierarkiassa alemmille sivustoille ja ylemmille sivustoille. Reunanavigaatio tarjoaa navigointimahdollisuuden alisivustoille ja ylänavigaatio päätason sivustoille.



Kuva 11. Intranet-portaalin Oma sivusto.

Kuvassa 11 näkyvä oma sivusto eroaa ulkoasultaan huomattavasti muista sivustoista. Huomattavista ulkoasullisista eroista muihin sivuihin nähden perustyylisivu on käyttöliittymän elementeiltään lähes identtinen etusivun perustyylisivuun nähden. Oma sivuston sivunasettelun web-osat tuovat kaiken kuvassa 11 valkoisella taustalla näkyvän sisällön. Oman sivuston navigaatiota kustomoitiin huomattavasti oletusnavigaatioon verrattuna. Kuvassa 10 keltaisella rajatut kolme linkkiä eivät alun perin kuuluneet oma sivuston perustyylisivuun lainkaan. Lisäksi etusivun ja osiosivun perustyylisivuista tutut otsakkeet kopioitiin osaksi oma sivusto perustyylisivua. Oma sivuston tarkoitus on näyttää käyttäjää koskevaa informaatiota, antaa mahdollisuus käyttäjäprofiilin asetusten muokkaamiseen ja antaa käyttäjän hallita informaatiota. Oma sivusto jakaantuu kolmeen sivustoon, joita ovat oma sisältö, oma profiili ja oma sivusto. Oma sisältö antaa käyttäjälle mahdollisuuden hallinnoida keskitetysti käyttäjän Sharepointtiin lataamaa dataa. Oma profiili mahdollistaa käyttäjätietojen muokkaamisen ja oma sivusto näyttää julkisesti käyttäjää koskevaa dataa, kuten työtoverit, jäsenyydet työtiloihin ja muistitaulun. Lisäksi käyttäjällä on mahdollisuus päivittää blogiaan oman sivun kautta. Oma sivusto näyttää myös käyttäjän tietoa käyttäjän työtovereista, organisaatiosta ja jäsenyydet työtiloihin.

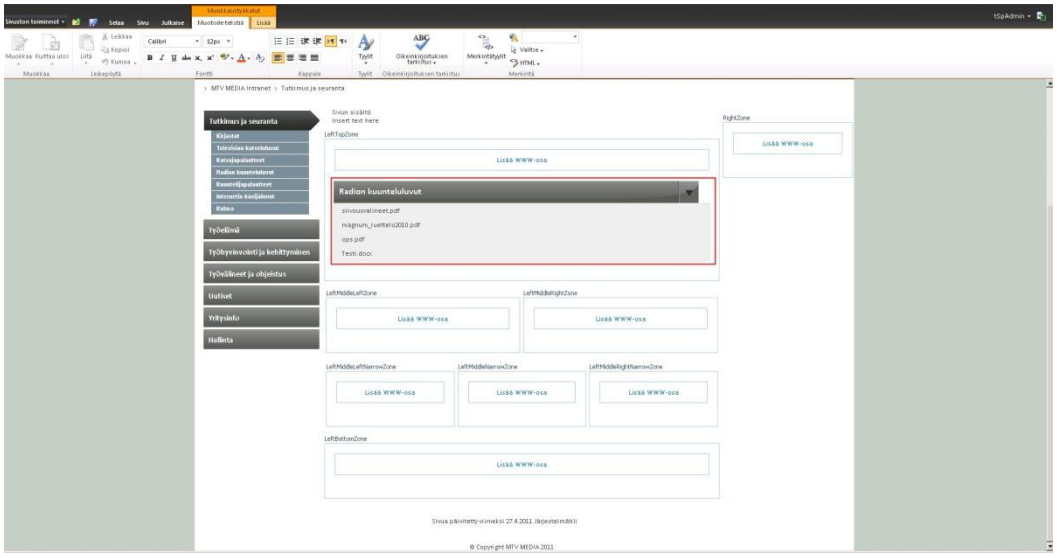
5.2 Sivunasettelut

Kuten luvusta 5.1 kävi ilmi, sivunasetteluilla on huomattava vaikutus sivun ulkonäköön. Sivunasetteluita Intranet-portaalia varten tehtiin kaikkiaan yli kymmenen kappaletta. Sivunasettelut toteutettiin käyttämällä ASP.NET-kehyksestä tuttuja aspx-sivuja ja taulukkotaittoa. Taulukkotaittoon päädyttiin siksi, että sivujen muokkaus-tila toimii varmemmin ilman häiriöitä taulukkotaiton kanssa. Puhtaasti div-taiton ollessa käytössä havaittiin, että tietyt käyttöliittymän elementit saattavat vaihtaa sijaintiaan siirryttäessä sivunmuokkaustilaan. Sivunasettelut saatiin rakennettua melko nopeasti, eikä eroavaisuuksia havaittu tuettujen internet-selainten välillä.

Taulukko 4. Sivunasettelut

Sivunasettelun nimi	Sivunasettelun käyttötarkoitus
etusivu	Intranet-portaalin etusivu. Näyttää ajankohtaisimman sisällön.
osio	Toimii etusivuna alemman tason sivuille, jotka kuuluvat tiettyyn osioon.
sisältö	sisältösivut ovat osiosivuihin liittyviä alemman tason sivuja.
pääuutiset	Listaa uusimmat uutiset
hakutulokset	Näyttää hakutulokset sisällöstä ja ihmisistä
henkilöhakutulokset	Näyttää hakutulokset vain ihmisistä
kysymykset ja vastaukset	Näyttää web-osat, jotka mahdollistavat kysymysten määrittelyn ja vastausten selailut
työtilat	Näyttää työtilat, joihin käyttäjä kuuluu.

Taulukossa 4 on näkyvillä intranet-portaalia varten tuotetut sivunasettelut käyttötarkoituksineen. Sivunasettelut eroavat toisistaan lähinnä web-osia varten luotujen palstojen osalta. Suuri osa sivunasetteluiden esittämästä sisällöstä Intranet-portaalista tulee web-osien kautta.



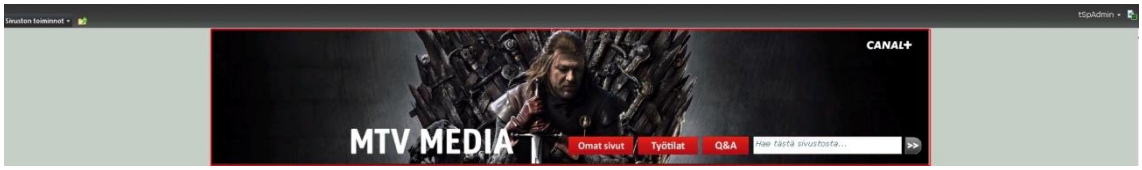
Kuva 12. Osiosivun sivunasettelu.

Kuvassa 12 nähdään osiosivun sivunasettelun määrittämät web-osien palstat. Osiosivun sivunasettelu mahdollistaa web-osien asettamisen monelle palstalle rinnakkain. Kuvassa punaisella rajattu web-osa on asetettu leveimpään mahdolliseen sivunasette-

lun palstaan. Intranet-portaalia varten luotiin lukuisia kustomoituja web-osia sisällön näyttämiseen. Kustomoituja web-osia olivat esimerkiksi kysymykset&vastaukset ja sosiaalisen median sovelluksista tuttu statuspäivityksen mahdollistava web-osa. Osiosivujen sivunasettelut suunniteltiin varsinaista Intranet-portaalin sisältöä varten. Osiosivut mahdollistivat staattisen sisällön syöttämisen ja web-osien asettelun monilla eri palstaleveyksillä. Osiosivunasettelu mahdollistaa hyvin erinäköisten sivujen luomisen useiden eri leveyksien omaavien web-osa alueiden ansiosta. Kuvan oikeassa reunassa näkyvä kapea palsta suunniteltiin lähinnä kuvia esittävää web-osaa varten.

5.3 Tyylitiedostot

Tyylitiedoston toteuttaminen on periaatteessa hyvin suoraviivainen prosessi. Katsotaan perustyylisivusta tai sivunasettelusta elementit identifioivat tunnukset ja määritellään tyylit. Tyylejä määriteltäessä tulee ottaa huomioon resoluutiovaatimukset. Esimerkiksi mobiili-versiota varten tyylimääritykset ovat todennäköisesti erilaiset tietokoneiden näytölle suunniteltuihin tyylimäärityksiin verrattaessa. Intranet-portaalia varten tyylitiedostoja määriteltiin kokonaisuudessaan kymmenen kappaletta**.** Tyylitiedostoilla muokattiin hyvin laajasti perustyylisivujen elementtejä, web-osia, typografiaa ja web-osien tulostamaa sisältöä. Pääperiaate tyylejä muokattaessa oli ylikirjoittaa Sharepointin omat tyylimääritykset.
Kätevimmäksi työkaluksi tyylien muokkaukseen osoittautui Mozilla Firefoxin liitännäinen nimeltä Firebug. Firebug mahdollisti html-elementtien tyylimääritysten muuttamisen interaktiivisesti, CSS-sääntöjen kopioinnin ja jopa JavaScriptin virheiden etsinnän. Ainoa ongelma Firebugin avulla kehitettävissä tyylimäärityksissä oli Internet Explorerin eroavaisuudet.



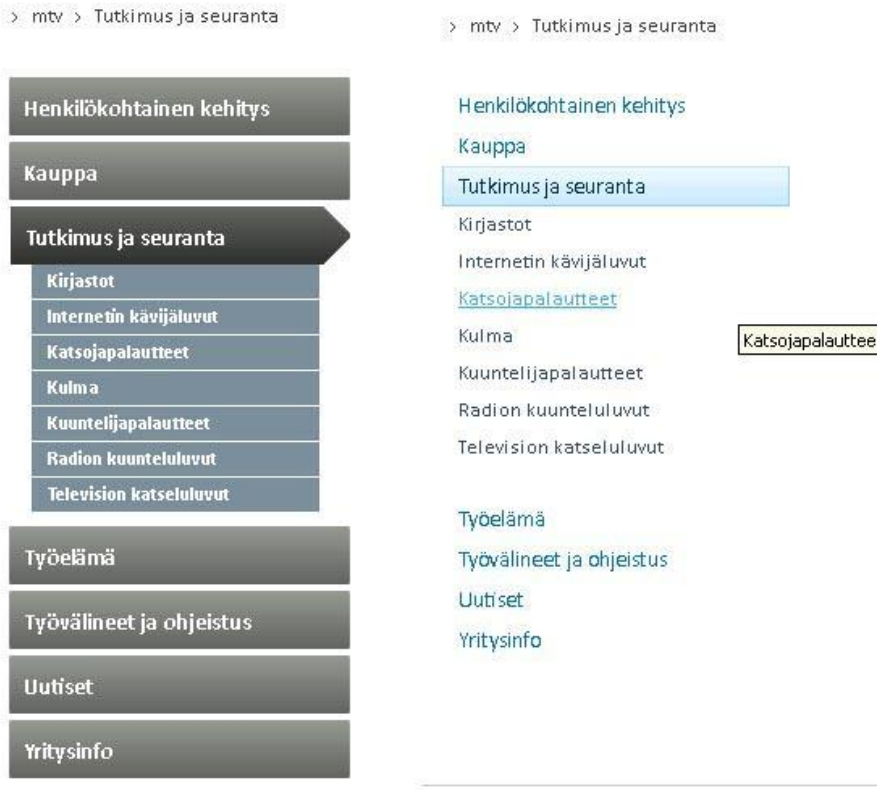
Kuva 13. Perustyylisivun otsake.

Perustyylisivujen tyylitiedostoilla muokattuihin elementteihin kuului otsake, ylänavigaatio, reunanavigaatio ja alatunniste. Kuvassa 13 on nähtävillä perustyylisivun otsake. Otsaketta varten määriteltiin taustakuva, kolme navigaationappia ja hakukenttä. Ot-

sakkeen taustakuvaa varten tehtiin JavaScriptillä Kuvakaruselli luo Intranet-portaaliin eloisan vaikutelman staattiseen taustakuvaan nähden.

Kuva 14. Ylänavigaatio.

Kuvassa 14 nähtävää ylänavigaatiota muokattiin huomattavasti oletusversioon nähden. Kuvassa punaisella rajattuna on nähtävillä, miltä oletusnavigaatio näyttää. Oletus ylänavigaatio korvattiin kokonaan kolmannen osapuolen navigaatioliitännäisellä. Kolmannen osapuolen liitännäistä ylänavigaatioon päätettiin käyttää, sillä kyseisellä liitännäisellä oli helpompi määritellä ylänavigaation alasvetovalikoiden efektit. Ylänavigaation vasempaan reunaan laitettiin kuvassa keltaisella rajattu kuva, jota napsauttamalla voi navigoida Intranet-portaalin etusivulle sijainnista riippumatta.



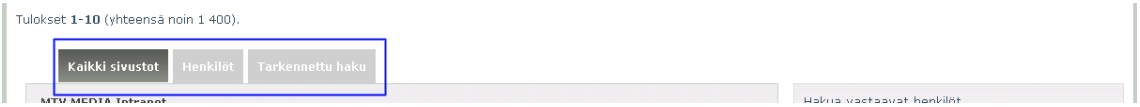
Kuva 15. Reunanavigaatio + web-osa.

Oletusreunanavigaatiota kustomoitiin huomattavasti tyylitiedostojen avulla. Kuvan 15 oikealla puolella näkyy oletus reunanavigaatio ja vasemmalla puolella kustomoitu navigaatio. Reunanavigaation ulkoasun yhtenäistäminen tuetuilla selaimilla osoittautui hyvin haastavaksi tehtäväksi. Jokin tyylimääritelmä saattoi toimia kahdella tuetulla se-

laimella, mutta ei kolmannella. Lopulta oli välttämätöntä joidenkin tyylimääritysten osalta käyttää internet-selain kohtaisia tyylimäärityksiä esimerkiksi Internet Explorer 7:ää varten.

Kuva 16. Oletus-web-osa ja kustomoitu web-osa.

Web-osien oletusulkoasua muokattiin huomattavasti tyylimäärityksiä hyväksikäyttäen. Kuvan 16 yläreunassa on nähtävillä web-osa oletustyylimäärityksillä ja alareunassa kustomoiduin tyylimäärityksin. Kuvassa punaisella rajattu web-osan otsake oli haastavin alue kustomoida web-osien osalta. Tietyillä tyylimäärityksillä web-osan kuvassa vihreällä rajatusta valikkonappulasta ilmestyvä valikko siirtyi ruudun yläreunaan Firefox-internet-selaimella. Lisäksi sisällön asemoiminen kustomoiduissa web-osissa tuotti ongelmia. Lopulta web-osat kuitenkin saatiin toimimaan tuetuissa internet-selaimissa halutulla tavalla.



Kuva 17. Hakutulossivun navigaatio.

Käyttöliittymän ulkoasun puolesta kaikkea tarvittavaa ei kuitenkaan saatu muokattua pelkästään tyylitiedostoja käyttämällä. Tyylitiedostojen vaikutuksen ulkopuolella olivat Sharepointin kontrollien kautta luotujen kuvien tiedostopolut ja tietyt kontrollien luomat elementit, kuten kuvassa 17 sinisellä rajattu paneelinavigaatio. Nämä ongelmatapaukset ratkaistiin käyttämällä hyväksi jQuery JavaScript -kirjastoa. JavaScriptilla muokattiin html-elementtien attribuutteja halutun ulkonäön saavuttamiseksi.

5.4 Javascript ja Ajax

Intranet-portaalissa käytettiin kattavasti hyväksi jQuery JavaScript -kirjastoa. jQuery todettiin huomattavasti helppokäyttöisemmäksi ja tehokkaammaksi ratkaisuksi kuin natiivi JavaScript. Intranet-portaalissa käytettiin hyväksi JavaScriptia nappiefekteihin, kuvakarusellin luomiseen sekä käyttöliittymän muokkaukseen. Lisäksi JavaScriptia käytettiin Client-objektimallin hyödyntämiseksi. Client-objektimalli mahdollistaa palvelinpuolen objektimallia vastaavan mallin hyödyntämisen JavaScriptin avulla. Client-objektimallin käytännön hyötyjä oli esimerkiksi kalenterin tapahtumien hakeminen valitun päivämäärän mukaisesti. JavaScriptilla ja Ajaxilla toteutettujen toiminnallisuuksien etu on siinä, että loppukäyttäjän ei tarvitse odottaa sivun päivittymistä palvelinpuolen dataa hyödynnettäessä.

Kuva 18. Nappi ennen ja jälkeen häilymisefektin.

Kuvassa 18 nähdään kuvakaappaus kuvaelementistä, jota käytettiin nappina. Hiiren kohdistimen liikkuessa napin päälle kuvatiedosto vaihtuu. Perinteisesti nämä niin sanotut häilymis-efektit tehdään CSS-tyylimäärityksiä hyväksikäyttäen. Häilymis-efektillä tarkoitetaan tapahtumaa, jossa hiiden kohdistin tuodaan jonkin html-elementin päälle ja samalla kyseisen html-elementin tyylimääritykset muuttuvat. Häilymisefektiä ei kuitenkaan voida tehdä kuvatiedostoille, mikäli kuvan halutaan muuttuvan, sillä Internet Explorer ei tue tätä ominaisuutta. Tästä syystä kaikki kuvatiedostoille toteutetut häilymisefektit toteutettiin JavaScriptia hyödyntäen.

Kuva 19. Henkilöhakutulossivu ennen ja jälkeen JavaScriptin.

JavaScriptia hyödynnettiin myös käyttäjän opastamiseen. Mikäli käyttäjä suoritti intranet-portaalissa haun, joka ei sisältänyt henkilöitä, henkilöhakutulokset-näkymä näytti kuvassa 19 punaisella rajattua tyhjää elementtiä. JavaScriptin avulla tyhjä hakutuloksen elementti saatiin piilotettua niissä tapauksissa, kun hakuja ei löytynyt. Tyhjää ha-

kutuloksia näyttävävää elementtiä ei olisi voitu piilottaa tyylimäärityksiä hyväksikäyttäen, sillä silloin hakutulokset olisivat aina olleet piilotettuna. Nappiefektit ja käyttöliittymän muokkaukset JavaScriptia hyödyntämällä saattavat vaikuttaa pieneltä hienosäädöltä, mutta tosiasiassa ne tekevät käyttöliittymästä huomattavasti eloisamman ja parantavat käytettävyyttä. Esimerkiksi tyhjä hakutuloslaatikko voisi aiheuttaa käyttäjälle varmasti hetken hämmennystä.

Joitakin käyttöliittymän elementtejä ei olisi voitu kustomoida ilman JavaScriptia. Esimerkiksi hakutulossivun graafikon antaman ohjeistuksen mukainen kustomointi tuotti ylitsepääsemättömiä haasteita pelkkiä tyylimäärityksiä hyväksikäyttäen. Tämä johtui siitä, että Sharepoint loi hakutulossivulle html-taulukon, jonka tiettyjä attribuutteja pystyi muuttamaan jälkikäteen vain JavaScriptilla. Kuvassa 19 nähdään sinisellä rajattuna hakutulossivun navigaatiopaneelit pelkillä tyylimäärityksillä ja vihreällä rajattuna JavaScriptia hyödyntäen.

```
11  //hides people search results or main search results if results are empty
12  function hidesearchBox() {
13
14      var test = $(".srch-federationarea").height();
15      var box1 = $(".srch-maintop2").height();
16
17      if (test < 20) {
18          $(".srch-federationarea").css("display", "none");
19      }
20      else {
21          $(".srch-federationarea").css("display", "block");
22      }
23
24      if (box1 < 20) {
25          $(".srch-maintop2").css("display", "none");
26      }
27      else {
28          $(".srch-maintop2").css("display", "block");
29      }
30
31  }
32  //modify the position of search tabs to be attached to the search results container
33  function modifySearchTabs() {
34      $("TABLE.ms-ptabarea").attr("cellSpacing", "5");
35  }
```

Koodiesimerkki 4. Hakutulossivun muokkaus.

Koodiesimerkissä 4 nähdään hakutulossivun paneelinavigaation muokkaukseen käytettyä JavaScriptia. Koodiesimerkissä näkyy, kuinka helposti jQuerylla voidaan muokata html-elementin tyylimäärityksiä. Rivillä 34 näkyy, kuinka html-taulukon sarakkeiden väljyyttä muokataan. Riviltä 12 alkaa funktio, jota käytettiin tyhjän hakutuloslaatikon piilottamiseen. Funktiossa yksinkertaisesti katsotaan hakutuloslaatikon korkeus ja piilotetaan hakutuloslaatikko, mikäli korkeus ei ole riittävän suuri, eli toisin sanoen hakutuloksia ei löytynyt.

Intranet-portaalin otsakkeessa toimiva kuvakaruselli toteutettiin puhtaasti JavaScriptia hyödyntäen. Kuvakaruselli hyödyntää myös Ajax-tekniikkaa. Ajax-tekniikka auttaa huomattavasti web-sovellusten käytettävyyden ja interaktiivisuuden parantamisessa. Kuvakaruselli hyödyntää Ajax-tekniikkaa hakemalla listasta kuvakarusellissa käytettävät konfiguraatioasetukset ja kuvatiedostot. Konfiguraatioasetukset määrittävät kuvien vaihtumisvälin sekä kuvien siirtymisnopeuden. Konfiguraatioasetukset mahdollistavat Intranet-portaalin ylläpitäjien lisätä kuvia ja muuttaa kuvakarusellin ominaisuuksia itsenäisesti.

```
9   $(document).ready(function () {
10      var configurationFound = new Boolean();
11      configurationFound = false;
12      var transitionTime = 0;
13      var timeOut = 0;
14      //get configuration values
15      $.getJSON("/_vti_bin/ListData.svc/MTVMEDIAKonfiguraatioarvot", function (data) {
16          $.each(data.d.results, function (i, result) {
17              var title = result.Title;
18              var otsikko = result.Otsikko;
19              //check banner image swap interval
20              if (title == "Bannerikuvan vaihtoväli" || otsikko == "Bannerikuvan vaihtoväli") {
21                  timeout = result.ConfiguarionTexts;
22                  configurationFound = true;
23
24              }
25              //check banner image transition time
26              if (title == "Bannerikuvan transition kesto" || otsikko == "Bannerikuvan transition kesto") {
27                  transitionTime = result.ConfiguarionTexts;
28              }
29
30          });
31          //if configuration found, cache carousel images and start carousel based on configuration values
32          if (configurationFound) {
33              cacheCarouselImages();
34              tid = setInterval(function () { switchImage(parseInt(transitionTime)) }, parseInt(timeout));
35          }
36      });
```

Koodiesimerkki 5. Kuvakarusellin JavaScript.

Koodiesimerkissä 5 nähdään osa kuvakarusellin toiminnallisuuden toteuttavasta JavaScriptista. Koodiesimerkin rivillä 15 haetaan Ajax-tekniikkaa hyväksikäyttäen konfiguraatioarvot listasta. Konfiguraatioarvot sisältävät otsakekuvan vaihtovälin ja siirtymisen keston. Kuvakarusellin toiminnallisuuteen kuului myös kuvien tallentaminen välimuistiin. Välimuistiin tallentaminen tapahtui käytännössä siten, että kuvatiedostot lisättiin kuvaelementteinä piilotettuun html-elementtiin. Internet-selaimet tallentavat automaattisesti html-dokumentin kuva-elementit välimuistiin. Internet-selaimen välimuistin hyödyntäminen parantaa oleellisesti web-sovelluksen suorituskykyä kuvien lataamissa. Koodiesimerkissä kutsutaan kuvat välimuistiin tallentavaa funktiota rivillä 33. Lopuksi koodiesimerkissä rivillä 34 hyödynnetään JavaScriptin setInterval-funktiota, joka mahdollistaa funktion toistamisen säännöllisin väliajoin. Tässä tapauksessa funktiossa kutsutaan konfiguraatiolistan määrittelemän ajan välein kuvakarusellin kuvaa vaihtavaa funktiota.

5.5 Typografia

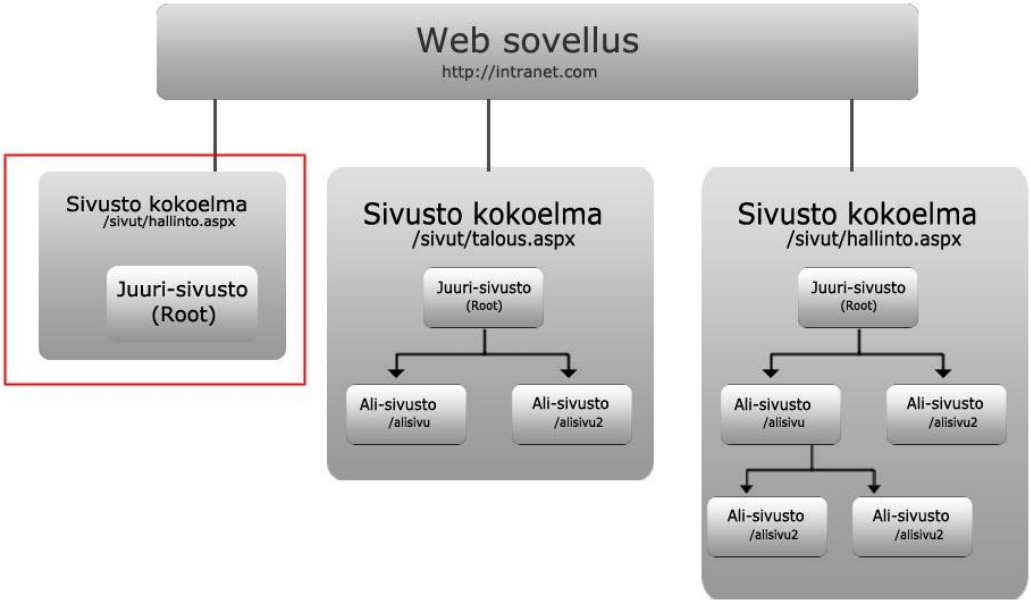
Typografialla tarkoitetaan pelkistettynä tekstin ulkoasuun liittyviä tekijöitä [19, s. 98] Typografiaa pidetään yleisesti jopa taiteena, ja se on verrattavissa maalaamiseen, kuvanveistoon ja tanssiin [20, s. 8]. Typografialla on olennainen osa web-sovellusten käyttöliittymän ulkoasussa. Mikäli typografia toimii hyvin, ei siihen välttämättä kiinnitä erityisempää huomiota. Typografian toimiessa huonosti käyttäjä kokee tekstin lukemisen työlääksi, eikä hahmota tekstisisällön suhteita toisiinsa. Aivan kuten sanomalehdissä, myös web-sovelluksissa tulee selkeästi korostaa tiettyjä tekstielementtejä, kuten otsikkoja. Myös linkit tulee typografian osalta erottua selkeästi muusta tekstikokonaisuudesta.

Intranet-portaalin typografia määriteltiin graafikon toimesta. Typografia sisälsi käytettävien tekstielementtien fonttikoot ja fonttien värit. Typografia toteutettiin tyylimäärityksiä hyväksikäyttäen. Ongelmaksi kuitenkin muodostui Sharepointin tottelemattomuus fonttimääritysten suhteen. Typografian määrittäminen tyylitiedostossa kerran ei riittänyt, vaan tietyissä tilanteessa fonttimääritykset tulivat täysin Sharepointin omista tyylitiedostoista. Tällaisiä tapauksia olivat esimerkiksi niinsanotut rikkaat html-kentät ja web-osat. Rikas html-kenttä on ASP.NET:ssä valmiiksi esiintyvä komponentti, joka antaa käyttäjälle mahdollisuuden syöttää sisältöä tekstin, kuvien ja taulukoiden muodossa. Ongelma kuitenkin ratkesi luomalla typografia määritykset erikseen sisältöeditoreille ja web-osille.

5.6 Navigaation yhtenäistäminen

Yhtenäinen navigaatio internet-sivujen välillä on tärkeä ominaisuus web-sovelluksessa. Se parantaa käytettävyyttä ja tuo selkeyttä käyttöliittymään. Intranet-portaalin päänavigaationa toimi otsakkeen alapuolella oleva ylänavigaatio. Ylänavigaation tarkoitus on näyttää linkit päätason sivuille ja alisivuille. Ylänavigaation ulkonäön muokkaaminen graafista ohjeistusta vastaavaksi tapahtui tyylitiedostoja hyväksikäyttäen. Ulkonäön muokkaaminen sujui melko nopeasti muiden käyttöliittymäelementtien tyylien muokkauksista omaksutuilla käytännöillä. Ongelmaksi kuitenkin muodostui itse navigaatiolinkkien säilyminen siirryttäessä sivustokokoelmasta toiseen. Siirryttäessä esimerkiksi oma sivu -sivustokokoelmaan, navigaatiolinkit tulivat oma sivu -sivustokokoelmasta pääsivu-

kokoelman sijasta. Tämä aiheutti epäyhtenäisen navigaation siirryttäessä sivustokokoelmista toiseen.



Kuva 20. Sharepoint-hierarkia.

Navigaatio-ongelman ratkaisemiseksi ei ollut valmista ratkaisua. Kuvassa 20 on havainnollistettu sivustokokoelmien hierarkiaa kuvitteellisessa sovelluksessa. Jotta navigaatio toimisi oikein, täytyisi navigaatio-linkkien tulla kuvassa punaisella rajatusta sivustokokoelmasta. Oletusarvoisesti ylänavigaation navigaatiolinkit tulevat samasta sivustokokoelmasta, johon käyttäjä on navigoinut. Navigaatio-ongelman ratkaisemiseksi täytyi navigaatiota varten tehdä käyttäjäkontrolli ja http-käsittelijä. Navigaation käyttäjäkontrollin ja http-käsittelijän toteuttaminen oli melko työlästä toteuttaa.

```
public void ProcessRequest(HttpContext context)
{
    //write your handler implementation here.
    string url = context.Request.Url.ToString();
    //Depth = int.Parse(context.Request.QueryString["Depth"]);
    SPSite site = SPContext.Current.Site;
    SPSecurity.RunWithElevatedPrivileges(GetGlobalNav);

}
//Get global navigation nodes
private void GetGlobalNav()
{
    try
    {
        SPSecurity.RunWithElevatedPrivileges(delegate()
        {
            //Note: PortalSiteMapProvider returns data for the current request's site
            SPSite rootsite = SPContext.Current.Site;
            PortalSiteMapProvider provider = new PortalSiteMapProvider();
            provider.CurrentSite = rootsite;
            var rootNode = provider.RootNode;
            if (rootNode != null)
            {
                var nodes = PortalSiteMapProvider.GlobalNavSiteMapProvider.GetChildNo
                //get the serialized navigation nodes
                SerializeWrite(GetSerializableSiteMap(nodes));
            }
        });
    }
```

```
using (SPSite elevatedSite = new SPSite(siteGuid))
{
    try
    {
        var app = elevatedSite.WebApplication;
        StringReader strRead = null;
        XmlSerializer xmlSerializer = null;
        XmlReader xmlReader = null;
        //get the main site ciollections rootweb
        var mainWeb = GetRootWeb(elevatedSite);
        using (SPSite rootSite = app.Sites[0])
        {

            //get the current user
            var currentUser = mainWeb.CurrentUser;
            //uri to httpHandler
            var uri = rootSite.RootWeb.Url + "/_layouts/Digia.MTVMedi
            //perform webrequest to the httpHandler
            var request = WebRequest.Create(uri);
            request.Method = "GET";
            request.UseDefaultCredentials = true;
            request.PreAuthenticate = true;
            request.Credentials = CredentialCache.DefaultCredentials;
            //Get response from httpHandler
                    var response = request.GetResponse();
                    if (response != null)
                    {
                        var stream = response.GetResponseStream();
                        var reader = new StreamReader(stream);
```

Koodiesimerkki 6. Http-käsittelijä ja käyttäjäkontrolli.

Koodiesimerkissä 6 nähdään vasemmalla puolella osa http-käsittelijää ja oikealla puolella osa käyttäjäkontrollia, joita käytettiin navigaation yhtenäistämiseen oma sivusto sivustokokoelmissa. Koodiesimerkin vasemmalla puolella rivillä 36 oleva ProcessRequest metodi vastaa käyttäjäkontrollin http-pyyntöön ja kutsuu navigaation xml-muotoon serialisoivaa GetGlobalNav metodia rivillä 42. Varsinaista navigaation serialisoivaa metodia kutsutaan rivillä 61. Lopputuloksena http-käsittelijä tulostaa pääsivustokokoelman navigaation xml muodossa. Koodiesimerkin oikealla puolella 53 nähdään, kuinka käyttäjäkontrolli muodostaa http-pyynnön http-käsittelijälle WebRequest.Create(Uri) -metodilla. parametri Uri on url-osoite http-käsittelijän ashx-tiedostoon. Käyttäjäkontrollin riveillä 54-7 nähdään, kuinka request-objektin omaisuudet asetetaan. Http-pyynnön vastauksen käsittely aloitetaan käyttäjäkontrollin koodiesimerkin rivillä 59. Käyttäjäkontrollin koodi jatkuu siten, että http-käsittelijän xml-data deserialisoidaan, jotta xml-datasta voidaan muodostaa navigaatio-objekteja.

Oli yllättävää todeta, että navigaation yhtenäistäminen sivustokokoelmien välillä vaati melko kustomoitua ratkaisua. Sharepointin oma ratkaisu olisi tarjonnut vain linkin pääsivustokokoelmaan ponnahdusvalikon kautta. Navigaation yhtenäistäminen herätti kysymyksiä siitä, miksi näin yksinkertainen asia on täytynyt tehdä niin vaikeaksi. Navigaation yhtenäistäminen vaati satoja rivejä koodia. Yhtenäinen navigaatio on kuitenkin hyvin olennainen elementti käyttöliittymässä, joten saattaisi olettaa, että Microsoft olisi keksinyt valmiin ratkaisun ongelmaan.

6 Testaus ja projektin kulku

6.1 Johdatus käyttöliittymän testaamiseen

Testaus on olennainen osa ohjelmistoprojekteja. Testaus on hyvä aloittaa jo ohjelmistoprojektin alkuhetkistä lähtien ja jatkaa projektin loppuun asti. Käyttöliittymän testaus web-sovelluksissa on melko työläs prosessi. Käyttöliittymän testausta varten on olemassa työkaluja, jotka testaavat esimerkiksi, ovatko oikeat käyttöliittymän elementit näkyvillä tai onko käyttöliittymän elementeissä määritelty sisältö [21]. Automatisoidut testityökalut eivät kuitenkaan kerro, näyttävätkö käyttöliittymän elementit siltä kuin niiden pitää eri internet-selaimissa. Tästä syystä käyttöliittymän testaukseen käytetyt menetelmät olivat manuaalisia.

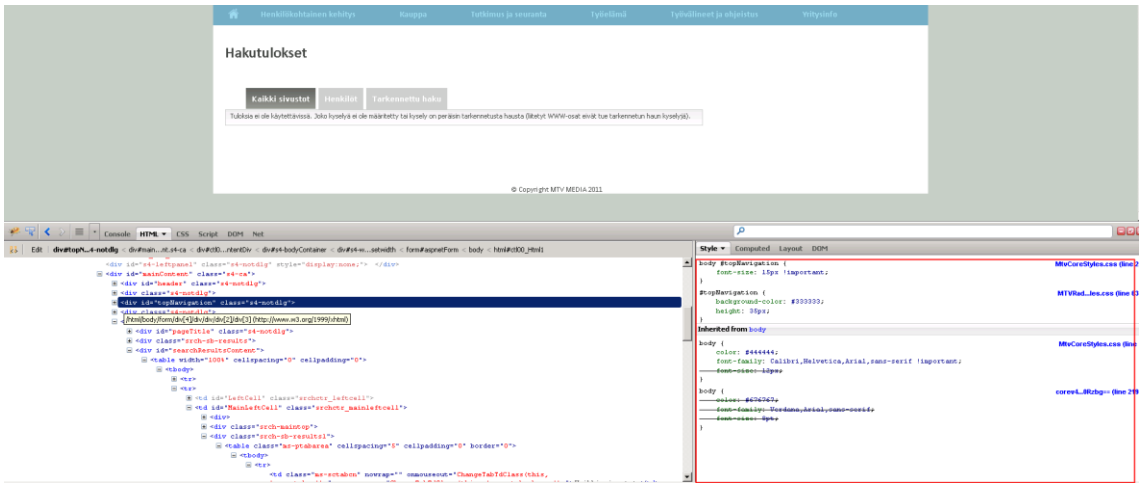
Ennen käyttöliittymän testaamista tulee olla selvää, kuinka käyttöliittymän virheet esiintyvät. Yleisin käyttöliittymän virhe on elementtien siirtyminen pois paikoiltaan eri internet-selaimilla. Käyttöliittymä saattaa näyttää täydelliseltä Mozilla Firefox internetselaimella, mutta Internet Explorer antaa hyvin erilaisen version käyttöliittymästä. Elementit saattavat siirtyä pois paikoiltaan myös internet-sivuston sisällön lisääntyessä. Esimerkiksi tekstielementit saattavat työntää käyttöliittymän elementtejä pois paikoiltaan, mikäli tyylimääritykset eivät ole kunnossa.

Käyttöliittymän testaus oli hyvin itseään toistava prosessi. Uusien käyttöliittymäelementtien tyylimääritysten jälkeen tulos tuli ensiksi tarkistaa kaikilla tuetuilla internetselaimilla. Sovellus yksinkertaisesti avattiin eri internet-selaimissa ja todettiin silmämääräisesti, että kaikki on kunnossa. Tämän jälkeen tuli tarkistaa, että käyttöliittymän elementit toimivat yhdessä muiden elementtien kanssa. Toimivuus muiden elementtien kanssa todettiin testaamalla tyylitellyn käyttöliittymäelementin ympärillä olevien elementtien, kuten web-osien -toimintaa. Joissakin tapauksissa kaikki näytti aluksi toimivan uusien käyttöliittymäelementtien tyylimääritysten osalta, kunnes sisältöä lisättiin. Esimerkiksi yksi tavallista pidempi sana saattoi rikkoa joidenkin web-osien käyttöliittymän siirtämällä elementtejä pois paikoiltaan. Käyttöliittymän laaja testaaminen ei suinkaan tarkoita, ettei virheitä tulisi esiintymään tulevaisuudessa. Mitä enemmän ja mitä erilaisemmilla tavoilla sovellusta käytetään, sen todennäköisemmin uusia virheitä löytyy. Uusien virheiden löytymisen yhteydessä asiakas todennäköisesti raportoi virheestä, ja sovelluksen ylläpidosta vastaavat tahot tekevät parhaansa virheen korjaamiseksi.

6.2 Internet-selainten erot haasteena

Suurimmaksi haasteeksi käyttöliittymän rakentamisessa tyylitiedostojen osalta osoittautui internet-selaimien väliset erot. Alunperin intranet-portaalin määrittelyissä vaadittiin, että sovelluksen tulee tukea viittä eri internet-selainta. Valitettavasti mitään Intranetportaalin sivua ei saatu yhtenäistettyä viidelle internet-selaimelle ulkoasun osalta ensimmäisen kahden viikon aikana. Tämän jälkeen projektin johto totesi, että Sharepoint -alustalla on turha lähteä tukemaan kaikkia suosittuja internet-selaimia, mikäli käyttöliittymää räätälöidään paljon. Lopulta tuettujen internet-selaimien määrä pudotettiin kolmeen.

Intranet-portaalin ulkoasun yhtenäistäminen pelkästään kolmelle internet-selaimelle ei ollut helppo tehtävä. Oli hyvin turhauttavaa todeta, että jokin käyttöliittymän elementti toimii täydellisesti Mozilla Firefoxilla ja Internet Explorer 8:lla, mutta ei Internet Explorer 7:llä. Vikojen selvittäminen oli pitkälti jatkuvaa tyyliasetusten muuttamista ja testaamista. Mozilla Firebug osoittautui erinomaiseksi työkaluksi esimerkiksi uusien tyylimäärittelyjen testaukseen. Toisin kuin Internet Explorerin web-kehittäjille suunnattu developer-tools, Mozilla Firebug mahdollisti kätevästi uusien tyylimääritysten luomisen interaktiivisesti ja mahdollisti jopa JavaScriptin virheiden etsinnän.



Kuva 21. Intranet-portaalin sivun elementti Firebugin tarkastelussa.

Kuvassa 21 näkyy Intranet-portaalin käyttöliittymän elementti Firebugin tarkastelun alla. Tarkasteltava elementti on kuvassa vaaleansinisenä näkyvä ylänavigaatio. Firebug listaa tyylimäärittelyt kuvan punaisella rajatulla alueella. Tyylimäärityksiä voi kytkeä päälle ja pois kuvassa näkyvistä tyylimääritysten viereen ilmestyvistä valintaruuduista. Myös html-dokumentin muokkaaminen onnistuu Firebugilla helposti. Yksi tyylimääritysten laatimista huomattavasti nopeuttava tekijä on mahdollisuus kopioida Firebugilla tehdyt tyylimääritykset ja CSS-valitsijat. Firebugin käyttömahdollisuudet eivät suinkaan rajoitu vain html-elementtien muokkaukseen ja tyylimääritysten määrittelyyn, vaan sillä voi myös esimerkiksi ajaa JavaScriptin virheiden etsintää, suorittaa JavaScript-skriptejä ja tarkkailla http-liikennettä. Http-liikenteen tarkkailu on erityisen hyödyllinen lisä, sillä se paljastaa sovelluksen pullonkaulat http-pyynnöillä haettujen tiedostojen latausaikojen perusteella.

6.3 Oikeellisuuden tarkistaminen ja todentaminen

Ohjelmistoprojektien loppupuolella suoritetaan yleensä todentamisvaihe ja oikeellisuuden tarkistusvaihe. Todentamisella tarkoitetaan sitä, että ohjelmisto vastaa sille laadittua spesifikaatiota [22, s. 1]. Oikeellisuuden tarkistamisella tarkoitetaan sitä, että asiakas toteaa tuotteen vastaavan sille asettamiaan odotuksia [22, s. 1]. Käyttöliittymän toimimattomuus saattaisi hyvin johtaa esimerkiksi todentamisen estymiseen. Toisin sanoen mikäli esimerkiksi käyttöliittymä ei ole yhdennäköinen kaikilla selaimilla, fonttimääritykset eivät täsmää tai sivunasettelu rikkoontuu, käyttöliittymä voidaan todeta virheelliseksi.

Intranet-portaalin käyttöliittymän todentaminen muodostui käytännössä kolmesta tekijästä. Ensinnäkin käyttöliittymän tuli vastata laadittua spesifikaatiota. Mikäli spesifikaatiossa on esimerkiksi mainittu, että tietyllä sivulla tulee olla laaditut linkit, pitää linkkien olemassaolo pystyä toteamaan käyttöliittymää katsomalla. Toiseksi tulee tarkistaa käyttöliittymän ulkonäkö. Käyttöliittymästä laadittiin projektin alussa graafiset versiot. Asiakas olettaa, että käyttöliittymä näyttää samanlaiselta kuin alkuperäisissä graafisissa versioissa. Haastavaksi tämän todentamisen vaiheen teki spesifikaatiossa määriteltyjen internet-selaimien erilaisuudet. Kolmanneksi käyttöliittymän tulee toimia odotetulla tavalla. Mikäli esimerkiksi jotkin käyttöliittymän elementit siirtyvät pois paikoiltaan esimerkiksi sisällön syöttämisen yhteydessä, käyttöliittymässä on virhe.

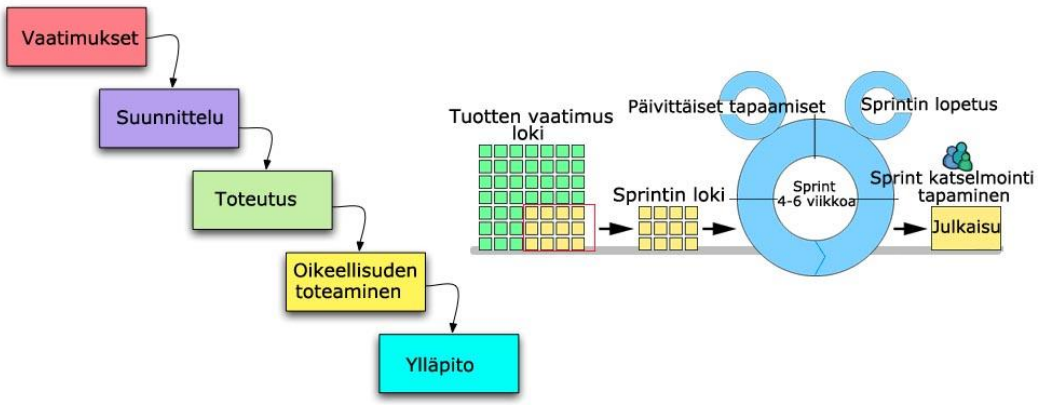
Todentamisen ja oikeellisuuden tarkistuksen yhteinen tavoite on taata, että sovellus vastaa käyttäjän tarpeita [22, s. 1] Oikeellisuuden tarkistus on suoritettu onnistuneesti, kun asiakas toteaa, että tuote vastaa sille paperilla asetettuja vaatimuksia. Vaikka todentamisen yhteydessä todettaisiin tuotteen vastaavan spesifikaatiota, se ei suinkaan tarkoita, että tuote olisi virheetön. Virheetön koodi ohjelmistotuotannossa on fiktiota. todentamiseen riittää, että tuote on käyttötarkoitukseensa riittävän hyvä. Jos tuote vastaa spesifikaatiota, se ei kuitenkaan välttämättä täytä asiakkaan odotuksia. Asiakkaan nähdessä valmiin tuotteen saattaa ilmaantua asiakkaan puolesta tarve vielä muuttaa tuotetta. Tällöin puhutaan jatkokehityksestä.

Intranet-portaali -projektissa todentamista harjoitettiin säännöllisesti alusta loppuun. Toisinaan asiakas huomautti jostakin toiminnallisuudesta, joka oli määritelty spesifikaatiossa mutta puuttui tuotteesta. Suurin osa virheistä löytyi kuitenkin testauksen yhtey-

dessä. Käyttöliittymässä ilmenneitä virheitä oli kokonaisuudessaan lähemmäs sata kappaletta. Virheitä korjattiin sitä mukaa, kuin niitä ilmaantui. Käyttöliittymävirheet olivat pääasiassa internet-selainten eroavaisuuksista johtuvia.

6.4 Scrum projektinhallintamuotona

Intranet-portaalissa käytettiin niin sanottua Scrum-projektinhallintamenetelmää. Scrum on pääasiassa ohjelmistokehitysmenetelmä, mutta sitä voidaan käyttää myös muihin kuin ohjelmistoprojekteihin [23]. Scrum eroaa huomattavasti perinteisestä vesiputousmallista. Vesiputousmalli on tuttu käsite esimerkiksi rakennusteollisuudesta. Vesiputousmalli pohjautuu siihen, että asiakkaan tarpeet eivät muutu kesken projektin ja mikäli tuote vastaa alunperin laadittua spesifikaatiota, asiakas on tyytyväinen. Ohjelmistokehitysprojektit tarvitsevat vesiputousmallia joustavampaa muotoa, jossa asiakas osallistuu tuotteen kehitykseen. Asiakkaan osallistuminen tuotteen kehitysvaiheeseen tulisi johtaa asiakkaan tarpeet tyydyttävään ohjelmistoon.



Kuva 22. Projektien hallintamenetelmät [23]

Kuvassa 22 on havainnollistettu vesiputousmallia ja Scrum-mallia. Kuvan vasemmalla puolella näkyvä vesiputousmalli havainnollistaa, kuinka suoraviivaisesta projektimallista on kyse. Scrum-projektimalli on havainnollistettu kuvan oikealla puolella. Scrumissa projekti on jaettu osiin eli sprintteihin. Sprintti koostuu vaatimuslokista muodostetuista tehtävistä. Tehtäviä otetaan yksittäiseen sprinttiin sellainen määrä, josta kehittäjät uskovat selviytyvänsä sprintin aikana. Tehtävät tulee myös priorisoida tärkeysjärjestykseen. Jokaisen sprintin jälkeen katsotaan asiakkaan kanssa, että kehityksessä on liikuttu oikeaan suuntaan. Yksittäisen sprintin pituus on tavanomaisesti kahdesta neljään

viikkoa. Scrumia käytettäessä projektitiimi pitää päivittäin lyhyen palaverin siitä, mitä on saatu aikaan edellisenä päivänä ja mitä tullaan tekemään seuraavaksi. Jokaisen sprintin jälkeen suoritetaan sprintin katselmointi. Katselmointiin tulisi osallistua kaikki projektitiimiin kuuluvat asiakkaan edustajat mukaan lukien. Intranet-portaali projektissa sprinteille olisi voinut nostaa tavoitteet selkeämmin esille. Pelkkien työtehtävien priorisointi johtaa kehittäjien osalta helposti siihen, että kaikki keskittyvät vain työstämään omaa osaansa, mutta kokonaiskuvan ajatteleminen jää vähemmälle. Työtehtävää suoritettaessa yhteisen tavoitteen huomioiminen sprintin osalta saattaisi herättää kehittäjien keskuudessa herkemmin kysymyksiä työtehtävien ongelmakohdista ja pullonkauloista.

Scrum-mallin mukaisesti projektitiimi on niin sanotusti itseorganisoituva [24, s. 8]. Tiimille annetaan vapaus toteuttaa työtehtävät parhaalla katsomallaan tavalla. Tiimin jäseniä kannustetaan kommunikoimaan keskenään hyvien ratkaisujen löytämiseksi. Vapaa työskentelytapa ei kuitenkaan tarkoita kevyempää työtahtia. Jokaisesta kehittäjille annetuista työtehtävistä oli muodostettu työmääräarvio, johon tuli pyrkiä. Työmääräarvioista muodostui melko summittaisia, sillä ohjelmistoprojekteissa on melko tavanomaista, että kaikki ei mene aina suunnitelmien mukaisesti. Intranet-portaali -projektissa scrum-tiimi sisälsi viisi kehittäjää, arkkitehdin, testaajan ja projektipäällikön. Kommunikointi oli hyvin avointa ja vaivatonta, sillä tiimin jäsenet istuivat testaajaa lukuun ottamatta lähellä toisiaan. Ensivaikutelma scrumista oli tuottava ja tehokas. Tiimin jäsenet kommunikoivat päivittäin ja ongelmia saatiin ratkottua aivoriihen tavoin.

Oppikirjojen mukaan scrum-malliin ei kuulu projektipäällikköä, vaan niinsanottu ”scrum-master” [24, s.9]. Intranet-portaali -projektissa scrum-masterin roolista ei ollut sovittu, joten projektia hallitsi tavanomainen projektipäällikkö. Sprintit olivat neljän viikon mittaisia. Scrum malli tuntui toimivan hyvin, sillä päivittäisten palaverien ansiosta kaikki olivat perillä siitä, mitä projektissa tapahtuu. Palavereissa kehittäjät kertoivat mitä, olivat saaneet aikaan ja mitä tulevat tekemään päivän aikana. Myös havaituista ongelmista ja niiden ratkaisuista kerrottiin. Testaaja vastaavasti kertoi löytämistään virheistä sovelluksessa. Omalta osaltani raportoin pääasiassa käyttöliittymän virheiden korjauksista Scrum-palavereissa. Arkkitehti näytti scrum-palavereissa aika ajoin miltä kokonaiskuva näyttää esimerkiksi sovellusten virheiden osalta. Jokaisen sprintin aikana saimme myös kuulla asiakkaan palautteen projektin edistymisestä.

Scrum-mallin mukaisesti asiakkaan kanssa tehdään tiivistä yhteistyötä ohjelmistoprojektin alusta loppuu [24, s. 10]. Jokaisen sprintin jälkeen oli määrä katsoa asiakkaan kanssa, onko sovellus kehittymässä haluttuun suuntaan. Asiakkaan palaute oli projektin alussa melko negatiivista, tosin palaute kohdistui pääasiassa käyttöliittymään. Tähän vaikutti kenties runsas virheiden määrä käyttöliittymässä projektin alkuvaiheessa. Loppua kohden asiakaspalaute oli huomattavasti positiivisempaa kaikilta osin. Positiivinen palaute oli hieman harhaanjohtavaa, sillä projektin ensimmäisessä hyväksymisvaiheessa asiakas kuitenkin raportoi lukuisista puutteista. Positiivinen palaute antoi myös kehittäjille sellaisen kuvan, että projekti on loppusuoralla ja tuote vastaa spesifikaatiota. Asiakas olisi voinut osallistua aktiivisemmin ja kriittisemmin sprinttien katselmointiin ja kiinnittää enemmän huomiota toiminnallisuuteen, vaikka käyttöliittymä onkin tärkeä osa-alue. Tällöin sovelluksen pullonkauloihin olisi voitu kohdistaa enemmän huomiota aikaisemmassa vaiheessa.

6.5 Työmääräarviot

Työmääräarviot ovat hyvin tärkeitä projektin budjetin kannalta. Työtehtäviin käytetyt budjetit määritellään työmääräarvioiden pohjalta. Mikäli työtehtävien suorittamiseen menee pidempi aika kuin arvioitu, projektin budjetti saattaa ylittyä. Työmääräarviot laativat pääasiassa sovelluskehittäjät. Itse en kuitenkaan määritellyt käyttöliittymän rakentamisen kokonaisvaltaista työmääräarviota, sillä en ollut projektin jäsen vielä projektin aloitusvaiheessa. Työmääräarvio käyttöliittymän rakentamiseen oli arvioitu huomattavasti alhaisemmaksi, kuin todellisuus vaati. Pelkästään käyttöliittymän virheiden korjaamiseen kului aikaa moninkertaisesti koko käyttöliittymään rakentamiseen budjetoidun ajan verran. Mikäli käyttöliittymä olisi toteutettu tukemaan useampaa kuin kolmea internet-selainta, kuten alkuperäinen spesifikaatio vaati, työmääräarvio olisi ollut pielessä pahemman kerran.

Ydinongelma työmääräarvioiden suhteen oli projektitiimin kokemattomuus Sharepoint 2010 -alustasta, sillä tuote on melko uusi. Etenkin käyttöliittymän kehittämisestä Sharepoint 2010:lle kenelläkään projektin jäsenellä ei ollut aikaisempaa kokemusta. Lisäksi projektin aikana kävi vahvasti ilmi, että raskas räätälöinti käyttöliittymän ja toiminnallisuuden osalta Sharepoint-projekteissa johtaa hyvin helposti työmääräarvioiden ylittymiseen ja sitä kautta kannattavuuden kärsimiseen. Monet käyttöliittymän toteutus-

tehtävistä olivat näennäisesti hyvin yksinkertaisia. Esimerkiksi intranet-portaalin jokaiselle käyttäjälle luotava ”oma sivu” tuli olla hyvin pitkälti alustan standardi-sivu perustyylisivua lukuun ottamatta. Ongelmaksi muodostui odottamattomat tekijät, kuten navigaation yhtenäistämisen vaikeudet eri sivustokokoelmien välillä. Jotta Sharepoint-projekteista saataisiin helpommin hallittavia ja taloudellisesti kannattavampia, ohjelmistotalon tulisi kehittää tuotteistettu ratkaisu, jonka toiminnallisuudet ja käyttöliittymä ovat valmiiksi hyvin testattuja. Tuotteistettu ratkaisu nostaisi myös asiakastyytyväisyyttä, sillä aikamääreet ja spesifikaatiota vastaava toiminnallisuus saataisiin tavoitettua helpommin.

6.6 Projektin asiakaspalaute

Intranet-portaali projektin alussa suurin osa asiakkaan huomiosta kohdistui käyttöliittymään. Käyttöliittymässä oli alkuvaiheessa puutteita monilta osin ja tämä johti suureen kritiikin määrään. Omalta osaltani suuri huomio käyttöliittymää kohtaan asetti runsaasti paineita, jotta päästäisiin spesifikaation mukaisiin määritelmiin. Lähtökohta oli selkeästi se, että käyttöliittymän tulee olla täysin virheetön. Jopa yhden pikselin ylimääräinen marginaali elementin sijoittelussa tuli korjata.

Käyttöliittymän tarkkuuteen kohdistunut kritiikki saattaa ensiksi kuulostaa liioitellulta. Intranet-portaali ratkaisut Sharepoint alustalla eivät kuitenkaan ole halvimmasta päästä ohjelmistoprojekteja. Yrityksen investoidessa suuren summan ohjelmistoon, tulee ohjelmiston luonnollisesti olla juuri sitä mitä on tilattu, eikä melkein. Käyttöliittymä ei ole tässä suhteessa poikkeus. Ohjelmistoprojektin hinnan kasvaessa myös ohjelmistoon kohdistuvat odotukset ovat korkealla.

Toisinaan sovelluksen muutostoiveissa oli ristiriitoja. Tämä näkyi käytännössä siten, että samasta ohjelmaan kohdistuvasta toiveesta esitettiin kaksi versiota, jotka saattoivat olla jopa toistensa vastakohtia. Kommunikaatiotaidot ovat merkittävässä roolissa muutoshallinnasta puhuttaessa, jotta kaikki osapuolet ymmärtävät toisiaan. Kommunikaatio asiakkaan ja kehittäjien välillä toimii parhaiten silloin, kun yksi kehittäjätiimin jäsen on suorassa yhteydessä asiakkaaseen. Mikäli kommunikaatio asiakkaan ja kehittäjien välillä toimii ainoastaan projektipäällikön kautta, kommunikointi on työlästä ja väärinkäsitysten todennäköisyys on suuri.

6.7 Projektin käytäntöjen hyödyntäminen tulevaisuudessa

Ohjelmistoprojekteissa on hyvä pyrkiä luomaan komponentteja, jotka ovat helposti uudelleenkäytettävissä vastaavissa tulevaisuuden ohjelmistoprojekteissa. Käyttöliittymä on olennainen osa web-sovelluksia. Perustyylisivut, sivunasettelut ja tyylitiedostot voi helposti toteuttaa siten, että ne ovat uudelleenkäytettävissä myös jatkossa. Intranet-portaali–projektin loppupuolella varmistin käyttöliittymäkomponenttien uudelleenkäytettävyyden. Perustyylisivut ja sivunasettelut toimivat sellaisenaan uusissa projekteissa ongelmitta. Sen sijaan tyylitiedostot vaativat hieman muokkaamista, jotta niidenkin uudelleenkäyttäminen sujuisi helposti.

Ongelma tyylitiedostojen muokkaamisessa on niiden laajuus. Mikäli kaikki Intranet-portaalin tyylimääritykset olisi yhdistetty yhteen tiedostoon, rivimäärä olisi lähennellyt tuhatta. Tyylimääritykset hajautettiin tästä syystä moneksi eri tiedostoksi käyttötarkoitustensa mukaan. Tyylitiedostojen hajauttaminenkaan ei tehnyt uudelleenkäyttöä helpoksi, sillä jotkin tyylitiedostot olivat silti satojen rivien mittaisia. Päädyin lopulta muokkaamaan tyylitiedostoja siten, että syntyi niin sanottu CSS-kehys käyttöliittymän elementtien muokkausta varten. CSS-kehys mahdollistaa tyylimääritysten muokkaamisen pienillä tyylitiedostomuutoksilla. Esimerkiksi Intranet-portaalin reunanavigaatiota varten määriteltiin CSS-kehys, joka mahdollistaa reunanavigaation, web-osien, ylänavigaation ja otsakkeen muokkaamisen.

```
/*-----leftpanel DIMENSIONS-----*/
#s4-leftpanel{
    /*width:200px !important;*/
    /*margin-right:30px;*/
}
/*root sites height*/
#s4-leftpanel UL.root > LI > .menu-item{
    height:30px;
}
/*subsites height*/
#s4-leftpanel UL.root > LI > UL > LI .menu-item{
    height: 25px;
}
/*subsites width*/
#s4-leftpanel UL.root > LI > UL{
    /*width:170px;*/
}

/*adjust width of subsites of selected link*/
#s4-leftpanel ul.root > li.selected > ul.static > li.static{
    /*width: 95%;*/
}
/*text positioning of root sites*/
#s4-leftpanel UL.root > LI > A .menu-item-text{
    /*margin-top:4px;
    margin-left:2px;*/
}
/*text positioning of selected root site*/
#s4-leftpanel UL.root > LI.selected > A .menu-item-text{
    /*margin-top:1px;*/
}
/*text positioning of subsites*/
#s4-leftpanel UL.root > LI > UL > LI .menu-item-text{
    /*margin-top:5px;
    margin-left:0px;*/
}

/*--------------------END DIMENSIONS-------------------------*/
```

```
/*Quicklaunch sp classes*/
.s4-ql a.selected{
    background: none;
}
.s4-ql ul.root > li.selected
{

    color: White;
    margin: 0px 1px 0px 2px;
    *margin: 0 1px 1px 2px;
    background: none;
    position: relative;
    top: -4px;
}
.s4-ql UL.root > li.static > UL.static > li.selected > A
    text-decoration:underline;
}
.s4-ql UL.root > li.static > UL.static > li.selected{
    background-color: #7B8F99;

}
.s4-ql  li.selected ul
{
    background: White;
    padding-left: 15px;
    width: 100%;
    width: 191px;
}
.s4-ql a.selected
{
    border: none;
}

/*blue background lin*/
.s4-ql  li.selected li.static
{
    background-color: #7B8F99;
    margin-top: 1px;
```

Koodiesimerkki 6. Leftpanel CSS-tyylimäärityksiä.

Koodiesimerkissä 6 näkyy vasemmalla puolella osa reunanavigaatiota muokkaavaa tyylitiedostoa CSS-kehyksen muodostamisen jälkeen. Verrokkina koodiesimerkin oikealla puolella on nähtävillä, miltä tyylitiedosto näytti ennen CSS-kehystä. Ennen CSS-kehystä reunanavigaation tyylien muokkaaminen edellytti syventymistä tyylitiedoston kuhunkin tyylimääritykseen, jotta saatiin selville mitä tyylejä tulee muokata halutun tyylimuutoksen aikaansaamiseksi. CSS-kehyksessä tyylit on eritelty siten, että esimerkiksi reunanavigaation ulkoasua pystyy muokkaamaan elementtikohtaisesti. Esimerkiksi kaikkien reunanavigaation alisivustojen taustakuvan korkeuden muokkaaminen onnistuu yhtä parametria muuttamalla. CSS-kehyksessä on lisäksi kommentoitu ohjeistus, mitä tyyliä kukin tyylimääritys muuttaa. Pahimmillaan satojen rivien mittainen tyylitiedosto ilman ohjeistavia kommentteja voi johtaa siihen, että sovelluskehittäjä luo tyylitiedoston kokonaan uudelleen. Saattaa olla huomattavasti nopeampaa selvittää itse, mitä tyylimäärityksiä haluttu tulos edellyttää sen sijaan, että tulkitsee toisen ihmisen tekemiä tyylimäärityksiä. CSS-kehys tuo selkeän ja helppokäyttöisen ratkaisun tyylimääritysten uudelleenkäyttöä varten. Muita uudelleenkäytettäviä komponentteja oli sivustokokoelmien navigaation yhtenäistävä käyttäjäkontrolli ja http-käsittelijä.

7 Yhteenveto

Insinöörityön tarkoituksena oli tutkia käyttöliittymäsuunnittelun haasteita Sharepointille ja muodostaa tutkimustulosten perusteella hyviä käyttöliittymän ongelmien ratkaisumalleja. Ratkaisumalleilla tavoitellaan koodin uudelleenkäytettävyyttä ja pyritään välttämään samojen asioiden ratkaisemista uudelleen. Suurin osa ratkaisumalleista koski tyylimäärityksiä. Tyylimääritysten pohjalta luotu CSS-kehys tarjoaa sellaisenaan ratkaisumallin käyttöliittymän elementtien ulkoasun muokkaamiseen siten, että ulkoasu näyttää samalta sovelluksen tukemissa internet-selaimissa. Tuettujen internet-selainten lista jäi melko lyhyeksi. Jatkokehityksenä voitaisiin kehittää CSS-kehystä siten, että useampi internet-selain on tuettuna.

Lopputuloksena syntyi uudelleenkäytettäviä perustyylisivuja ja sivunasetteluita, CSS-kehys elementtien tyylimäärityksiä varten, JavaScript-komponentteja, käyttäjä-kontrolli navigaation yhtenäistämiseksi, perustyylisivun vaihtavia tapahtuman käsittelijöitä ja ominaisuuksien nitojia. Kaikkia käyttöliittymän luomiseen käytettyjä komponentteja voidaan hyödyntää tulevaisuuden projekteissa. Etenkin CSS-kehys nopeuttaa huomattavasti käyttöliittymän ulkoasun muokkaamista. Insinöörityö itsessään voisi hyvin toimia johdatuksena käyttöliittymäsuunnitteluun ja toteutukseen Sharepoint 2010 -alustalla.

Intranet-portaalin osalta projektitiimi sai paljon uutta kokemusta ja taitoa. Intranet-portaalin ja muiden yrityksen tuottamien Sharepoint-ratkaisujen osalta vaikuttaa vahvasti siltä, että vahvasti räätälöidyt ratkaisut ovat hyvin haastavia toteuttaa. Tästä syystä kysyntä vahvasti tuotteistetuille kiinteähintaisille ratkaisuille on nousussa. Yritysten tulisi tuottaa valmiiksi testattuja komponentteja ja hyödyntää maksimaalisesti valmiiksi olemassa olevaa toiminnallisuutta raskaiden räätälöintiratkaisujen sijasta. Tällä tavoin saataisiin kenties pidettyä Sharepoint-projektit kannattavina ja asiakastyytyväisyys hyvänä.

1 Lähteet

1. Krug, Steve. 2006. Don’t put make think. Berkeley, California: New Riders.
2. Tervakari & Silius. 2005. Arvioinnin kohteena informaatioarkkitehtuuri ja toiminta-prosessit. Luentomoniste. Tampereen Teknillinen Korkeakoulu.
3. Casteleyn, Sven. 2009. Engineering Web Applications. New York: Springer.
4. Mannonen, Mika. 2011. Projektipäällikkö, Digia Oy. Helsinki. Keskustelu 30.9.2011.
5. Holland, Charlie.Microsoft Sharepoint 2010 Web Applications. New York:McGraw-Hill.

6. Follette, D., Stubbs P. 2010. Accessing Sharepoint Data in Office 2010. MSDN Magazine. Vol.25. s. 20-22.

7. Sharepoint 2010 Logical Architecture. 2010. Verkkodokumentti. RDA. <http://collab.rdacorp.com/2010/03/sharepoint-2010-logical-architecture.html>. Luettu 30.9.2010

8. Kirk, Evans. 2009. Using Business Connectivity Services in Sharepoint 2010. Verkkodokumentti. <http://msdn.microsoft.com/en-us/magazine/ee819133.aspx>. Luettu 30.9.2011.

9. MacDonald, Matthew. 2010. Pro ASP.NET 4 in CSharp 4th edition. New York: New York: Springer Science+Media.

10. Rizzo, T., Alirezaei R., Swider P., Hillier S., Fried J., Schaefer K. 2010. Professional Sharepoint 2010 Development. Indiapolis:Wiley Publishing Inc.

11. Modules. 2010. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/library/ms453137.aspx>. Luettu 25.7.2011.

12. Using features in Sharepoint Foundation . 2010. Verkkodokumentti.<http://msdn.microsoft.com/en-us/library/ms460318.aspx>. Luettu 26.7.2011

13. Juvonen, Vesa. 2010. Sharepoint 2010 and web templates. Verkkodokumentti. Microsoft. <http://blogs.msdn.com/b/vesku/archive/2010/10/14/sharepoint-2010-and-web-templates.aspx> Luettu 25.7.2010

14. Plan browser support. 2010. Verkkodokumentti. Microsoft. <http://technet.microsoft.com/en-us/library/cc263526.aspx>. Päivitetty Syyskuu 2011. Luettu 26.7.2011.

15. Drisgill, Randy.2011. Professional Sharepoint 2010 Branding and User Interface. Indianapolis:Wiley Publishing Inc.

16. MacDonald, Matthew. 2010. Beginning ASP.NET 4.0 in C# 2010. New York: Springer Science+Media.

17. Esposite, Dino. 2008. Programming ASP.NET 3.5. Redmond, Washington:Microsoft Press.

18. How to: Customize XSL for the SharePoint Content By Query Web Part. 2011. Verkkodokumentti. Microsoft. <http://msdn.microsoft.com/en-us/library/bb447557.aspx>. Luettu 28.7.2010.

19. Beaird, Jason. 2007. The Principles of Beautiful Web Design. Collingwood:SitePoint Pty. Ltd.

20. Solomon, Martin. 1994. The Art of Typography. New York: Art Direction Book Company.

21. Microsoft. 2010. Testing the User Interface with Automated UI Tests. < http://msdn.microsoft.com/en-us/library/dd286726.aspx>. Verkkodokumentti. Luettu 25.7.2011.

22. Taina, Juha. 2005. Ohjelmistotuotanto, verifiointi ja validointi. Luentomoniste. Helsingin Yliopisto.

23. Scrum is innovative approach to get work done. Verkkodokumentti. Scrum Allience. <http://www.scrumalliance.org/pages/what_is_scrum>. Luettu 6.8.2011.

24. Pichler, Roman. 2010. Agile Project Management with Scrum. Boston: Pearson Education Inc.