

Ilmo Raunio

KOEPALVELUN NÄKYMIIEN
KOOSTAMINEN
DIGITAALISEEN
OPPIMISJÄRJESTELMÄÄN

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma


Marraskuu 2013




MIKKELIN AMMATTIKORKEAKOULU

Mikkeli University of Applied Sciences

KUVAILULEHTI

 MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences		Opinnäytetyön päivämäärä 29.11.2013
Tekijä(t) Ilmo Raunio	Koulutusohjelma ja suuntautuminen Tietojenkäsittelyn koulutusohjelma	
Nimeke Koepalvelun näkymien koostaminen digitaaliseen oppimisjärjestelmään		
Tiivistelmä <p>Suomessa eräitä tunnettuja oppimisalustoja ovat Fronter, Peda.net ja Moodle. Paremmille oppimisalustoille on kuitenkin kysyntää, kuten pedagogisessa diskurssissa on huomattu. Gemilo on pyrkinyt tarjoamaan kouluille omaa oppimisalustaansa. Gemilo on yhteistyöalustapalvelu, jonka keskeisinä teemoina ovat keskustelu ja yhteistyö. Gemilon eräänä puutteena on tähän mennessä ollut automatisoitavat kokeet, ja siihen tässä opinnäytetyössä on paneuduttu näkymien koostamisen perspektiivistä. Työ nähdään esiasteena Digipaperi-palvelulle, jonka avulla kuka tahansa voisi luoda interaktiivisia oppimateriaaleja.</p> <p>Tässä opinnäytetyössä tavoiteltiin koepalveluun kuuluvien käyttötapauksien mallintamista ja koostamista työnantajan toivomien vaatimusten mukaisesti käyttäen Mako Templates -templatointimoottoria sekä jQuery ja jQuery UI -JavaScript-kirjastoja. Kehitysympäristönä oli Python ja Pylons-web-framework. Lisäksi työssä käytettiin mittavasti CSS-merkintäkieltä.</p> <p>Kehitystyön tuloksina tavoiteltiin alun perin toiminnallista prototyyppiä Opetuksen uusia tuulia - tapahtumaa varten 10.10.2013. Tuloksina esiteltiin sen sijaan koepalvelun visuaalista versiota, jonka tarkoituksena on tulevaisuudessa korvata perinteinen paperinippukoe. Opinnäytetyö ei pyrkinyt kytkemään näkymiä jo valmiiksi backend-logiikkaan, vaan esitteli vain koostettuja näkymiä ja sitä, millä tavoin eri näkymät liittyvät loogisesti toisiinsa.</p> <p>Opinnäytetyön aspektina oli modulaarisuus ja kehitystyössä tavoiteltiin komponentteja, johon muidenkin tiimin jäsenten olisi helppo palata. Lähes ohjelmointikieltä muistuttavalla Mako Templates -työkalulla näkymiä on helppo pilkkoa pieniin ja logiisiin osiin. Näin voidaan myös välttää toistoa, tai toisinpäin yhtä komponenttia on mahdollista käyttää mahdollisimman monessa paikassa. Käyttämällä geneerisiä komponentteja vähennetään ylläpidettävien visuaalisten komponenttien määrää.</p>		
Asiasanat (avainsanat) Mako Templates, Python, CSS		
Sivumäärä 47	Kieli Suomi	URN urn:nbn:fi:amk-2013120119257
Huomautus (huomautukset liitteistä)		
Ohjaavan opettajan nimi Janne Turunen	Opinnäytetyön toimeksiantaja Gemilo Oy	

DESCRIPTION

 <p>MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences</p>		Date of the bachelor's thesis 29 November 2013
Author(s) Ilmo Raunio	Degree programme and option Business information technology	
Name of the bachelor's thesis Designing the view templates of an automatized test service for a digital learning environment		
Abstract <p>Some of the more known virtual learning environments in Finland are Fronter, Peda.net and Moodle. However, pedagogic discourse has called for better environments. Gemilo Oy, among others, has strived to offer schools and universities their own virtual learning environment. This far Gemilo has been lacking in automatized tests, and they were the topic of this bachelor's thesis. This work could also be seen as a precursor to the Digipaper service with which any user would be able to create interactive learning materials in the future.</p> <p>The aim of this thesis was to design and compose the use cases as defined by the client by using Mako Templates, a templating engine enabling the use of programmatic view templating alongside with Python. Such libraries as jQuery and jQuery UI were used to power existing frontend JavaScript logic, but they were also used to create slight additions to it. As mentioned, this thesis relied on a Python development environment with Pylons web framework as a major background component. Additionally, the style-scripting language CSS was used extensively to design views of each use case.</p> <p>A functional prototype for the pedagogic event called Opetuksen uusia tuulia, arranged on 10 October 2013, was defined as the aim of this project. Due to lack of time and human resources the project was considered to have failed its initial aim. Instead, visual compositions were created and used in the event. Connecting frontend functionality into their backend counterparts was not the aim of this project, but it showed the visual representation of different use cases and their logical connections to each other as well as introduced the varying states of elements used in multiple use cases.</p> <p>The aspect of this thesis was modularity, and the aim was to create easily modifiable components. With Mako Templates it was easy to separate different sections of view templates programmatically and to create small and logical routines that uphold the DRY principle. With templates one can avoid repetition or, vice versa, use one component in as many places as possible. By using generic components the number of visual components that needed to be supported could be reduced.</p>		
Subject headings, (keywords) Mako Templates, Python, CSS		
Pages 47	Language Finnish	URN urn:nbn:fi:amk-2013120119257
Remarks, notes on appendices		
Tutor Janne Turunen	Bachelor's thesis assigned by Gemilo Oy	

SISÄLTÖ

1	JOHDANTO	2
2	TEKNIIKAT	3
2.1	HTML	3
2.2	DOM	4
2.3	JQuery	5
2.4	Ajax.....	6
2.5	JavaScript.....	7
2.6	ECMAScript	9
2.7	Python	10
2.8	Mako Templates	12
2.9	Cascading Style Sheets	15
2.10	MVC	16
2.11	Rutiinit	16
2.12	Software-as-a-Service.....	18
3	KEHITYSALUSTA JA KEHITTÄMISTEHTÄVÄ	19
3.1	Gemilo ja Gemilo Social	19
3.2	Gemilo Socialin luokkarakenne.....	20
3.3	HTML-näkymien modularisointi Mako-templaateilla	23
4	TOTEUTUS	25
4.1	Templaattihierarkia.....	25
4.2	Näkymät.....	28
4.2.1	Kokeen muokkaus.....	32
4.2.2	Kokeen esikatselu	37
4.2.3	Kokeiden arviointi	38
4.2.4	Tulosten tarkastelu opettajan näkökulmasta	39
4.2.5	Tulosten tarkastelu oppijan näkökulmasta.....	41
4.3	Implementoinnin arviointi	42
5	PÄÄTÄNTÖ	46

LÄHTEET

LIITE

1 Yhden näkymän Mako-templaattiriippuvuudet

1 JOHDANTO

Erilaiset oppimisalustat ovat herättäneet opetuslalla ja opettajien keskuudessa innokasta vuoropuhelua tulevaisuuden työkaluista. Samaan aikaan kouluissa suunnitellaan, kuinka digitaalisilla työkaluilla voitaisiin nykyaikaistaa opettaminen tälle vuosikymmenelle. Joissain osissa Suomea opetuksen modernisoiminen voi tarkoittaa edelliseltä vuosituhannelta siirtymistä tälle vuosituhannelle. Oppimisalustoja on maailmalla tuhansia, mutta Suomessa eräitä tunnettuja oppimisalustoja ovat muun muassa Moodle, Fronter, Peda.net ja Blackboard.

Gemilo on yksi oppimisalustojen tarjoajista. Se on pyrkinyt tuomaan markkinoille tuotteen, jonka avulla opettajat voivat luoda luokkatiloja ja harjoituksia responsiivisella käyttöliittymällä. Lisäksi Gemilon oppimisalusta mahdollistaa yhdessä työstetyn ja palautetun harjoituksen. Gemilon oppimisalusta on tällä hetkellä käytössä Kaarinan kaupungin kouluissa.

Tämän opinnäytetyön kehitystyönä oli koepalvelun näkymien koostaminen. Digitalisoiduille kokeille on kysyntää, sillä niiden automatisoinnilla saataisiin vähennettyä esimerkiksi kvantitatiivisten kielitestien, kuten monivalintakysymysten, tarkistustyötaakkaa. Digitalisoitujen kokeiden lisäetuna on se, että omaa koemenestystään on mahdollista vertailla muiden oppijoiden menestykseen.

Opinnäytetyön tilaajana oli Gemilo. Työn aspektina ei ollut käytettävyys tai visuaalinen rakenne vaan osanäkymien muokattavuus modulaaristen komponenttien avulla. Työ nähdään esiasteena Digipaperi-palvelulle, jonka avulla käyttäjät voivat koostaa tekstistä, kuvasta ja interaktiivisesta mediasta koostettuja paperimaisia näkymiä. Digipaperin alustava mutta tärkeä tavoite olisi antaa käyttäjille valta luoda uutta oppimismateriaalia. Nyt oppimismateriaalin luonti on ollut enimmäkseen kirjakustantajain vastuulla.

Opinnäytetyön tavoitteena oli saada toiminnallinen prototyyppi aikaiseksi. Aikarajaksi asetettiin Opetuksen uusia tuulia -tapahtuma, jonka ajankohta oli 10.10.2013. Työ alkoi syyskuun puolessa välissä työn ohella ja tulisi lopussa muuntumaan täysipäiväiseksi urakaksi.

2 TEKNIIKAT

Vaikka opinnäytetyössä teoreettinen pohja perustuu verkkopalvelutekniikkoihin, on taustalla yleiseen ohjelmistoarkkitehtuuriin liittyvää teoriaa, jotta kyetään luomaan monikäyttöistä ja hyvin jäsenneltyä koodia. Tässä luvussa käydään myös läpi Makotemplaattikieleen liittyvä teoriaa. Sen käytön ymmärtämällä voidaan rakentaa monikäyttöisiä komponentteja, millä on suuri rooli tämän opinnäytetyön käytännön työn osuudessa.

2.1 HTML

HTML tai Hypertext Markup Language on merkintäkieli, jonka avulla voidaan kertoa selaimelle dokumentin oikea esitystapa (Schafer 2010, 6). Merkintäkieli on systematisoitu ja standardisoitu merkintäkäskykanta eli joukko erilaisia esihyväksytyjä merkintämuotoja (Denz & Durant 2003, 6). HTML-merkintäkielellä voidaan kuvata WWW-sivujen sisältöä kulmamerkkien sisällä olevalla koodilla. Näitä kutsutaan tageiksi, mutta paremmin ne tunnetaan HTML-elementteinä. (Järvinen 2003, 270, W3C 2013, b.) HTML:n merkintäkäskykanta on yksinkertaistettu versio SGML-standardin käskykannasta (Järvinen 2003, 270).

HTML:n alkuperäisenä tarkoituksena oli erottaa tekstin sisältö ja ulkoasu toisistaan niin, että dokumentin esitystapa olisi tietokoneiden välillä yhtenäinen. (Järvinen 2003, 270.) Tosin Schafer (2010, 6) väittää, että kielen päämääränä oli tarjota tapa esittää typografisia tyylejä, kuten esimerkiksi kursivointia, alleviivausta tai lihavoitinta. Denz & Durant (2003, 6) väittävät, että HTML:n tarkoituksena on esittää datan piirteitä verkkosivuilla, ei kuvata niinkään datan rakennetta.

HTML:n rakennuspalikat ovat elementtejä. Niillä voi olla attribuutteja, jotka määrittelevät elementtiä itseään ja jotka muuttavat sitä, miten elementit toimivat. Attribuutti koostuu nimestä ja yleensä arvosta. Nimen ja arvon välinen suhde ilmoitetaan yhtäläisyysmerkillä. Attribuutin arvo ilmoitetaan sitaatteihin merkittynä, tosin tämä ei välttämättä ole pakollista, arvoa ei halutessaan tarvitse määritellä. Tilanteissa, joissa attribuutilla on vain nimi, jätetään myös yhtäläisyysmerkki pois. (W3C 2013, b.)

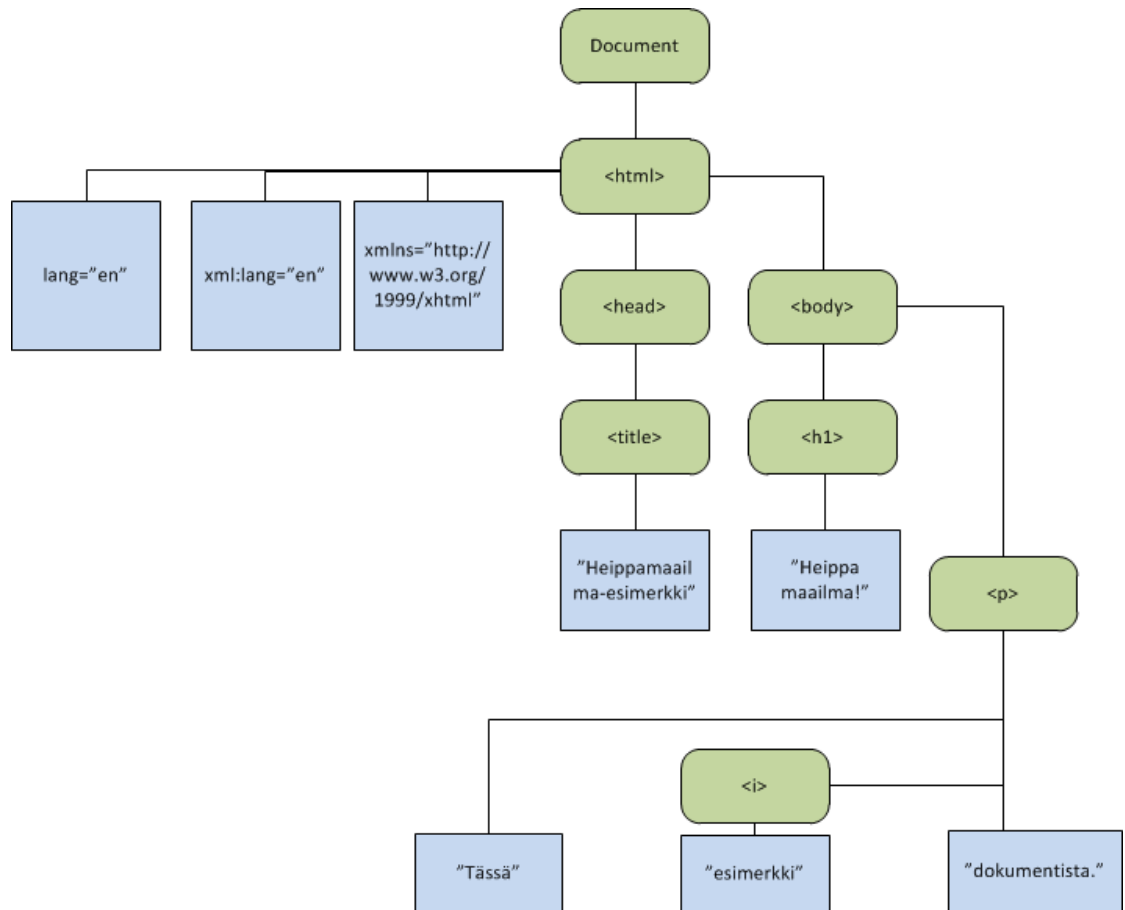
Järvinen (2003, 270) mainitsee harhaanjohtavasti, että jokainen elementti merkataan pareittain. HTML4.01 tai XHTML 1.1 -standardeissa jokainen elementti täytyy olla eksplisiittisesti suljettu joko erillisellä sulkeutuvalla elementtillä tai vinoviivalla ennen alkavan elementin päättävää kulmamerkkiä. (Schafer 2010, xli.) Uudessa HTML5-standardissa yksittäinen elementti ei tarvitse päättävää vinoviivaa (W3C 2013, a).

Validi HTML5-dokumentti sisältää DOCTYPE-tagin, joka estää selainta siirtymästä dokumentin kanssa epäyhteensopivaan renderöintimoodiin. Lisäksi dokumentissa tulee olla html-elementti. (W3C 2013, a.) Html-elementti edustaa HTML-dokumentin juurta, jonka alle määritellään metatiedoista koostuva head-elementti sekä body-elementti, joka sisältää dokumentin esitettävän sisällön (W3C 2013, b).

2.2 DOM

Document Object Model, tai DOM, on ohjelmointikielestä tai alustasta riippumaton rajapinta sisällölle ja sisällön rakenteelle. DOM mahdollistaa sen, että ohjelmat pystyvät lukemaan ja muokkaamaan dokumentin sisältöä, rakennetta tai tyyliä. Mikä olennaista DOM tekee mahdolliseksi myös sen, että muokatun dokumentin muunnos voidaan päivittää vanhentuneen DOMin tilalle. (World Wide Web Consortium 2005.)

Ladatessaan HTML-dokumentin selain jäsentää HTML-dokumentissa määritellyt hierarkiset elementit ja luo solmuolioista kuvan 1 mukaiseksi puu-tietomalliksi. (Lindley 2013, 1.) Selain pakatoi HTML-sivun sisällön hierarkisesti globaalin olion alle, jota pystytään ohjelmallisesti muokkaamaan (Lindley 2013, 2).



KUVA 1. Esimerkkipuun yksinkertaisesta verkkosivun rakenteesta

Kuva 1 koostuu kolmentyyppisestä solmusta. Puun juurisolmu on dokumenttisolmu, joka edustaa koko dokumenttia. HTML-elementtejä muistuttavat solmut ovat elementtisolmuja ja puun lehdet kuvassa 1 ovat tekstisolmuja. Dokumentti- ja elementtisolmu ovat hyvin merkittäviä tekijöitä DOM-rakenteessa. (Flanagan 2011, 363.)

2.3 JQuery

JQuery on JavaScriptin suosituin kirjasto. JQuery pakatoi halutut DOM-elementit elementtijoukkoon, jota on mahdollista jatkokäsitellä helposti. Kaikki DOM-elementtihaut tehdään CSS-valitsinsyntaksilla. Dollarimerkkiä kutsutaan jQuery-olioksi, ja se on globaalisti saatavilla. JQuery-olio on funktio-olio, joten sen kutsuminen merkitään kuten mikä tahansa muu funktiokutsu JavaScriptissa: $\$(\text{...})$. Sulkujen sisällä olevaa parametriä kutsutaan valitsimeksi. (He 2012.)

JQuery ratkaisee ongelman, missä kehittäjä joutuu huomioimaan erilaisten selainten JavaScript-toteutukset ja sen, miten DOM-käsittely toimii kussakin selaimessa. Sen muita hyötyjä ovat ilmaisuteho suhteessa koodirivien määrään sekä aktiivinen kehittäjäkanta. Aktiivinen kehittäjäkanta tarkoittaa kolmannen osapuolen tekemien laajennusten lisäksi myös sitä, että JQuery-kirjaston tuki on vakaalla pohjalla. (Otero ym. 2012, 25–26.)

JQuery UI on kirjastolaajennus jQuery-kirjastolle, ja se pitää sisällään käyttöliittymäkirjaston erilaisia toiminnallisuuksia, erikoistehosteita, animaatioita sekä widgettejä (Otero ym. 2012, 177). Opinnäytetyön kontekstissa käytettäviä widgettejä ovat päivämäärän valitseminen, elementtikokoelman uudelleenlajittelu sekä raahaaminen ja tiputtaminen. Päivämääränvalitsemis-widgetti tunnetaan jQuery UI -kirjastossa nimellä datepicker, raahaus-widgetti nimellä draggable, tiputus-widgetti nimellä droppable sekä lajittelu-widgetti nimellä sortable. Näistä sortable-widgetti määritellään sellaiseksi joukoksi elementtejä, jotka ovat uudelleenlajiteltavia komponentteja (Otero ym. 2012, 206).

2.4 Ajax

Ajax ei ole teknologia vaan kokoelma useita teknologioita (Garrett 2005). Ajax syntyi alustavassa muodossaan ensimmäisen kerran selaimen Internet Explorer versiossa 5.0 (Dutta 2006). Garrett määritteli käsitteen aikoinaan artikkelissaan *Ajax: a New Approach to Web Applications* (2005) ja luetteli kyseiset verkkoteknologiat sisältymään kyseisen käsitteen alle:

1. Standardipohjaiset merkintäkielet XHTML ja CSS
2. Document Object Modelin eli DOMin
3. XML- ja XSLT-merkintäkielet datansiirto- ja datamanipulaatioformaattina
4. Ennaltamääritellyn asynkronisen datanhakemisolion XMLHttpRequest
5. komentosarjakieli JavaScriptin.

Ajax eliminoi perinteisen verkkosivumallin, jossa käyttäjään tietokone tekee pyynnön palvelimen tiettyyn osoitteeseen ja renderoi sivun kokonaan uudelleen. Ajax-pyyntö on asynkroninen: pyyntö tai pyynnot kohdistuvat tiettyyn osoitteeseen, mutta

ne eivät pysäytä sivun toimintaa pyyntöjen ajaksi. Saatu tieto käsitellään selaimen implementoiman Ajax-rajapintaolio XMLHttpRequest avulla. (Garrett 2005.)

Nykyisin Ajaxin datansiirtoformaattina käytetään JSON-formaattia. Kahden formaatin, XML-merkintäkielen ja JSON-formaatin, välillä on eroja. Uudemman JSONin hyöty on siinä, että se ilmaisukykyisempi pienemmällä määrällä merkkejä. Lisäksi se myötäilee syntaksiltaan JavaScriptia. (Koch 2013.)

2.5 JavaScript

Tunnetuin implementaatio ECMAScript-standardista on JavaScript, jonka väitetään olevan maailman levinnein ohjelmointikieli. Siinä missä HTML määrittelee sisällön rakenteen ja CSS määrittelee rakenteelle esitysmuodon, niin JavaScriptin tarkoituksena on manipuloida sivuston käyttäytymistä. (Flanagan 2011, 1.)

JavaScriptissa oliot ovat muuttuvia avaimellisia kokoelmia. Primitiivityyppejä ovat numerot, merkkijonotaulukot, totuusarvot, *null* ja *undefined*. Muutoin kaikki on JavaScriptissa olioita. Esimerkiksi taulukot, funktiot ja säännölliset lausekkeet ovat olioita. (JavaScript: The Good Parts 2008, 20.) Olioilta voidaan periä prototyypillisesti. Prototyypillisuus tarkoittaa, ettei kieli instansioi luokista olioita, vaan oliot perivät ominaisuuksia suoraan toisilta olioilta. (JavaScript: The Good Parts 2008, 3.) Olioiden ja taulukoiden arvoja voidaan muuttaa, mutta numeroita, totuusarvoja, tyhjää, arvoa *undefined* tai merkkijonoa ei voi muuttaa, kun ne on kerran luotu. JavaScriptin muuttujat ovat tyyppivapaita: yhden muuttujan arvo voi vaihdella tyyppistä toiseen, eikä muuttujan tyyppiä tarvitse ilmoittaa etukäteen. (Flanagan 2011, 30–31.)

Muuttujilla voi olla joko koko ohjelman kattava, globaali näkyvyysalue. Tällöin muuttujaa voidaan tarkastella ja muokata mistä tahansa ohjelmaa. Funktiossa määritellyt muuttujat näkyvät vain funktiorutiinin sisällä, ne ovat paikallisia muuttujia ja niillä on tällöin paikallinen näkyvyysalue. Funktion parametrimuuttujat toimivat samalla tavalla. Jos globaali ja paikallinen muuttuja jakavat yhteisen muuttujanimen, paikallinen arvo asetetaan muuttujan arvoksi. (Flanagan 2011, 53.)

Eräs JavaScriptin mielenkiintoisemmista ominaisuuksista on sisäiset funktiot, jotka näkyvyysalueen ansiosta pääsevät käsiksi ulkoisen funktion määrittelemiin muuttujiin

(JavaScript: The Good Parts 2008, 37). Näkyvyysketju on olennainen käsite sulkeumien ymmärtämiselle. Seuraavaksi käsittelemme lyhyesti, mikä on näkyvyysketju.

Olennaisesti kaikilla funktioilla tai globaalilla, funktioiden ulkopuolisella koodilla on liitettyä oma näkyvyysketjunsä. Ketjussa määritellään lista tai ketju olioita, jotka olio kykenee näkemään. Tarkastaessaan muuttujan arvoa JavaScript käy läpi ketjun oliot ja niiden arvot ensimmäisestä viimeiseen, kunnes löydetään muuttujan arvo tai annetaan virhe. Kun funktiota kutsutaan, JavaScript luo uuden olion säilöäkseen paikallisia muuttujia siinä ja samalla liittyy itseensä näkyvyysketjun, johon on lisätty äsken kutsutun funktion olio. Jos sisäisen funktion ulompaa funktiota kutsutaan, joudutaan sisäfunktion määrittelemään uudestaan. Välttämättä sisäfunktion näkyvyysketju ei tällöin ole samanlainen jokaisella kutsukerralla. (Flanagan 2011, 55–56.)

Näkyvyysketju mahdollistaa sulkeuman. JavaScriptissa kaikki funktiot ovat sulkeumia, koska ne ovat olioita ja niihin on assosioituna oma näkyvyysketju. Mielenkiintoisempi tapaus sulkeumasta on tilanne, jossa näkyvyysketju poikkeaa olion aiemmasta näkyvyysketjusta. Näin voi käydä, kun ulommasta funktiosta palautetaan sisäinen funktio. (Flanagan 2011, 180–182.)

```
var uniikkiNumero = (function() { ..... // Määrittele ja kutsu heti
..... // var laskuri = 0; ..... // Yksityinen arvo
..... // return function() { ..... // Julkinen rajapinta
..... //     return laskuri++;
..... // }
..... // }());

> uniikkiNumero();
=> 0
> uniikkiNumero();
=> 1
> uniikkiNumero();
=> 2
```

KUVA 2. Esimerkki sulkeumasta JavaScript-ohjelmointikielessä

Kuvassa 2 määritellään itseään kutsuva funktio, joka palauttaa funktion rajapinnan eli oman sisäisen funktion muuttujaan uniikkiNumero. Millään muulla rutiinilla koodissa ei ole luku- tai muokkausoikeutta laskuri-muuttujan arvoon. (Flanagan 2011, 182.) Tämä tekee sulkeumista hyödyllisiä sellaisissa tilanteissa, kun halutaan eristää olioiden jäsenmuuttujien näkyvyyttä ulkopuolelle. Eri muuttujiin asetetut instanssit

uniikkiNumero-funktiosta jakavat kuitenkin laskurimuuttujan tilan keskenään, kuten huomataan kuvasta 3.

```
> var foo = uniikkiNumero;
> var bar = uniikkiNumero;
> foo();
=> 0
> bar();
=> 1
```

KUVA 3. Sulkeuman modulaarisuuden havainnollistaminen

Kuvassa 2 muuttujaan asetettu funktio on eristetty suluilla, jotta funktio eristäisi oman nimiavaruutensa globaalista nimiavaruudesta. JavaScriptissa luokat voivat noudattaa kuvan 2 moduulisuunnittelumallia, mikä helpottaa koodiorganisointia. Crockford (JavaScript: The Good Parts 2008, 41) määrittelee moduulin funktioksi tai olioksi, jonka palautusarvona on olio ja joka esittää funktion ulkopuolelle julkisen rajapinnan mutta joka piilottaa globaalilta näkyvyysalueelta muuttujiensa tilan sekä toteutuksen. Näitä kahta piirrettä kutsutaan vastaavasti kapseloinniksi ja abstraktioksi. Jos abstraktio eli implementaation piilotustaso on matala, seuraa kapseloinnin taso perässä - ne ovat siis toisiinsa sidottuja. Eräs merkittävä tapa parantaa abstraktiota ja kapselointia on minimoida pääsy muuttujiin. (McConnell 2004, 139.) Moduulit eliminoivat JavaScriptista sovelluskohtaiset globaalit muuttujat lähes kokonaan (JavaScript: The Good Parts 2008, 41).

2.6 ECMAScript

ECMAScript-standardi pohjautuu useaan teknologiaan, joista kaksi tunnetuinta ovat Netscapen kehittämä JavaScript ja Microsoftin JScript (Ecma International 2011, 11). Kielen alkuperäinen kehittäjä Brendan Eich oli töissä Netscapella, kun häntä vuonna 1995 pyydettiin työstämään kymmenessä päivässä toimiva prototyyppi ohjelmointikielestä, joka muistuttaisi Javaa (Severance 2012). JavaScript ilmestyi ensimmäisen kerran Navigator 2.0 -selaimessa, ja Microsoftin JScript ilmestyi Internet Explorer 3.0 -selaimessa (Ecma International 2011, 11). Vaikka nimeämiskäytänteitä ECMAScript-standardin implementaatioille on monia, on yleiseksi käytänteeksi muodostunut nimitys JavaScript. Viime vuosikymmenen aikana kaikki selaimet ovat tukeneet ECMAScript-standardista implementoitua kolmatta painosta. Mozilla käyttää

tästä versionimitystä JavaScript 1.5. Uutta ECMAScript-standardin viidettä painosta voi nähdä kutsuttavan JavaScriptin versionumerolla 1.8. (Flanagan 2011, 2.)

ECMAScript-standardi määrittelee ohjelmointikielen, jossa pääsääntöisesti kaikki perustuu olioihin lukuun ottamatta primitiivi-arvoja, joihin kuuluvat *Undefined*, *Null*, *Boolean*, *Number* tai *String*. ECMAScript ei kuitenkaan käytä vastaavantapaisia luokkia kuin mitä Javassa tai C#-ohjelmointikielessä esiintyy. Olion pystyy muodostamaan lyhennetyllä merkintätavalla, jonka implementaatio JavaScriptissa on `{}`, tai pidempikaavaisen konstruktorin avulla. Konstruktori ECMAScriptissa on funktio, joka sisältää ominaisuuden *prototype*. Tämä ominaisuus on perusta prototyypiselle perinnälle sekä yhteisille ominaisuuksille. (ECMA International 2011, 3.)

ECMAScript ei kuitenkaan ole laskennallisesti itsenäinen, eikä standardin spesifikaatiossa määritellä provisioita ulkoisen syötteen käsittelylle tai tiedon viennille ulos. Standardi olettaa, että kielen implementaatioissa käytetään spesifikaation määrittelyjen lisäksi tiettyjä ympäristökeskeisiä isäntäolioita, joita ECMAScript-toteutus voi kutsua. (ECMA International 2011, 1.) Isäntäolio on mikä tahansa spesifikaation ulkopuolinen toteutus oliosta (ECMA International 2011, 5). JavaScriptissa tunnettuja isäntäolioita ovat *window*, *document* ja *console*.

2.7 Python

Python on vahvasti olio-ohjelmointiin perustuva tulkettava ja heikosti tyyпитetty ohjelmointikieli. Python käännetään siis ohjelman ajoaikana Javan tavukoodia vastaavalle Pythonin omalle välikielelle. Pythonin heikko tyyпитitys tarkoittaa sitä, etteivät muuttujien arvot ole sidottu tiettyyn muuttujatyyppiin, esimerkiksi kokonaislukuun tai merkkijonotaulukkoon. (Kokkarinen 2004, 253–254.)

Suoritettavaa Python-ohjelmaa kutsutaan moduuliksi (Kokkarinen 2004, 254). Pythonissa moduuli määrittelee oman nimiavaruutensa, jonka alla sijaitsee kaikki kyseisen moduulin suorituksen aikana määritellyt muuttujanimet. Muuttujanimi syntyy silloin, kun siihen joko asetetaan eksplisiittisesti arvo tai kun muuttuja asetetaan *def*-lauseella osoittamaan lennossa rakennettavaa funktio-oliota. Kun jotain

tällaista *def*-lauseella määriteltyä funktiota kutsutaan, funktion vartalon sisällä olevat lauseet suoritetaan ylhäältä alas -järjestyksessä.

Pythonissa kaikki luokat ovat olioita, niin sanottuja luokkaolioita, jotka voivat sisältää metodeja ja jäsenmuuttujia (Kokkarinen 2004, 281). Python-versio 2.2 tutustutti uuden ominaisuuden, jossa luokan voi periyttää sisäänrakennetuilla tietotyypeillä *object*, *str*, *list*, *int* tai esimerkiksi *dict*. Tietotyyppi *object* on pohjaluokka kaikille edellä mainituille ja mainitsemattomille sisäänrakennetuille tietotyypeille. Siitä voidaan periyttää silloin, kun yksikään sisäänrakennettu tietotyyppi ei sovi käyttötarkoituksiin. (Python 2013.) Funktiot, moduulit ja esimerkiksi perinteiset primitiivi-argumentit kuten kokonaisluvut sekä muut muuttujat ovat myös olioita (Kokkarinen 2004, 281). Kielen kehittäjä van Rossum (2009) väittää, että Pythonin oliot ovat tästä johtuen ensiluokkaisia: kukin olio voidaan niin sanottuna tasa-arvoisena kansalaisena sijoittaa muuttujaan, kokoelmaan, sanakirjaan tai vaikka passata argumenttina.

Funktio-oliolla voi olla paikallisia muuttujia, jotka ovat olemassa vain kyseisen funktion suorituksen verran. Jokainen funktiokutsu synnyttää uuden nimiavaruuden, jossa on kutsutun funktion paikalliset muuttujat. Globaalin muuttujan määrittelee yksi sääntö: jos funktio sisältää muuttujaan sijoittamisen joko eksplisiittisesti tai implisiittisesti, muuttuja on paikallinen, muussa tapauksessa muuttuja on globaali. Koska kaikki Pythonin muuttujat ovat viitteitä olioihin, argumenttina oleva olio viittaa funktion sisällä ja ulkopuolella samaan olioon. Näin funktion sisällä tehdyt muutokset heijastuvat globaalille tasolle. (Kokkarinen 2004, 275.)

Opinnäytetyön kontekstissa käytettävät tietorakenteet koostuvat yleisesti listoista tai sanakirjoista. Lista ottaa vaikutteita perinteisten ohjelmointikielten listoista ja taulukoista. Listan alkiot ovat tyypittämättömiä viitemuuttujia, mikä mahdollistaa erityyppisten olioiden käsittelyn listassa. Listan yksinkertaisin muodostussyntaksi on tyhjät hakasulkeet `[]`, missä alkiot erotetaan toisistaan pilkuilla. (Kokkarinen 2004, 265.) Lista on erikoistapaus yleisemmästä sarjasta. Sarja on jossain muodossa esitetty jono alkiota, jota voidaan indeksoida, siivuttaa ja jonka alkiot voidaan käydä yksitellen läpi `for`-silmukalla. Myös merkkijonot ovat sarjoja. Monikko on kiinteä lista, jonka sisältöä ja pituutta ei voida muuttaa. Monikko voidaan muodostaa tavallisilla sulkeilla `(foo,bar)`. (Kokkarinen 2004, 269.)

Sanakirja voi sisältää avaimen ja arvon muodostamia assosiaatioita. Toisin sanoen sanakirjalla on mahdollista ylläpitää kokoelmaa avain-arvo-pareista. Sanakirja määrittellään aaltosulkeilla `{'foo': 'bar'}`, missä avain-arvo-parit erotetaan pilkulla. Sanakirjojen avaimille ei ole määritelty järjestystä. (Kokkarinen 2004, 273.)

Funktion parametreille voidaan määrittää oletusarvoja. Syntaksi oletusarvon asettamiselle kuuluu `foo='bar'`. Nimetyillä parametreilla voidaan simuloida funktioiden ylikuormittamista. Kuitenkaan nimetyn parametrin jälkeen ei saada esitellä tavallisia parametrimuuttujia, koska parametrilistan järjestys ei tällöin olisi yksiselitteinen. (Kokkarinen 2004, 277.)

Funktio voidaan myös asettaa vastaanottamaan mielivaltainen määrä parametreja. Jos funktion parametrilistan viimeisen parametrin nimen edessä on asteriski `*`, tämä parametri saa monikkona arvokseen kaikki funktiokutsussa annetut ylimääräiset parametrit. Jotta sama voidaan toteuttaa nimetyillä parametreilla, täytyy viimeisen parametrin nimen eteen sijoittaa kaksi asteriskia `**`. Tällöin arvot sisällytetään sanakirjaan. Funktio voi vaatia sekä mielivaltaisen määrän nimettömiä parametreja että mielivaltaisen määrän nimettyjä parametreja, kuitenkin tuossa nimenomaisessa järjestyksessä. (Kokkarinen 2004, 277–278.)

2.8 Mako Templates

Mako Templates on templaattikirjasto. Se mahdollistaa verkkosivujen koostamisen pienistä funktioista – blokeista – ja tukee Mako-tiedostojen periyttämistä. Mako yhdistää HTML-merkkaukielen, Pythonin sekä Makon omaa syntaksia muodostaakseen oman täsmäkielen. (Mako Templates For Python - Mako 0.9.1 Documentation 2013.) Mako-templaatti voidaan jäsentää mielivaltaisesti määritellystä tekstisisällöstä riippumatta dokumentin syntaksista. Templaatti käännetään ajonaikaista suoritusta varten Python-koodiksi. (Syntax - Mako 0.9.1 Documentation 2013.)

Mako-templaatin yksinkertaisin lauseke koostuu eteensijoitetusta dollarimerkistä sekä aaltosulkeista, jonka sisällä oleva koodi evaluoidaan Python-syntaksin mukaisesti, vaikka Mako antaa mahdollisuuden sisällyttää kuvan 4 tapaisia suodatusfunktioita.

Niiden käyttö on kuitenkin vähäistä Gemilo Socialissa. (Syntax - Mako 0.9.1 Documentation 2013.)

```
#{ "Tämä teksti tulee ulos välilyönnittömänä" | u }
=> 'Tämä+teksti+tulee+ulos+välilyönnittämänä'
```

KUVA 4. Suodatusfunktion käyttöesimerkki

Mako sallii ohjausrakenteita, kuten *if/else*, *while*, *for* sekä *try/except*. Mako-ohjausrakenteiden eteen lisätään aina prosenttimerkki, ohjausrakenne päätetään kuvan 5 lailla *end*-etuliitteellä. Jokaisella templaatilla on kuvan 5 mukainen kontekstimuuttuja *c*, johon voidaan kontrollerissa passata näkymille oleellista tietoa. (Syntax - Mako 0.9.1 Documentation 2013.)

```
% if c.can_view:
... <p>Secret</p>
% endif
```

KUVA 5. If-rakenteen syntaksi kontekstioliolla Mako-templaatussa

Python-koodia voidaan käsitellä templaatussa itsessään. Kuvassa 6 esitellään Mako-funktion kutsuminen templaatussa määritellyllä taulukkoparametrimuuttujalla. (Syntax - Mako 0.9.1 Documentation 2013.) Gemilo Socialissa Python-koodin määrittelyä yritetään välttää näkymissä, vaikka prototyypitasolla sitä esiintyykin.

```
<%
... params = [ (u"Sisältö 1"),
...             (u"Sisältö 2"),
...             (u"Sisältö 3") ]
%>

#{ tulosta sisalto(params) }
```

KUVA 6. Python-koodin määrittäminen suoraan Mako-templaattiin

Mako-templaattikirjaston `<%def>`-tagilla merkitään mikä tahansa teksti- tai koodiblokki renderöintifunktioksi. `<%def>`-tagin muodostussyntaksi muistuttaa Pythonin funktion muodostamissyntaksia. Kuvassa 7 esitellään yksinkertainen funktio

ja funktiokutsu, jolle voidaan halutessa määritellä parametrivuttuja tai nimettyjä parametrivuttuja. (Syntax - Mako 0.9.1 Documentation 2013.)

```
<%def name="heippa()">
... <p>Heippa maailma!</p>
</%def>

Def: ${heippa()}
```

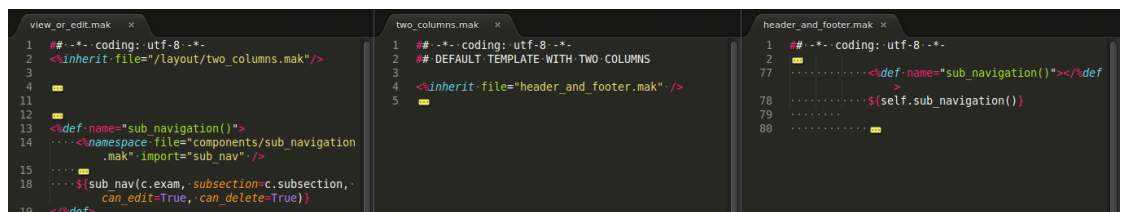
KUVA 7. Funktion määrittely ja sen kutsuminen Mako-templaattissa

`<%namespace>`-tagi vastaa Pythonin `import`-komentoa. Sen avulla pystytään tuomaan renderöintifunktioita ja metadataa toisista templaattitiedostoista (Syntax - Mako 0.9.1 Documentation 2013). Lisäksi se mahdollistaa Gemilo Socialissa generisten komponenttien luomisen ja tuomisen kuvan 8 kaltaisella tavalla.

```
<%def name="scripts()">
... <%namespace file="/layout/components/html_editor.mak" import="add_html_editor" />
... ${add_html_editor()}
</%def>
```

KUVA 8. Funktioiden tuominen nimiavaruuksien avulla Mako-templaattissa

Mako-templaattit tukevat templaattien periyttämistä. Tämä mahdollistaa kahden tai useamman templaatin sisällön ja renderöintifunktioiden sekoittumisen keskenään. (Inheritance - Mako 0.9.1 Documentation 2013.) Kuvassa 9 käydään läpi pieni Mako-periytymisketjuesimerkki.



```
view_or_edit.mak x
1 ## -*- coding: utf-8 -*-
2 <inherit file="/layout/two_columns.mak"/>
3
4
11
12
13 <def name="sub_navigation()">
14 ... <%namespace file="components/sub_navigation
15 .mak" import="sub_nav" />
16
17 ... ${sub_nav(c.exam, subsection=c.subsection,
18 can_edit=True, can_delete=True)}
19 </def>
```

```
two_columns.mak x
1 ## -*- coding: utf-8 -*-
2 ## DEFAULT TEMPLATE WITH TWO COLUMNS
3
4 <inherit file="header_and_footer.mak"/>
5
```

```
header_and_footer.mak x
1 ## -*- coding: utf-8 -*-
2
77 ..... <def name="sub_navigation()"></def>
78 ..... <self.sub_navigation()>
79 .....
80
```

KUVA 9. Mako-templaatin periytymisesimerkki

Kun kuvan 9 `view_or_edit`-templaatti renderöidään, ohjaus siirtyy välittömästi `two_columns`-templaattiin, jonne on myös määritelty periyttävä templaatti `header_and_footer`. Ohjaus siirtyy välittömästi sinne `two_columns`-templaatin renderöityessä. Templaattiin `header_and_footer` on määritelty tyhjä renderöintifunktio `sub_navigation`, jota kutsutaan heti määrittelyn jälkeen. Koska periytymisketjussa

templaatit kykenevät näkemään toistensa renderöintifunktioita, ylikuormittaa *view_or_edit*-templaatussa määritelty *sub_navigation* tyhjän funktion. (Inheritance - Mako 0.9.1. Documentation 2013.) Näin hierarkisesti ylemmille tasoille voidaan luoda paikanvaraajafunktioita, joiden ei kuvan 9 välttämättä tarvitse olla tyhjänä ja jotka voidaan ylikuormittaa näkymäkohtaisesti uudelleen.

2.9 Cascading Style Sheets

Cascading Style Sheets, tai CSS, on merkintäkielillä kirjoitettujen dokumenttien kuvaamista varten suunniteltu deklarativinen kieli. CSS-kielillä voidaan siis määrittellä dokumentin muoto, elementtien visuaalinen rakenne sekä värimaailma. (York & Pouncey 2011, 35.) CSS-kielessä tyyliteltävä elementti määritellään valitsinlausekkeella, jonka jälkeen aaltosulkeiden sisälle määritellään tyyliä määritteleviä ominaisuusarvopareja, missä ominaisuus määrittelee elementin tietyn visuaalisen ominaisuuden ja annettu arvo tai arvot ovat sidonnaisia ominaisuuden hyväksymään arvoon (Schafer 2010, 406).

Valitsimet ovat kaavoja, joita kehittäjä voi määrittellä tähdätäkseen haluttua tai haluttuja elementtejä esitettävässä dokumentissa. Valitsin sisältää yleisimmiten elementtityypin, kuten `html` tai `body`, universaalinen valitsinmerkin eli asteriskin, luokkanimen, `id`-attribuutin tai attribuutti-arvo-parin tahi pelkästään attribuutin. Hyvin usein näitä valitsinosia kirjoitetaan peräkkäin välilyönnein tai `>`-merkillä erotettuna, jossa välilyönti spesifioi kahden välisen elementin syvän hierarkisen perinnöllisyyden ja `>`-merkki välittömän perinnöllisyyden. (Schafer 2010, 407-412.)

CSS-kielessä tyylit voivat kasautua toistensa päälle: tästä nimitys Cascading Style Sheets. Kuten Schafer (2010, 436) listaa W3C-organisaation määrittelemästä spesifikaatiosta, CSS-kielessä valitsimen lauseen tarkkuutta painotetaan tietyin periaattein:

1. Lasketaan ID-attribuuttien määrä valitsimessa ja asetetaan määrä muuttujaan A
2. Lasketaan muut valitsimessa esiintyvien attribuuttien sekä pseudoluokkien lukumäärä ja lisätään nämä muuttujaan B
3. Lasketaan elementtityyppien nimien sekä pseudoelementtien lukumäärä ja lisätään nämä muuttujaan C

4. Lisätään muuttujien A, B ja C numerot peräkkäin, jolloin saadaan kyseisen valitsinlauseen tarkkuus.

Jos tai kun valitsinlauseen tarkkuus on korkeampi kuin aiempi tyylimääritelmä, tarkempi määritelmä voittaa. Kun tarkkuus on kahden tai useamman kilpailevan valitsimen välillä tasapeli, jälkimmäinen tapahtuva määritelmä sovelletaan dokumenttiin (Schafer 2010, 435). Koska valitsimien tarkkuus halutaan pitää mahdollisimman yksinkertaisena, on yleistä pitää valitsimen valitsinosien määrä hyvin pienenä.

2.10 MVC

Gamman ym. (2001, 4) määritelmä MVC- tai Model-View-Controller-arkkitehtuurityylin komponenteista on seuraava: Model tai Sisältö on sovelluksen olio, View tai Näkymä on esitettävä näkymä ja Controller - Kontrolleri - vastaa sitä tapaa, jolla käyttöliittymä reagoi käyttäjän antamaan syötteeseen. Voidaan vastaavasti puhua datan, käyttöliittymän ja bisneslogiikan eriyttämisestä (Weisfield 2009, 289-290).

Aikaisemmissa design-paradigmoissa MVC-paradigman eri komponentteja saatettiin käyttää vaihtuvasti toisiinsa paritettuna, joskus kaikkia kolmea komponenttia yhdessä komponentissa. MVC-malli ratkaisee parituvuusongelman eriyttämällä nämä kolme komponenttia toisistaan, näin jokaiselle komponentille saadaan selkeä oma rajapintansa. Ideaalisti toteutettuna ulkoasuun tehtävät muutokset voidaan tehdä vaikuttamatta sovelluksen bisneslogiikkaan tai dataan - tai toisinpäin sillä oletuksella, että rajapintayhteydet eri komponentteihin säilyy ennallaan. (Weisfield 2009, 289-290.)

2.11 Rutiinit

Rutiini on yksittäinen metodi tai tapahtuma, jolla on vain yksi tavoite (McConnell 2004, 161). Tämän opinnäytetyön kontekstissa rutiini voi olla JavaScript-ohjelmointikielen tai Mako-templaattikielen funktio, mutta rutiini-käsitettä sovelletaan myös olio-ohjelmoinnin luokkien metodeihin. Hyvän rutiinin määrittelee McConnellin (2004, 164-165) mukaan seuraava lista tekijöitä:

1. Rutiinille on oltava selkeä, abstrakti nimi.
2. Rutiinin tarkoituksena on välttää saman koodin toistuminen, toisin sanoen keskittää yhteistä koodia hyödyntävät toiminnallisuudet yhden rutiinin alle.
3. Rutiinin on tuettava aliluokitusta. Rutiinin on oltava helposti ylikuormitettavissa.
4. Rutiinien tulisi piilottaa muiden rutiinien järjestyksiin riippuvainen käyttö. Rutiini itse ei siis saa tehdä oletuksia itsensä ulkopuolella tapahtuvasta invokaatiojärjestyksestä.
5. Joissain tapauksissa rutiinin olisi syytä eristää järjestelmä- tai ympäristösidonnaiset – kuten ei-standardit kieliominaisuudet, järjestelmäriippuvuudet tai käyttöjärjestelmäriippuvuudet – itseensä. Näin identifioidaan sellaiset rutiinit, joiden toiminnallisuus täytyy korjata siirrettäessä sovellusta vaikkapa toiseen tuotantoympäristöön.
6. Rutiinien avulla voidaan piilottaa monimutkaiset totuuslausekepuut.
7. Keskitetyt rutiinit luovat hyvän pohjan optimoinnille ja refaktoroinnille.

Hyvät rutiinit edistävät hyviä suunnitteluperiaatteita ja ohjelmointityötä. Mikkosen ja Haikalan (2011, 180–181) mukaan yksinkertaisuus ja suoraviivaisuus, osittaminen, lokaalisuus, abstraktioiden hyödyntäminen, rajapinnat ja yhdenmukainen toteutusfilosofia edesauttavat selkeän ja ymmärrettävän ohjelmistokoodin parissa työskentelyä. Osittamiseksi Mikkonen & Haikala (2011, 181) ehdottavat, että kokonaisuus jaetaan saman tason toisistaan riippumattomiin osakokonaisuuksiin. Lokaalisuus tarkoittaa osittamisen suunnittelua niin, että suunnittelupäätökset kapseloidaan osien sisään niin hyvin kuin mahdollista. Näin kapseloimalla tulevat muutokset eivät kohdistu ohjelmiston muihin osiin. (Mikkonen & Haikala 2011, 182.)

Joskus rutiinin luominen pelkän kahden-kolmen koodirivin operaatioita varten tuntuu asioiden monimutkaistamiselta. Pienetkin rutiinit kuitenkin lisäävät luettavuutta, olettaen että rutiinin nimi ilmaisee tarkoituksensa selkeästi. Joskus pienet rutiinit paisuvat reunatapausten johdosta, jolloin ne kuitenkin piilottavat toiminnallisuutensa paisumisen. (McConnell 2004, 166.) Epäolennaisen ongelman ratkaisuun liittyvän toteutuksen piilottamista kutsutaan tiedon kätkenäksi (Mikkonen & Haikala 2011, 182).

2.12 Software-as-a-Service

SaaS eli Software as a Service on yksi kolmesta erityyppisestä pilvilaskennallisesta palvelutyypistä. Se viittaa sovelluspalveluiden tarjontaan pilviverkossa eli verkossa, jonka vastuu ei kuulu asiakkaalle itselleen. (BCS The Chartered Institute for IT 2012, 2.)

Historiallisesti ennen pilvilaskennan käsitettä ohjelmistopalveluita toimitettiin asiakkaille etäpalvelinten avulla Internetin välityksellä. Tällöin se tunnettiin käsitteellä Application Service Provision tai ASP. ASP mallina epäonnistui kuitenkin siksi, että sen asennus ja konfigurointi asiakaspäädystä oli hankalaa ja että ASP-mallissa yksi toimittaja pyrki palvelemaan vain yhtä asiakasorganisaatiota toisin kuin nykyinen pilvilaskentamalli, missä yksi toimittaja palvelee montaa asiakasta yhdellä ohjelmistoinstanssilla. (BCS The Chartered Institute for IT 2012, 3.)

SaaS nousi pinnalle vuonna 2001. Se toi käytänteen keskitetystä ohjelmistosta, jota voitaisiin päivittää keskitetysti ja instanssikohtaisesti eikä asiakaskohtaisesti päivityspakettien avulla. Siinä missä sovellusta varten oli myönnetty lisenssi sovelluspaketin käyttöä varten, allekirjoitettaisiin SaaS-mallin alla palvelukäyttösopimus verkkopohjaisten palveluiden käyttöä varten. (BCS The Chartered Institute for IT 2012, 3-4.)

Pilvipalveluiden hyödyiksi nähdään BCS The Chartered Institute for IT -järjestön (2012, 8–9) mukaan yleensä viisi tekijää:

1. Hyvä hinta-laatu-suhde. Yleensä hintamallit perustuvat kulutukseen, esimerkiksi käyttäjien, laitteiden, instanssien tai servereiden määrään.
2. Helposti otettavissa käyttöön. SaaS-pilvipalvelun asennus, ylläpito ja laitteiston huolto on jätetty palvelutoimittajalle.
3. Päivitettävyyys. Palvelutoimittajalle on nopeaa päivittää, korjata ja lisätä uusia ominaisuuksia ohjelmistoon.
4. Skaalautuvuus. Hintamallit ja laitteisto tukevat äkillisen palvelukysynnän nousua. Erityisesti hintamallit rakennetaan yleensä joustamaan kysynnän mukaan.

5. Saavutettavuus. SaaS mahdollistaa sovelluksen käyttämisen mistä tahansa sijainnista, missä käyttäjällä on pääsy Internetiin.

Koska data on keskitetty mutta käyttäjät eivät, korostuu verkon käyttö, mikä voi aiheuttaa haasteita esimerkiksi WAN-siirtonopeudessa tai -latenssissa. Toinen tekninen haaste pilvipalveluissa on vikasietoisen järjestelmän aikaansaaminen. Paikallisesti asennetuissa palveluissa offline-käyttö on mahdollista, pilvipalveluiden kohdalla toimimaton Internet-yhteys tai palvelun alhaalla oleminen tarkoittaa tuottamatonta seisonta-aikaa. (BCS The Chartered Institute for IT 2012, 9.)

Pilvipalvelun käyttöönoton haasteena on lisäksi turvallisuusaspekti, jota puolet asiakkaista ei BCS The Chartered Institute for IT -järjestön mukaan (2012, 9) huomioi kunnolla. Järjestö (2012, 9–10) luokittelee seuraavat tietoturvaan liittyvät kysymykset tärkeiksi asiakkaan kannalta:

- Voidaanko pilvipalvelu integroida osaksi paikallisia IT-järjestelmiä?
- Onko koulutus, dokumenttitemplaattien käännoistyö sekä automaatio hinnoiteltu järjestelmämigraatioon?
- Kuinka dokumentoidut proseduurit muuttuvat?
- Miten henkilökunta ja järjestelmäadministraattorit koulutetaan?
- Minkälainen suunnitelma tehdään järjestelmästä poistumista varten? Tarjoaako pilvipalvelu siirrettävää formaattia palvelussa sijaitsevasta datasta, ja saako palvelusta ulos olennaisen bisneskriittisen tiedon?

3 KEHITYSALUSTA JA KEHITTÄMISTEHTÄVÄ

Tässä luvussa esitellään kehittämistehtävä ja siihen liittyvät asiakasvaatimukset. Käydään läpi katsantona Gemilo Social järjestelmänä. Lisäksi tarkastellaan kehittämistehtävässä vaadittavia sisältö- ja kontrolleriluokkarakenteita ja otetaan esille muutamia keskeisessä roolissa olevia toiminnallisuksia.

3.1 Gemilo ja Gemilo Social

Gemilo on yhteistyöalustapalvelu, jonka keskeisinä ominaisuuksina ovat aktiviteettiseinä, kaksijakoinen näkymä sekä Gemilon käyttöönotto SaaS-palveluna.

Aktiviteettiseinä, tai virta, mahdollistaa käyttäjille minkä tahansa sisältönäkymän kommentoimisen ja keskustelun. Se hyödyntää Gemiloon rakennettua kahtiajaettua näkymää, jossa virta esitetään komplementoivana osana oikean puolen päänäkymälle. Virta kokoaa koko palvelussa tapahtuneet muutokset työpöydälle yhteen paikkaan. Virta voi kerätä myös esimerkiksi yhdessä projektissa tapahtuneet muutokset projektinäkymään. Pääsääntöisesti vasemmalla puolella esiintyvä osanäkymä on hierarkiselta tasoltaan korkeampi kuin oikealla puolella esitettävä osanäkymä.

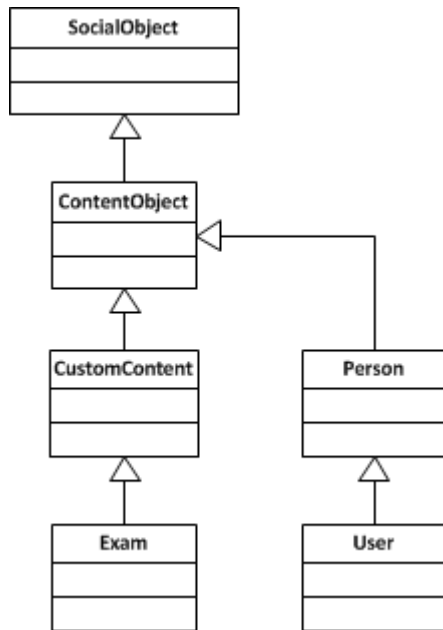
Gemilon muita ominaisuuksia ovat muun muassa seuraavat:

- Henkilökohtainen työpöytä ja profiili
- Projektinhallintaominaisuudet
- Luokkasivut
- Kurssit ja harjoitukset, jotka poikkeavat myöhemmin tässä opinnäytetyössä esitetyistä kokeista
- Blogit ja tilapäivitykset
- Tapahtumakalenteri
- Ryhmät
- Käyttöoikeuksien hallinta.

Gemilo-palvelun lähdekoodista puhutaan Gemilo Socialina. Gemilo Social on rakennettu Python-alustan päälle. Python-versiona käytetään vähintään julkaisunumeron 2 revisiota 6 pois lukien julkaisunumero 3. Järjestelmä käyttää Pylons-web-frameworkia, ja tietokanta mallinnetaan tietokantatyökalun SQLAlchemy avulla. Gemilo Socialin frontend-toiminnallisuus pohjautuu suurimmilta osin jQuery 1.6.1 -versioon ja jQuery UI 1.8.17 -versioon. Mukana on kuitenkin erinäisiä yksittäisiä kirjastoja ja laajennuksia jQueryyn, kuten flot.js, jolla mallinnetaan graafeja jopa vanhemmalla Internet Explorer 6 -selaimella.

3.2 Gemilo Socialin luokkarakenne

Tässä alaluvussa käsitellään Gemilo Socialin luokkarakennetta niiltä osin, mitä opinnäytetyön käytännön osuus edellyttää. Alaluvussa tarkastellaan luokkarakennetta käytettyjen sisältöjen *Exam* ja *User* näkökulmasta. Kuva 10 havainnollistaa sisältöjen periytymisjärjestystä.



KUVA 10. Sisältöjen *Exam* ja *User* periytyminen yksinkertaistettuna

Gemilo Socialissa kaikki sisällöt periytyvät *SocialObject*-sisällöstä, joka toimii pohjaluokkana ja abstraktiona periytyville luokille. *SocialObject*-luokka tarjoaa tietokantakyselyiden abstraktiot, eikä oliota itsessään ole mallinnettu tietokantaa varten. *SocialObject* periyttää alemmille luokille myös tietokantamallinnuksen. *SocialObjectin* pohjaluokkana on *object*.

ContentObject on sisältö, joka useassa tapauksessa edellyttää oman kontrollerinsa. *ContentObject*-sisältöjä Gemilo Socialissa ovat muun muassa kurssi, tapahtuma, blogi, blogikirjoitus, oppimistehtävä, koe, materiaali, organisaatio, sivu, projekti ja niin edelleen. Käytännössä kaikki isot kokonaisuudet Gemilo Socialissa periytyvät *ContentObject*-sisällöstä. Myös henkilö ja lopulta käyttäjä periytyy *ContentObject*-sisällöstä kaavion 10 mukaisesti. *ContentObject*-sisältö tarjoaa metodit oikeuksientarkastusta varten sekä tietokantamallinnuksen ja -rajapintakutsut. Lisäksi *ContentObject*-luokasta periytettyihin sisältöihin voidaan aina kommentoida.


```

... privileges = {
...     u'exam.create': lazy_ugettext(u'Can add exams'),
...     u'exam.delete': lazy_ugettext(u'Can delete exams'),
...     u'exam.edit': lazy_ugettext(u'Can edit exams'),
...     u'exam.list': lazy_ugettext(u'Can list all exams'),
...     u'exam.search': lazy_ugettext(u'Can search exams'),
...     u'exam.view': lazy_ugettext(u'Can view exams'),
... }

```

KUVA 11. Oikeuksienmäärittely sisällön *Exam* sanakirja-attribuutiksi

CustomContent-sisältöä käyttävät hyväksi sisällöt *Exam* ja *Poll*. *CustomContent*-sisältö hoitaa tietokantamallinnuksen näille periytyville luokille. Viimein *Exam*-sisällössä määritellään tietyt oikeudet sanakirjaan kuvan 11 mukaisesti.

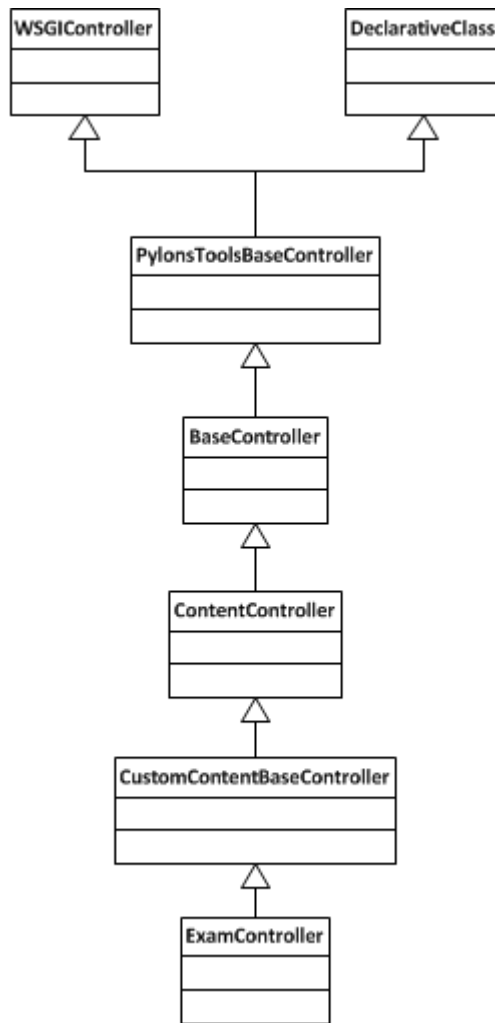
```

... def index(self):
...     self.require_auth(c.current_user.has_privilege('exam.list'))
...     c.exams = Exam.get_all(c.current_user)
...     return render('/exam/index.mak')
...
... @rest.dispatch_on(POST='_create')
... def new(self, id=None):
...     self.require_auth(c.current_user.has_privilege('exam.create'))
...     return render_form('/exam/new.mak')
...
... def edit(self, id):
...     id = h.safe_int(id) or abort(404)
...     c.exam = Exam.get(id=id) or abort(404)
...
...     self.require_auth(c.current_user.has_privilege('exam.edit'))
...     return render_form('/exam/edit.mak')

```

KUVA 12. Oikeuksientarkastus Gemilo Socialissa

Kuvassa 13 mallinnetaan periytymispuu *Exam*-kontrollerille. Opinnäytetyön kannalta olennaisin kontrolleriluokka on *Exam*. Navigointi Digipaperissa tapahtuu *Exam*-kontrollerin kautta. Uudelle kokeelle, kokeen ominaisuuksien muokkaukselle ja tarkastelulle ovat omat action-metodinsa *new*, *edit* ja *show*. *Show*-metodille voidaan passata *subsection*-avainsanaparametrina haluttu alanäkymä kokeen sisällä. Varsinaista Digipaperin sisältöä voidaan muokata passaamalla *subsection*-parametrille merkkijonoarvo *edit*. Muita mahdollisia merkkijonoarvoja ovat *returned* ja *results*, jotka tulevaisuudessa palauttavat palautettujen kokeiden näkymän ja tulospäkymän opettajalle tai oppijalle käyttäjän roolista riippuen.



KUVA 13. Kontrollerin Exam periytyminen yksinkertaistettuna

Oikeuksia tarkastellaan *PylonsToolsBaseController*-kontrollerissa määritetyllä *require_auth*-metodilla, jolle passataan senhetkisen käyttäjän oikeustarkistemetodi *has_privilege*. Kyseinen oikeustarkistemetodi on määritelty sisältöluokassa *User* samaan tapaan kuin kuvassa 11. Kuvassa 12 esitellään *Exam*-kontrollerin kolme metodia, joissa käyttäjältä tarkistetaan käyttötapauskohtaisesti oikeus resurssiin. *Privileges*-sanakirjan avain passataan senhetkisen käyttäjäolion metodille *has_privilege* parametrina. Jos käyttäjällä ei ole oikeutta resurssiin, ohjataan käyttäjä takaisin etusivulle. Tällainen oikeuksientarkastusmenetelmä on vakinaistunut käytänne Gemilo Socialissa.

3.3 HTML-näkymien modularisointi Mako-templaateilla

Tämän opinnäytetyön kehittämistehtävänä on ollut käyttöliittymän eri näkymien koostaminen tässä aluvussa esiteltävien yleisten asiakasvaatimusten mukaisesti.

Tässä opinnäytetyössä ei ole keskitytty backend-logiikan implementointiin. Opinnäytetyön tuloksena tavoiteltiin ensimmäisen version Digipaperi-palvelua, jonka avulla opettajat kykenevät luomaan luettavaa oppimateriaalia ja koeharjoituksia opiskelijoille ja jonka avulla oppijat pystyvät lukemaan ja vastaamaan digitaaliselle paperille sisällytettyyn tekstiin ja koetehtäviin. Digipaperi-palvelun tavoitteena on mahdollistaa tekstin, kuvan, upotetun median sekä kokeiden paketoiminen yhden kokonaisuuden alle. Edellä mainittuja osakokonaisuuksia kutsutaan tässä alaluvussa moduuleiksi.

Digipaperi tulee osaksi Gemilo Socialia ja pyrkii olemaan vaihtoehtoinen tapa järjestää testejä, kun ensisijainen tapa on yleensä ollut tulostettava paperinippu. Digipaperin muokkaus- ja katselunäkymä on rakennettu verkkosivupohjaiseksi esitykseksi, jossa hyödynnetään JavaScript-ohjelmointikielen kirjastoja jQuery ja jQuery UI sekä Gemilo Socialiin valmiiksi tehtyjä komponentteja. Näistä osa hyödyntää edellä mainittuja JavaScript-kirjastoja ja toinen osa on kokoelma erilaisia yleiskäyttöisiä Gemilo Socialissa käytettäviä osanäkymiä, kuten napit, sivurunko tai jäsenkutsumisblokki. Alustavalla Digipaperi-palvelulla tulisi pystyä toteuttamaan seuraavat yleiset asiakasvaatimukset:

1. Lisäämään uusia moduuleja haluamalleen digipaperille.
2. Järjestelemään moduuleja digipaperin sisällä.
3. Muokkaamaan moduulien sisältöä ja ominaisuuksia, jollaisia saattavat olla kysymyksen tai kokeen otsikko tai pistemäärä mutta myös moduulin suhteellinen leveys digipaperin sisällä.
4. Esikatselmaan koostettua digipaperia tai -papereita.
5. Arvioimaan oppijan antamat vastaukset.
6. Tarkastelemaan tuloksia niin opettajana kuin oppijana niin, että kokeen toteutukseen tai arviointiin osallistuneet opettajat näytetään.

Opinnäytetyön käytännön työn aspektina on ollut modularisoida ja keskittää osakokonaisuuksia omiin renderöintifunktioihinsa. Flanaganin (2011, 246) mukaan tarpeeksi geneerinen moduuli voidaan käyttää uudelleen toisessa projektissa. Gemilo Socialin lähdekoodissa on pyritty rakentamaan rutiineja mahdollisimman yleiskäyttöisiksi. Alustavassa opinnäytetyössä renderöintifunktioiden toteutuksessa ei tavoitella geneerisyyttä, vaan toteutuksessa tarkastellaan Digipaperi-implemентаatiota

funktion ulkoisen kuvaavuuden ja rajapinnan kautta. Eristämällä eri osakokonaisuudet omiin renderöintifunktioihinsa mahdollistetaan niiden muuntaminen yleiskäyttöisemmiksi tulevaisuudessa pienemmällä vaivalla. Vaikka työssä korostetaankin koodin merkitystä, tarkastellaan toteutusta myös näkymien kautta, kuten tullaan näkemään seuraavassa luvussa.

4 TOTEUTUS

Tässä luvussa käsitellään opinnäytetyön kehitystyön toteutusta. Alaluvuissa käsitellään muun muassa eri näkymien toteutusta ja käytettyjä ratkaisuja. Esimerkitapauksen kautta käydään läpi yhden näkymän edellyttämä templaattiriippuvuusketju.

4.1 Templaattihierarkia

Gemilo Socialin näkymät renderöidään koostaen useasta templaattitiedostosta, jotka on kategorisoitu eri kansioden alle käyttötapauksen mukaan. Templaatit sisällytetään joko `<%inherit>`-, `<%include>`- tai `<%namespace>`-tagin avulla. Tässä alaluvussa käsitellään Gemilo Socialin templaattihierarkiaa pääasiassa liitteen 1 esimerkin kautta. Kyseisessä liitteessä on mallinnettu koekokonaisuuden alanäkymän templaattirakenne, ja sitä kutsutaan tässä luvussa esimerkkikuvaksi.

Tiedostopolkujen nimet esimerkkikuvassa noudattavat Unix-tyyppisistä järjestelmistä tuttua tiedostopolkusyntaksia. Pisteellä alkavat polut ovat suhteessa hierarkisesti korkeamman templaattitiedoston nykyiseen hakemistoon ja vinoviivalla alkava polku viittaa *mod*-kansioon. Riippuvuusnuolen kohde on *include*- tai *namespace*-invokaation suorittava templaatti ja on näin siis hierarkisesti korkeampi templaatti suhteessa kutsuttuun templaattiin. Esimerkkikuvassa värit indikoivat templaattien erilaiset roolit: musta merkitsee alustavasti kutsuttavaa templaattia, sininen symboloi periyttävää tai periytettyä templaattia, harmaa templaattia jonka sisältö luetaan kokonaisuudessaan hierarkisesti korkeampaan templaattiin `<%include>`-tagin avulla sekä valkoinen templaattia, josta tuodaan ja kutsutaan `<%namespace>`-tagin avulla vain kuvassa merkatut funktiot. Viimeistä ja ensimmäistä templaattiroolia lukuun ottamatta kaikki kunkin templaatin määritellyt funktiot on listattu esimerkkikuvassa. Liitteen 1 oranssitaustaiset templaatit liittyvät muiden näkymien renderöintiin, eikä niiden

sisältöä käsitellä tässä alaluvussa. Viimeiseksi punaisella reunusviivalla esitetyt templaattit ovat tämän opinnäytetyön tuloksena syntyneitä uusia templaatteja.

Kun kontrolleri *Exam* saa komennon renderöidä show-templaatin, tarkastetaan kontekstimuuttujan *c.subsection* avulla, mikä neljästä näkymästä sisällytetään kuvan 14 mukaisesti. *Include*-tagilla kutsuttu *view_or_edit*-templaatti passaa välittömästi kontrollin *two_columns*-templaatile, joka puolestaan passaa kontrollin välittömästi *header_and_footer*-templaatile. Huomataan, että *header_and_footer*-templaatti tekee suurimman pohjatyön ja toimiikin eräänlaisena pohjana välissä olevalle *two_columns*-templaatile, joka puolestaan määrittelee sivun ulkoasupohjan. Työssäni olen määritellyt tälle kaksikolumniselle ulkoasupohjalle tyhjän *sidebar*-metodin, joka ylikuormitetaan templaateissa *view_or_edit*, *returned* ja *results*.

Joissain esimerkkikuvan templaateissa hakemistopolku voi alkaa hakemistolla *responsive* tai *default*. Gemilo Socialissa sisällytettävä templaattitiedosto sisällytetään ensisijaisesti toteutettavan *mod*-hakemiston alta. Jos samanniminen templaattitiedosto on olemassa niin *default*- kuin *responsive*-modin alla, sisällytetään siis vain *responsive*-modin alla oleva templaatti. Muussa tapauksessa sisällytetään vastaavan polun *default*-modin templaatti.

```

## -*- coding: utf-8 -*-

<%def name="show_exam(exam, assignment_types)">
... <%include file="view_or_edit.mak"/>
</%def>

<%def name="edit_exam(exam, assignment_types)">
... <%include file="view_or_edit.mak"/>
</%def>

<%def name="returned_exams(exam, assignment_types)">
... <%include file="returned.mak"/>
</%def>

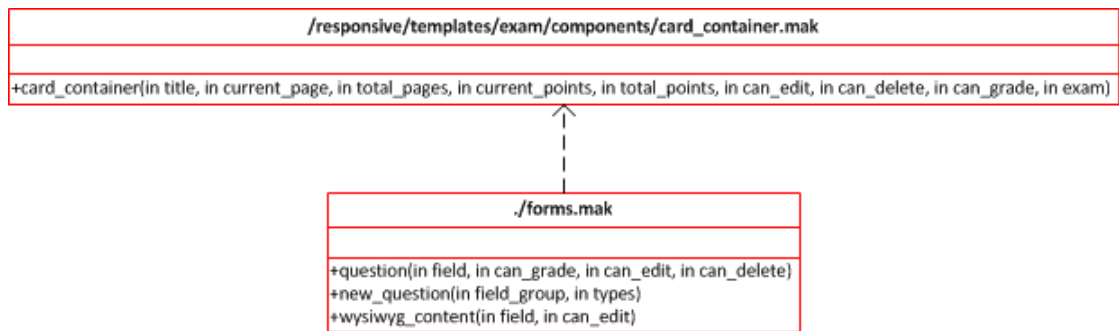
<%def name="results_exam(exam, assignment_types)">
... <%include file="results.mak"/>
</%def>

%if not c.subsection:
... ${show_exam(c.exam, c.assignment_types)}
%elif c.subsection == 'edit':
... ${edit_exam(c.exam, c.assignment_types)}
%elif c.subsection == 'returned':
... ${returned_exams(c.exam, c.assignment_types)}
%elif c.subsection == 'results':
... ${results_exam(c.exam, c.assignment_types)}
%endif

```

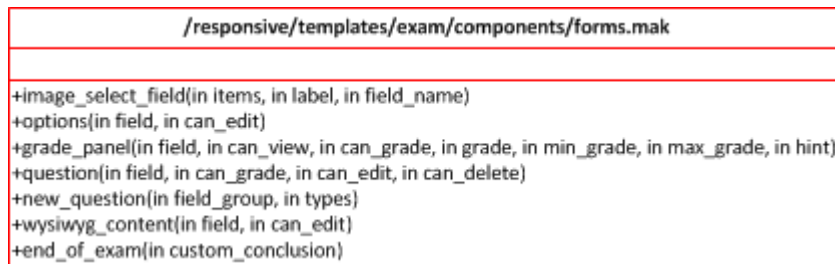
KUVA 14. Mako-templaatin show sisältö

Monessa uudessa templaatissa funktioiden parametrit sisältävät sellaisia yhteisiä parametrimuuttujia kuten *exam*, *can_view*, *can_edit_object_access*, *can_edit* ja *can_delete*. Perustelut *can*-alkuisille totuusarvoisille parametrimuuttujille on se, että näin kutsuttaviin funktioihin määritellään vain kerran funktion kutsumisen yhteydessä menetelmä, jolla katselu, muokkaus, käyttöoikeudet tai poisto määritellään. Funktio jätetään myös monikäyttöisemmäksi silloin, kun se ei ota kantaa oikeuksientarkastuksen implementointiin. Parametrimuuttujan *exam* tarkoituksena on passata tarkastettava resurssi sitä käyttäville funktioille. Sen passaamisen syyt ovat samat kuin *can*-alkuisissa tarkistemuuttujissa.



KUVA 15. Mako-templaatin `card_container` templaattiriippuvuus

Huomataan, että uusissa templaateissa sisältöä ei tuoda `<%include>`-tagin avulla. Esimerkiksi templaatti `card_container` tuo `<%namespace>`-tagin avulla forms-templaatin renderöintifunktiot `question`, `new_question` ja `wysiwyg_content` kuvassa 15 esitetyllä tavalla.



KUVA 16. Mako-templaatin `forms` sisältämät funktiot

Templaatin `forms` avulla toteutetaan tehtävien esitys, tehtävien arvioimisrajapinta, monivalintavaihtoehtojen esitys, uuden tehtävän esitys, kuvamonivalinnan esitys, WYSIWYG-tekstikentän esitys sekä kokeen päättymisnäkyvä kuvan 16 esitetyillä funktioilla ja niiden parametreilla. Joissain instansseissa funktioille passataan `field_group`- ja `field`-parametreja, joista ensimmäinen on lista `CustomFieldGroup`-luokan olioita ja jälkimmäinen on `CustomField`-luokan olio. `CustomField`-luokan instanssiolioissa säilötään tietoa esimerkiksi yksittäisistä tehtävistä, kun taas `CustomFieldGroup`-luokan instanssiolio voi edustaa yhtä digipaperia ja sen sisältöä.

4.2 Näkymät

Jotta kokeita voi suorittaa, erilaisia näkymätiloja vaaditaan luomista, muokkausta, esikatselua sekä suorittamista varten. Tämän lisäksi kokeeseen vaaditaan kokeen jälkeisiä tarkastelunäkymiä opettajaa ja opiskelijaa varten. Tarkalleen ottaen käyttötapaukset ovat seuraavat:

- Uusi koe
- Kokeen muokkaustila
- Kokeen muokkaustila lukitussa moodissa
- Kokeen suoritus
- Kokeen arviointi
- Palautettujen kokeiden näkymä
- Oppijan näkymä tuloksista
- Opettajan näkymä.

Kuvasta 17 nähdään, milloin yhteisesti käytettävien komponenttien tai osanäkymien eri toiminnallisuudet tai elementit näytetään eri käyttötapauksissa. Osat on jaettu suurempiin kokonaisuuksiin: alanavigaatio-, sivunavigaatio-, sivu- sekä korttikomponenttiin. Alanavigaatio viittaa tässä tapauksessa kokeen sisällä olevaan navigointipaneeliin, jossa kokeeseen tai kyseiseen näkymään liittyvät toiminnallisuudet sijaitsevat. Sivunavigaatio käsittää kokeeseen itseensä liittyvää tietoa tai oikeuksienhallintatoiminnallisuutta. Lisäksi sivunavigaatiossa voidaan esittää listamaista tietoa kokeesta, joka voi olla lista kaikista kokeen sivuista tai lista kaikista kokeeseen osallistuvista henkilöistä. Sivukomponentti on kokeeseen liittyvä yhdesti tai useammin esitettävä komponentti, joka sisältää tehtäviä tai HTML-sisältöä korttikomponenttien kautta esitettynä. Vastaisuudessa sivukomponentin sisällä olevia tehtäviä tai HTML-sisältöjä viitataan korteiksi.


```
/*Changing main-content visually into a sidebar and vice-versa*/  
.controller-exam.action-show.main-content,  
.controller-exam.action-show.subsection-returned.main-content,  
.controller-exam.action-show.subsection-edit.main-content,  
.controller-exam.action-show.subsection-results.main-content {  
  width:15%;  
}  
  
.controller-exam.action-show.secondary-content,  
.controller-exam.action-show.subsection-returned.secondary-content,  
.controller-exam.action-show.subsection-edit.secondary-content,  
.controller-exam.action-show.subsection-results.secondary-content {  
  width:77%;  
}
```

KUVA 18. Kontrolleripohjainen CSS-tyylin korvaaminen kehitystyössä

Vaikka CSS-valitsimien tarkkuus täytyykin joskus vastata tai olla aikaisemmin määritellyn valitsimen tarkkuutta korkeampi, ei tarkka valitsin ole toivottavaa kehittäjien kannalta. Työssä on näin ollen pyritty välttämään tarkkoja CSS-valitsimia silloin, kun on haluttu luoda uudelleenkäytettäviä komponentteja. Kuvassa 19 havainnollistetaan valitsinosien vähäisyyttä, jotta tyylien korvaaminen olisi tulevaisuudessa mahdollista käyttämättä *!important*-deklaraatiota tai kuluttamatta paljon aikaa monimutkaisen valitsimen luomiseen.

```
.grade-panel · ...  
.grade-panel · .message-wrapper · ...  
.action-edit · .grade-panel · ...  
.score-container · ...  
.score-container · * · ...  
.score-container · select · ...  
.grade-panel · .score-container · span · ...  
.header-panel · ...  
.header-panel-sub · ...  
.header-panel · img · ...  
.option-container · ...  
.poll-unit · .title · ...  
.poll-unit · .points · ...  
.poll-unit · textarea · ...
```

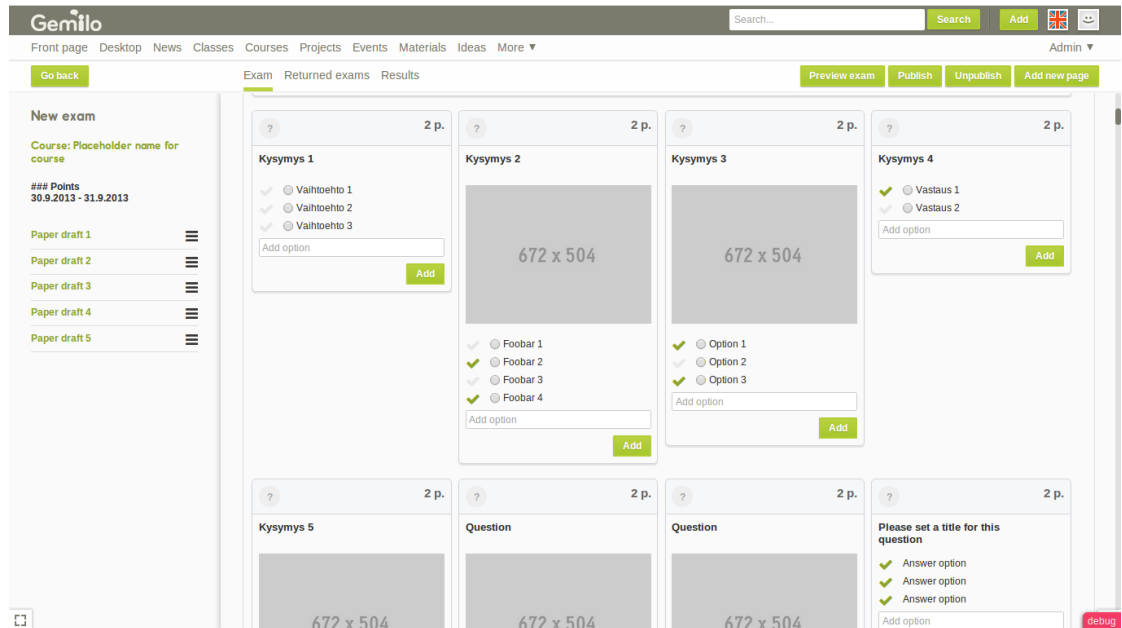
KUVA 19. CSS-valitsimien käyttö kehitystyössä

Olemme käyneet läpi yleiskatsausta siitä, mitä kaikkea näkymät vaativat ja minkälaisia yleisiä teknisiä periaatteita olen soveltanut työssäni. Seuraavaksi tarkastelemme kutakin näkymää niin visuaalisesti kuin sen puolesta, miten näkymät on teknisesti ratkaistu.

4.2.1 Kokeen muokkaus

Kokeen muokkaustila on oletuksellisesti ensimmäinen näkymä heti kokeen luomisen jälkeen. Oletamme, että opettajan pääasiallinen tavoite koetta tarkastellessa on sen

muokkaus. Muokkaustilassa esitetään muokattu kaksijakoinen näkymä, minkä tyylimäärityä demonstroititiin kuvassa 18.



KUVA 20. Koe muokkaustilassa

Kuvassa 20 huomataan, että alanavigaatio sisältää kyseisen kokeen sisällä olevia linkkejä opettajalle. Myös kokeen muokkaukseen liittyvät toiminnallisuudet on esitetty alanavigaatiossa oikealla. Opettaja voi merkitä kortin kysymyksen vastausvaihtoehdot oikeaksi painamalla harmaita oikein-symboleja tai toisinpäin painamalla niitä vihreinä.

```


$$\text{\$}\{\text{sub\_nav}(c.\text{exam}, \text{\cdot subsection}=c.\text{subsection}, \text{\cdot can\_edit}=\text{True}, \text{\cdot can\_delete}=\text{True})\}$$


```

KUVA 21. Alanavigaation esittävän funktion kutsumisesimerkki

Alanavigaatiofunktiota kutsutaan kuvan 21 esimerkillä passaamalla argumentteina globaalin kontekstimuuttujan avulla koe *c.exam* ja koenäkömän senhetkinen sijainti *subsection*-avainsanaparametrina. Avainsanoitetut parametrit *can_edit* ja *can_delete* kertovat eristetulle funktiolle oikeuksista näyttää. On huomautettava, että tautologiset arvot kuvassa 21 tulevat vaihtumaan tuotantoversioon mennessä bisnesloogisiksi tarkisteiksi.

The screenshot shows the 'New exam' interface in Gemilo. On the left, there's a sidebar with 'New exam' details: Course (Placeholder name for course), Points (30.9.2013 - 31.9.2013), and a list of paper drafts (1-5). The main area is titled 'Unnamed paper' and shows '1 / 3' cards and '1 / 3 points (total)'. Under 'Choose type', there are icons for HTML content, Text answer, Multiple choice (two options), and Essay. Below this are input fields for Title, Image (optional), and Help text (optional). A green 'Add new question' button is visible. A green arrow points to the bottom right of the main area, where a message says 'Add questions and content on your paper by adding a new card from above.'

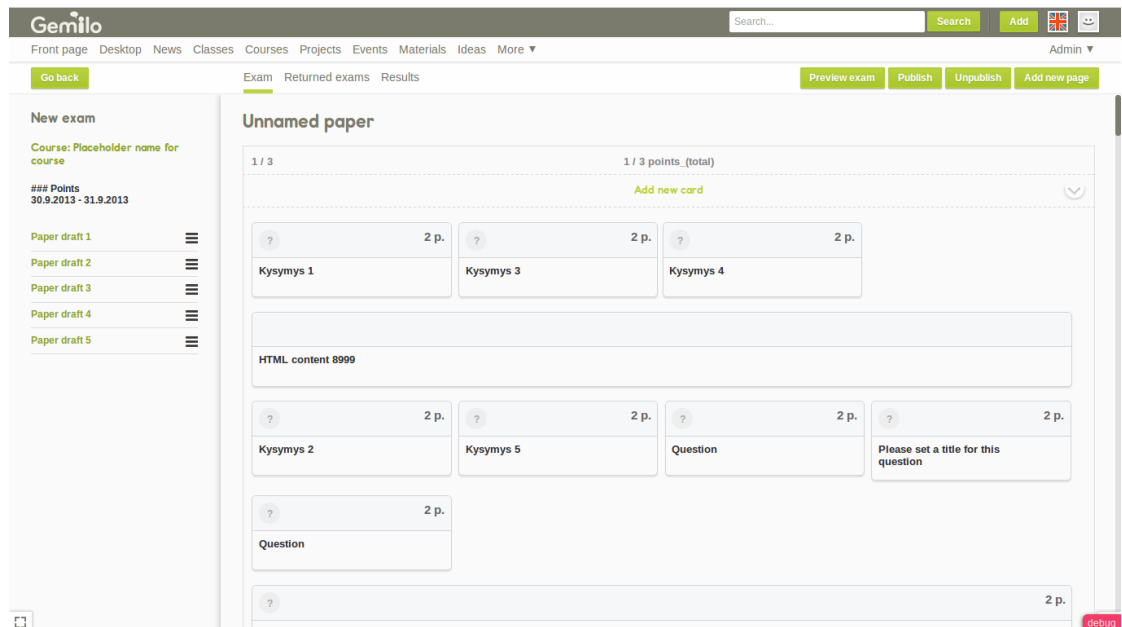
KUVA 22. Uuden sisällön lisääminen

Kortit esitetään digipapereissa, joihin opettaja voi lisätä uusia kortteja kuvan 22 mukaisesti digipaperin yläosasta. Kortin sisältö voi olla kysymys, johon opiskelija voi vastata monivalintatehtävällä, tekstivastauksella tai esseellä, tai HTML-sisältö. Digipapereiden esiintymisjärjestystä voidaan muokata sivunavigaatiosta JQuery UI -sortable-widgetin avulla. Lisäksi korttiin voidaan halutessa sisällyttää yksi kuva. HTML-sisällön hallinta perustuu kaupalliseen WYSIWYG-editoriin Redactoriin. Sen kautta lisättävien kuvien määrää ei ole rajoitettu.

The screenshot shows the 'General settings' dialog box for a question card. The dialog has a 'Title' field with the value 'Kysymys 3' and a 'Points' field with the value '10'. Below these is a 'Choose column width' section with four radio button options: '1/4 column', '2/4 column', '3/4 column', and '4/4 column'. The '1/4 column' option is selected. There is also a 'Required field' checkbox which is checked. At the bottom of the dialog is a green 'Save' button. The background shows the same 'New exam' interface as in the previous screenshot, but it is dimmed.

KUVA 23. Korttielementin asetukset

Korttien asetuksista mahdollistetaan otsikon, tehtävän maksimipistemäärän muuttaminen sekä tehtävän määrittely pakolliseksi. Kuvan 23 mallisesti opettaja voi lisäksi muokata kortin leveyttä neljäkolumnisella ruudukolla suhteessa paperin leveyteen. Kuvassa 23 ei näy mahdollisuus lisätä kuvaa tai vaihtaa olemassaolevaa kuvaa toiseen. Tämä tiedostettu virhe johtuu kehitysjajan loppumisesta.



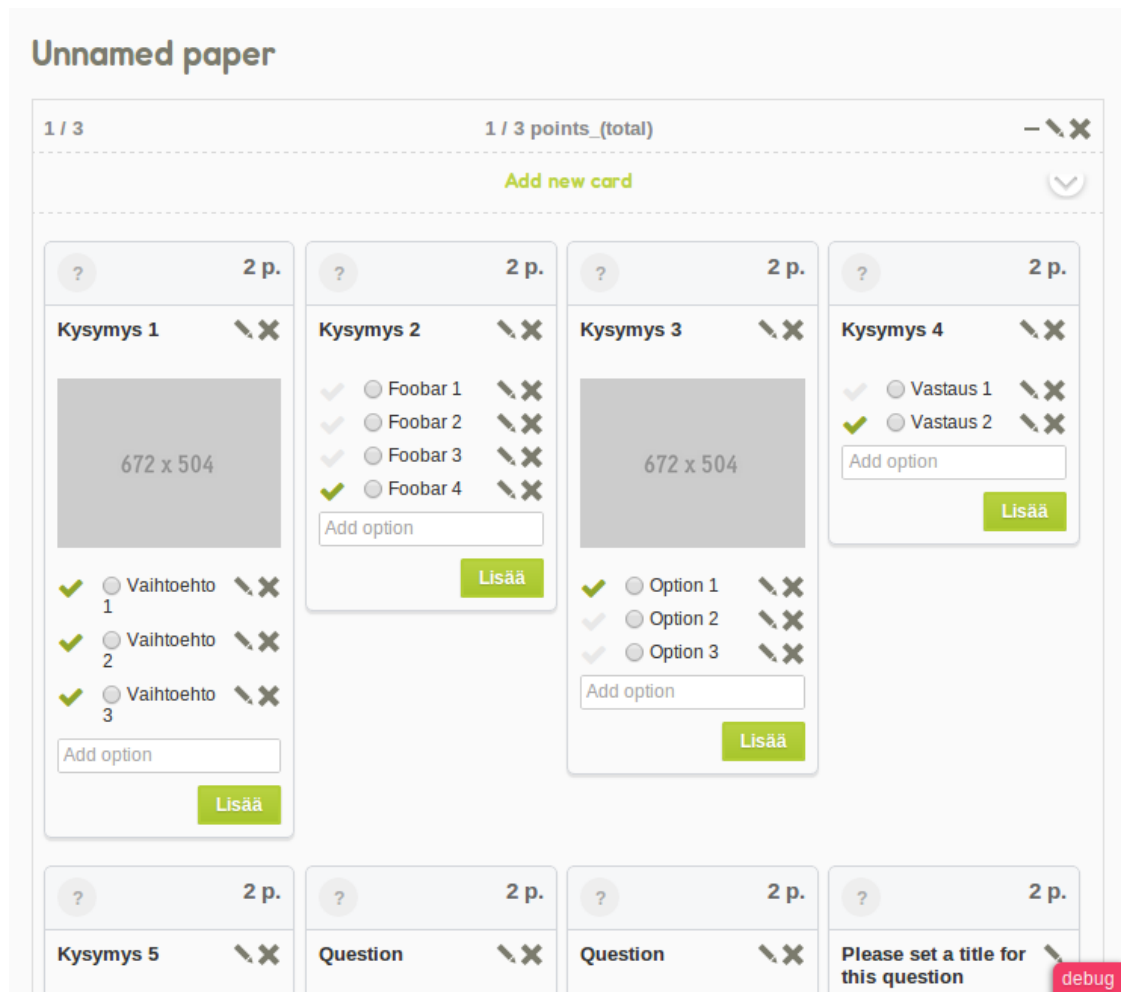
KUVA 24. Digipaperin sisällä olevat kortit pienennettyinä

Myös kortteja voi uudelleenjärjestellä JQuery UI -sortable-widgetin avulla. Ongelmaksi muodostui kehitysvaiheessa kuitenkin se, että kortit voivat olla sisällöltään hyvinkin isoja kortteja. Isojen korttien uudelleenjärjestely on pienellä näyttötilalla hyvin epäkäytännöllistä. Ratkaisuksi kehitettiin digipaperiin toiminto, jonka avulla kortit saadaan pienennettyä. Uudelleenjärjestely pienennetyillä korteilla sujuu kuvan 24 mukaisesti.

```
function targeted_toggle_hide()·{  
····$(".minimize").apply_once().click(function·()·{  
······var minimize_targets·=·$(this).data('minimize')·||·false;  
  
······if·(!minimize_targets)·return·false;  
  
······if·(!$(minimize_targets).hasClass('hidden'))·{  
········$(minimize_targets).addClass('hidden');  
······}·else·{  
········$(minimize_targets).removeClass('hidden');  
······}  
  
······return·false;  
····});  
}
```

KUVA 25. Korttien pienennyksen suorittava tapahtumapohjainen funktio kehitystyössä

Pienennys hoidetaan HTML-elementtien attribuuttien avulla kunnioittaen Gemilo Socialin deklarativisuusperinnettä: kuvassa 25 näytetään logiikka, joka kiinnittää *minimize*-luokilla julistettuihin elementteihin hiirenpainiketapahtuman. Toiminto olettaa, että kyseinen elementti sisältää myös *data-minimize*-attribuutin, joka sisältää selektorin, joihin kohdistaa piilottaminen. Koska myös *main.js*-tiedosto, jossa kuvan 25 funktio on määritelty, on käytössä kaikissa Gemilo Socialin modeissa, voi funktiolla olla yleisempääkin käyttöä.



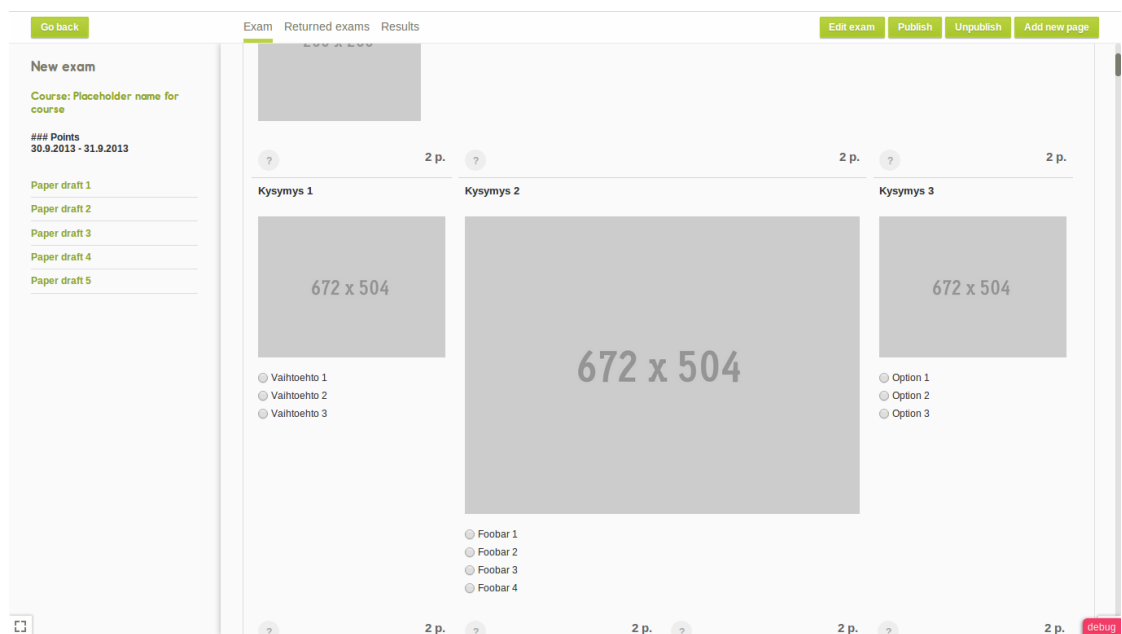
KUVA 26. Kokeen kysymysten muokkausnäkö, kun selainikkuna on leveydeltään alhainen

Aikaisemmista kuvista ei käy ilmi, kuinka tehtävien vastausvaihtoehtoja tai vastauksia pystyy muokkaamaan. Kutakin korttia on kuitenkin mahdollista muokata ja poistaa. Kuvassa 26 muokkaus- ja poistotoiminnot ovat näkyvissä niin kortille kuin vastausvaihtoehdoille. Normaalisti toiminnot tulevat esille, kun hiiriosoitin vietään kortin tai vastausvaihtoehdon päälle. Ennen tiettyä resoluutioleveyttä oletetaan, että päättekoneena voi olla tablettitietokone tai mobiilipäätelaite. Ongelmalliseksi tästä ratkaisusta tekee sen, että joissain nykyisissä tablettitietokoneissa resoluutio voi vastata tavanomaisen tietokoneen, kannettavan tai pöytätietokoneen, resoluutiota (Ebay 2013).

4.2.2 Kokeen esikatselu

Kokeen esikatselu on opettajalle ja oppijalle samannäköinen, paitsi että alanavigaation toiminnallisuudet ovat oppijalta piilotettu. Itse asiassa oppijan näkökulmasta esikatselu on kokeen suoritusta.

Kuvasta 27 käy ilmi, että sivujen ja korttien uudelleenjärjestely on poistettu esikatselusta. Korttimainen syvyyttä korostava ilme on hävitetty ja pelkistetty tasaiseksi siinä pelossa, että kortin muokkaustilan ilme veisi pois huomiota tehtävistä ja opettajan kirjoittamasta sisällöstä.



KUVA 27. Kokeen ennakkokatselu

Alustavassa versiossa kaikki digipaperit esitettäneen yhdessä näkymässä. Sivunavigaatiossa olevalla ankkurilinkitetyllä paperilistalla voidaan nopeasti siirtyä halutun paperin täyttämiseen tai lukemiseen. Näkymiä varten on kuitenkin suunniteltu liikkumista helpottavat ohjausnapit, jos päätettäisiin näyttää vain yksi digipaperi per näkymä. Kokeen päätteeksi oppijalle esitetään kokeen päättymisnäkymä ja tallennusnappi, jota painettua oppija lähetetään takaisin kurssinäkymän kokeet-alavalikkoon eli ulos koetilanteesta.

4.2.3 Kokeiden arviointi

Arviointinäkylässä opettaja valitsee sivunavigaatiosta halutun oppijan kokeen. Toiminnolla opettaja valitsee näytettäväksi haluamansa oppijan kokeen yhdelle näkymälle.

KUVA 28. Esseevastauksen arviointi

Tehtävistä, jotka ovat arvioitavissa diskreettisten vastausten perusteella, esitetään tehtäväkortin oikeassa ylänurkassa pistesuoritus. Esseetehtävän tai muunkaltaisen kvalitatiivisen tehtävän arviointi on haasteellinen ongelma, joten tehtäväkortin oikeassa yläkulmassa vaihteleva pistemäärä esitetään alavetovalikkona, sen oikealla puolella kokonaispistemäärä. Kuten kuvassa 28 näkyy, dynaaminen pistelaatikko on huomaamaton suhteessa muuhun visuaaliseen kontekstiin. Näin isommaksi huomion keskipisteeksi jää itse tehtävä ja sen vastaus.

4.2.4 Tulosten tarkastelu opettajan näkökulmasta

Tulosten tarkastelusivulla opettaja näkee taulukosta seuraavia ominaisuuksia kunkin oppijan tilanteesta kokeen suhteen:

- oppijan nimen
- kokeentekostatuksen
- kokeenlähetysajan
- arvioimattomien tehtävien määrän

- senhetkisen pistemäärän
- kokeen maksimipistemäärän
- senhetkisen pistemäärän suhde kokonaispistemäärään prosenttiyksikköinä.

NIMI	EXAM STATUS	TIME SUBMITTED	UNGRADED QUESTIONS	CURRENT POINTS	MAX POINTS	TOTAL PERCENTAGE
Student student	Palalettu	14:49	4	87	100	87.00 %
Student student	Arvosteltu	14:49	10	87	100	87.00 %
Student student	Arvosteltu	1.10.2013 14:49	10	87	100	87.00 %
Student student	Luonnos	14:49	2	87	100	87.00 %
Student student	Arvosteltu	14:49	9	87	100	87.00 %
Student student	Luonnos	14:49	3	87	100	87.00 %
Student student	Palalettu	14:49	2	87	100	87.00 %
Student student	Arvosteltu	1.10.2013 14:49	2	87	100	87.00 %
Student student	Palalettu	1.10.2013 14:49	8	87	100	87.00 %
Student student	Arvosteltu	14:49	4	87	100	87.00 %

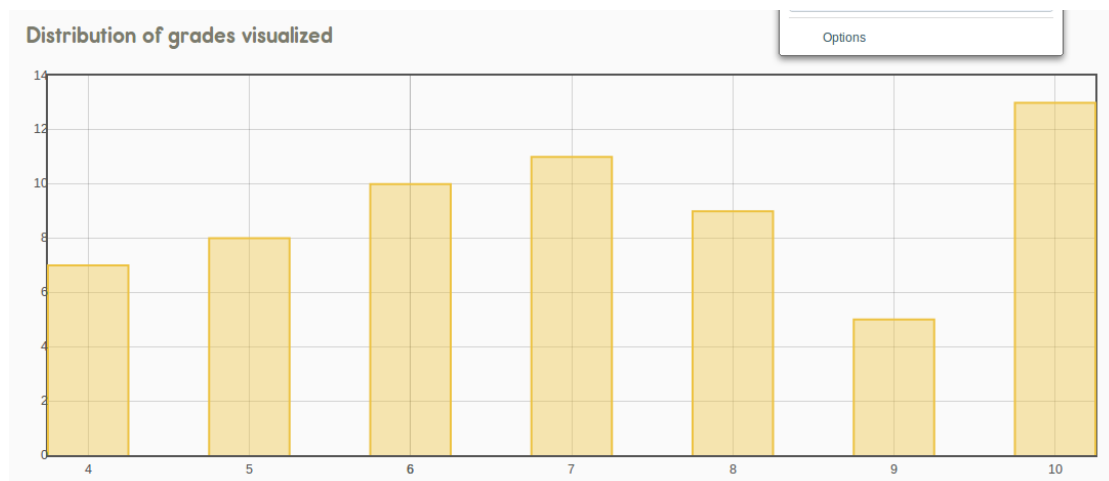
KUVA 29. Kokeen tulosten näyttäminen opettajalle

Visuaalisesti taulukon suunnittelussa on pyritty välttämään elementtejä, jotka toisivat taulukon irti taustaelementistä. Kuvassa 29 demonstroidaan tulosten katselua.

```
## - * - coding: utf-8 - * -
<?namespace file="/group/components/member_box.mak" import="member_box" />
... ${member_box(c.current_user, exam.get_role_group(u'teacher', lazy=True))}
...
```

KUVA 30. Osallistujaosanäkymän ja -toiminnallisuuden kutsuminen

Sivunavigaatiossa on tuotu esille ne opettajat, jotka osallistuvat kokeen arvioimiseen. Kyseinen osallistujatoimintokokonaisuus on Gemilo Socialin yleinen jäsenkomponentti *member_box*, joka pystytään kutsutaan sijoittamaan templaattiin kutsumalla *member_box*-funktioita senhetkisellä käyttäjällä sekä kokeen *role_group* parametreina kuvan 30 mukaisesti. Opettajat näkyvät niin opettajalle kuin oppijalle.



KUVA 31. Pylväsgraafi arvosanojen jakauman esittämistä varten

Tulosten tarkastelua varten haluttiin myös jonkinlaista esitystä arvosanojen jakaumasta. Koska haluttu tieto tulisi olemaan yksiulotteista, päädyttiin kuvan 31 mukaiseen pylväsgraafiin. Graafin toteutuksessa käytettiin flot.js-kirjastoa.

4.2.5 Tulosten tarkastelu oppijan näkökulmasta

Tulosten tarkastelunäkymällä oppija näkee kuvan 32 mukaisen taulukon. Koska koetta ja annettuja vastauksia ei esitetä oppijalle ainakaan alustavassa versiossa, on tarkastelunäkymä ainoa kokeen suorituksenjälkeinen oppijalle esitettävä näkymä. Tulostaulukossa esitetään seuraavia sivun ominaisuuksia:

- sivun nimi
- kysymysten määrä
- senhetkisen pisteiden määrä kunkin sivun kohdalla
- sivun maksimipistemäärä
- senhetkisen pistemäärä suhde kokonaispistemäärään prosenttiyksikköinä.

SIVUT	AMOUNT OF QUESTIONS	POINTS GOTTEN	MAX POINTS	TOTAL PERCENTAGE
Page 1	3	0	10	87.00 %
Page 2	6	4	10	87.00 %
Page 3	2	9	10	87.00 %
Page 4	10	6	10	87.00 %
Page 5	3	4	10	87.00 %
Page 6	3	4	10	87.00 %
Page 7	10	6	10	87.00 %
Page 8	6	5	10	87.00 %
Page 9	0	3	10	87.00 %
Page 10	8	11	10	87.00 %
Tilvisteelmä	50	96	100	22.00 %

KUVA 32. Kokeen jälkikäteistarkastelu oppijan näkökulmasta

Näkymän ensimmäisestä versiosta jäi uupumaan nopea tiivistys oppijan arvosanasta. Valmis arvosana esitettäneen vastaisuudessa korostetusti taulukon yläpuolella. Oppijan sivunavigaatio on identtinen opettajan tulostuloksen sivunavigaation kanssa.

4.3 Implementoinnin arviointi

Implementoinnin arvioimisessa oli huomattavia vaikeuksia sen suhteen, kuinka käytännön työn onnistumista voitaisiin - muutoin kuin ensisijaisen tavoitteensa lisäksi - objektiivisesti mitata. Koostetut näkymät olivat tämän opinnäytetyön kehittämistehtävänä, mutta niiden onnistumista tulisi tarkastella luotujen renderöintifunktioiden perusteella. On nimittäin todennäköistä, että ensimmäisen version komponentit suoraviivaistetaan ja refaktoroidaan lopulta muistuttamaan muita Gemilo Socialin komponentteja. Laadukkaat rutiinit, jollaisia renderöintifunktiot ovat tämän opinnäytetyön kontekstissa, edesauttavat muokattavuutta.

Edellä mainituista ongelmista huolimatta ensimmäisenä on tarkasteltu, kuinka valmiiksi näkymät on saatu koostettua suhteessa asiakasvaatimukseen. Toisena kriteerinä on katsottu, kuinka hyvin toteutuksessa on onnistuttu modularisoimaan näkymiä rutiinit-alaluvun esittämän tarkistelistan mukaisesti. Kolmanneksi näkymien käytettävyyttä on tarkasteltu tilankäytön kannalta.

Kuten edellisessä luvussa kävi ilmi, näkymissä on pyritty ottamaan huomioon kaikki aikaisemmin määritellyt asiakasvaatimukset. Ongelmakohdiksi muodostautuivat

pienemmät näkymät, joita ei ajateltu työtä tehdessä. Tällaisia osanäkymiä olivat asetuksiin liittyvät näkymät. Joitain asetuksiin liittyviä osanäkymiä on suunniteltu, kuten esimerkiksi yhteen digipaperiin liittyvät asetukset. Kuitenkin on selvää, ettei suunnitteluaikaa kaikille pienille osanäkymille annettu tarpeeksi. Esimerkiksi HTML-sisältöön liittyviä asetuksia ei juuri mietitty.

Digipaperin kvantitatiivisten vastausten tarkistaminen tullaan automatisoimaan. Ensimmäisessä versiossa ei kuitenkaan suunniteltu helpottavia näkymiä oppijan kvantitatiivisten vastausten arvioimista varten. Jos asiakasvaatimukset tarkastuksen suhteen ovat tulevaisuudessa yksinkertaiset ja vähäiset, jo luoduilla näkymillä pystytään toteuttamaan arvioimattomien tehtävien näyttäminen opettajalle. Tämä edellyttänee pientä muutostyötä luotuihin näkymiin. Jos vaatimuksia on paljon tai ne sisältävät hankalia reunaehtoja, voi toteutus edellyttää kokonaan uuden näkymän tai uusien näkymien luomista.

```

<%def name="image_select_field(items, label, field_name)"> % </%def>

<%def name="options(
..... field,
..... can_edit=False)"> % </%def>

<%def name="grade_panel(
... field=None,
... can_view=False,
... can_grade=False,
... grade=None,
... min_grade=0,
... max_grade=None,
... hint=None)"> % </%def>

<%def name="question(field, can_grade=False, can_edit=False, can_delete=False)"> % </%def>

<%def name="new_question(field_group, types)"> % </%def>

<%def name="wysiwyg_content(field, can_edit=False)"> % </%def>

<%def name="end_of_exam(custom_conclusion=None)"> % </%def>

```

KUVA 33. Mako-templaatin forms sisältämät renderointifunktiot

Rutiinit-alaluvussa on määritelty hyvän rutiinin tekijöitä. Subjektiivisesti oli vaikea määrittellä, ovatko esimerkiksi kuvan 33 rutiininimet selkeitä ja abstrakteja. Esimerkiksi rutiininimi *grade_panel*, joka viittasi digipaperin sisällä olevan kortin yläosaan, kuvasti melko tarkan käyttötapauksen. Välttämättä kortti, jota on edustettu rutiinilla *question*, ei tulevaisuudessa esiinny samalla nimellä. Sama pätee tietenkin rutiinille *new_question*. Rutiinin *end_of_exam* nimi ei ottanut huomioon generistä tilannetta, jossa koostettu kysely voisi olla jotain muutakin kuin koe. Kyseisen rutiinin

parametri *custom_conclusion* oli myös melko tarkka kuvaus muuttujalle, vaikka tosiasiaassa parametrin nimeksi olisi periaatteessa voinut riittää muuttujanimi *content*.

```

<%def name="sub_navigation()">
...<namespace file="components/sub_navigation.mak" import="sub_nav" />
...S{sub_nav(c.exam)}
</%def>

<div class="meta-content-wrapper">
...<namespace file="components/side_column_returned_exam.mak" import="side_view" />
...S{side_view(c.exam)}
</div>

<%def name="sidebar()">
...<namespace file="components/card_container.mak" import="card_container" />
...% for i in xrange(0,10):
...    S{card_container(exam=c.exam, can_grade=True, can_edit=False, can_delete=False)}
...% endfor
</%def>

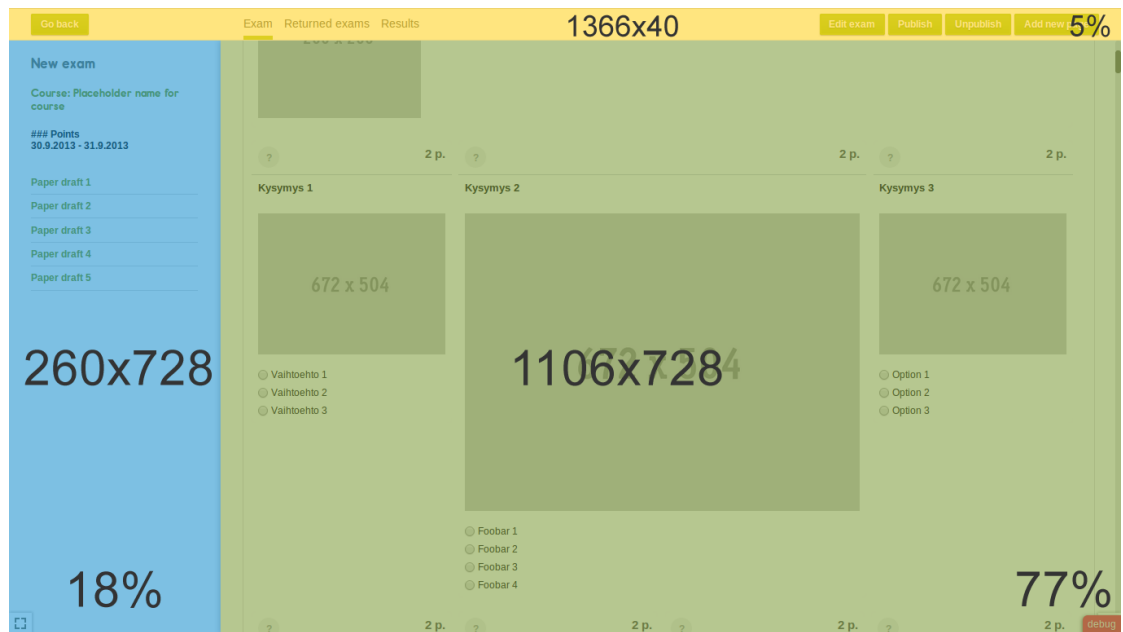
```

KUVA 34. Ote palautetun kokeen Mako-templaattista

Rutiineista on selkeä apu templaatteja tulkitessa. Kuva 34 on ote palautettujen kokeiden lähdekoodista. Huomataan, että templaattista voidaan nopeasti tulkita näkymän sisältö. Huomattakoon muuten, että kuvan 34 *for*-silmukan *xrange*-funktio oli havainnollistava eikä esiintyisi lopputuotteessa. Silmukka demonstroi, kuinka useita digipapereita renderöidään näkymälle. Rutiinissa *card_container* on nähty myös avainsanaparametrien hyöty: avainsanat denotoivat passattavien olioiden tarkoituksen.

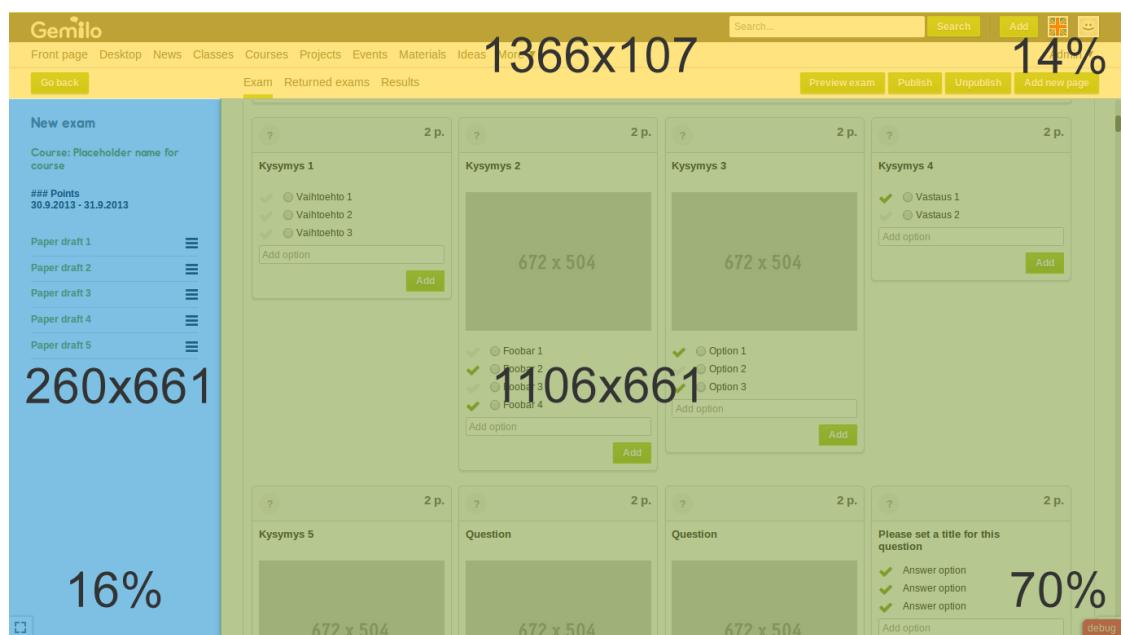
Rutiinien rakentamisessa on pyritty siihen, että rutiinit olisivat hyvin ositettuja. Esimerkiksi tästä syystä kaikki sisältöä *Exam* käyttävät rutiinit vastaanottavat sen parametrina. Näin rutiinit eivät ylläpitäisi riippuvuutta globaalin tason kontekstioliioon *c.exam*.

Kiireestä johtuen kaikki rutiinit eivät saaneet tasa-arvoista kohtelua parametriensa suhteen. Esimerkiksi kuvassa 15 kaikki rutiinit eivät saaneet oikeuksienhallintaan liittyviä *can_view*-, *can_edit*-, *can_edit_object_access*- tai *can_delete*-avainsanaparametreja. Tämä johtui siitä, ettei parametreja käytetty rutiinien sisällä. Kuitenkin templaattirajapinnan yhtenäisyyden vuoksi olisi järkevää sisällyttää kaikki edellä mainitut muuttujat kunkin rutiinin parametrilistaan.



KUVA 35. Kokeen esikatselunäkymän suhteellinen elementtien pinta-alan käyttö

Käytettävyyttä on tarkasteltu vain tilankäytön kannalta. Tilankäyttö on olennaista mobiililaitteille, tablettitietokoneille sekä pieniresoluutioisille tietokoneille. Sen huomioiminen tuli olennaiseksi siinä vaiheessa, kun erilaisia tilaa vieviä kokeen hallintaan liittyviä näkymiä ilmaantui yläpaneelin rinnalle. Todettiin, että hallitseva elementti ei saisi olla navigaatio vaan itse sisältö. Tästä syystä johtuen tähän aspektiin on perehdytty tämän kehittämistehtävän arvioinnissa.



KUVA 36. Kokeen muokkausnäkömön suhteellinen elementtien pinta-alan käyttö

Verkkosivun sisällön suhteellinen koko pitäisi olla vähintään 50 prosentista mieluiten aina 80 prosenttiin saakka sivun pinta-alasta. Sivun navigointiapuväleineille tulisi jättää alle 20 prosenttia. (Nielsen 2000, 22.) Kuvassa 35 havainnollistetaan sisällön ja navigointiapuväleiden suhteellinen pinta-ala. Huomattiin, että suhteellisissa pinta-aloissa on pysytty Nielsenin mainitsemisissa luvuissa. Kun koe oli muokkaustilassa kuvan 36 mukaisesti, tippui sisällön suhteellinen pinta-ala noin 70 prosenttiin. Tarkastelussa on otettu huomioon vain kannettavan tietokoneen näyttöresoluutio, joka kuvassa 35 ja kuvassa 36 oli 1366x768 pikseliä. Voidaan siis tulla johtopäätökseen, että sisällölle oli teoriassa annettu riittävästi tilaa – pois lukien pienen resoluution responsiivinen näkymätila, jonka tilankäyttöä ei tarkasteltu.

5 PÄÄTÄNTÖ

Kehittämistehtävän kesto oli työn ohessa 3–4 viikkoa ja sen ensisijaisena tavoitteena oli saada toimiva esimerkki Opetuksen uusia tuulia -tapahtumaan 10.10.2013. Vaikka alustavasti työtä tehtiin muutamana päivänä syyskuun aikana, kasvoi työurakka täysipäiväiseksi kolme viikkoa ennen aikarajaa. Tavoitteen saavuttaminen epäonnistui muutamista syistä: käyttötapauksia oli paljon ja erilaisten tilojen kartoittaminen ei ollut täysin triviaali tehtävä, lisäksi näkymien ja backend-logiikan yhdistämiseen ei ollut priorisoitu tarpeeksi resursseja. Vaihtoehtoisena ratkaisuna tarjottiin kuvankaappauksia tuotteen toiminnasta. Kuvankaappaukset otettiin sellaisista ominaisuuksista, jotka kuviteltiin olevan merkittäviä palvelun kokonaisuuden kannalta. Joitakin ominaisuuksia, kuten kappaleiden raahausta, esiteltiin, jotta palvelu voisi erottautua kilpailijoistaan. Kaiken kaikkiaan tilaajan mielestä näkymissä oli toteutettu niitä asioita, joita asiakasvaatimukseen oli määritelty. Jäi kuitenkin kirjaamatta asiakasvaatimus, jossa palvelulla ei välttämättä tehtäisi pelkkiä kokeita mutta myös kyselyjä tai pelkkiä digipapereita, jotka koostuisivat vain tekstistä ja kuvista.

Lukuun ottamatta ensisijaista tavoitettaan kehittämistehtävä onnistui yleisellä tasolla. Joitain puutteita oli selkeästi todettavissa, ja välttämättä kaikkien näkymien puutteita ei tiedetty vielä kehitystyön valmistuttuakaan. Kokonaiskuva kuitenkin saatiin rakennettua, ja näkymät saatiin aikarajaan mennessä koostettua.

Ei voida sanoa puhtaasti subjektiivisen arvioon perusteella, onnistuiko templaattien modularisointi. Objektivisempaa arviota varten olisi tarvittu vähintään vertaisten suorittama koodiarviointi, jota ei tässä opinnäytetyössä toteutettu. Myös diskreetimpi hyvän rutiinin kriteeristö olisi auttanut objektiivisen arvioon muodostamisessa. Lisäksi koska näkymien toiminnallisuutta ei implementoitu eikä sitä kyetty testaamaan käyttäjillä, on hankala sanoa, mitä kaikkia näkymiä ei ole mietitty tai kuinka käytettävä palvelu oikeasti olisi. Ensimmäiset näkymät muodostavat kuitenkin hyvän pohjan koenäköjen ja tulevaisuudessa Digipaperin implementoinnille.

Jatkokehityksen kannalta olennaista olisi tutkia, kuinka paljon kysyntää palvelulla oikeasti on. Tämä tarkoittaa sitä, että palvelua saadaan testattua ja näin palvelun laadusta ja toiminnallisuuksista kyetään saamaan suoraa palautetta mahdollisimman nopeasti. Palvelun tarkoituksena ei suoranaisesti siis ole ollut onnistua tilaajan määrittelemillä asiakasvaatimuksilla, vaan tarkoituksena on myös ollut saada vietyä palvelu nopeasti asiakkaille. Tämän opinnäytetyön kehitystyön jälkeen Digipaperi-palvelua ei ole kehitetty eteenpäin. Tuotteen kehityksen kannalta olennaista olisi selvittää vastaus siihen, onko tuotteelle aitoa kysyntää markkinoilla. Luonnollinen jatkokehityksen aihe olisikin toteuttaa kehitystyön backend-logiikka ulkoiselle asiakkaalle niin, että asiakasvaatimukset saataisiin kohdennettua oikein nopeatahtisella asiakaspalautteella.

LÄHTEET

Bayer, Michael 2013. Defs and Blocks. Mako Templates. WWW-dokumentti. <http://docs.makotemplates.org/en/latest/defs.html>. Ei päivitystietoja. Luettu 20.10.2013.

Bayer, Michael 2013. Inheritance. Mako Templates. WWW-dokumentti. <http://docs.makotemplates.org/en/latest/inheritance.html>. Ei päivitystietoja. Luettu 10.11.2013.

Bayer, Michael 2013. Mako Templates for Python. Mako Templates. WWW-dokumentti. <http://www.makotemplates.org/>. Ei päivitystietoja. Luettu 2.9.2013.

Bayer, Michael 2013. Syntax. Mako Templates. WWW-dokumentti. <http://docs.makotemplates.org/en/latest/syntax.html>. Ei päivitystietoja. Luettu 20.10.2013.

BCS The Chartered Institute for IT 2012. Cloud Computing: Moving IT out of the Office. British Information Society (BCS). E-kirja. <http://site.ebrary.com.ezproxy.mikkeli.amk.fi>. Päivitetty 1.3.2012. Luettu 21.10.2013.

Crockford, Douglas 2006. Request for Comments: 4627. Ietf.org. WWW-dokumentti. <http://www.ietf.org/rfc/rfc4627.txt?number=4628>. Päivitetty 1.6.2006. Luettu 31.7.2013.

Crockford, Douglas 2008. JavaScript: The Good Parts. Sebastopol: O'Reilly Media.

Denz, Brian & Durant, John R. 2003. XML Programming Bible. Indianapolis: Wiley Publishing, Inc.

Dutta, Sunava 2006. Native XMLHttpRequest object. Windows Internet Explorer Engineering Team Blog. Päivitetty 24.1.2006. Luettu 5.8.2013.

Ebay 2013. How Important is Screen Resolution when Buying A Tablet? Ebay. Artikkel. <http://www.ebay.com/gds/How-Important-is-Screen-Resolution-when-Buying-A-Tablet-/10000000177629444/g.html>. Päivitetty 20.8.2013. Luettu 16.10.2013.

ECMA International 2011. Standard ECMA-262 5.1 Edition. PDF-dokumentti. <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>. Päivitetty 3.6.2011. Luettu 31.7.2013.

Flanagan, David 2011. JavaScript: The Definitive Guide. Sebastopol: O'Reilly Media, Inc.

Gamma, Erich, Helm, Richard, Johnson, Ralph & Vlissides, John 2001. Olio-ohjelmointi - Suunnittelumallit. Helsinki: Oy Edita Ab.

Garrett, Jesse James 2005. Ajax: A new Approach to Web Applications. Adaptive Path. WWW-dokumentti. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>. Päivitetty 18.2.2005. Luettu 5.8.2013.

Haikala, Ilkka & Mikkonen, Tommi 2011. Ohjelmistotuotannon käytännöt. Hämeenlinna: Kariston Kirjapaino Oy.

He, Henry 2012. How jQuery works. CodeProject. WWW-dokumentti. <http://www.codeproject.com/Articles/426013/How-jQuery-works>. Päivitetty 22.7.2012. Luettu 8.8.2013.

Järvinen, Petteri 2003. IT-tietosanakirja. Porvoo: Docendo Finland Oy.

Koch, Peter-Paul 2013. The AJAX response: XML, HTML, or JSON? QuirksMode. WWW-dokumentti. http://www.quirksmode.org/blog/archives/2005/12/the_ajax_respon.html. Päivitetty 5.8.2013. Luettu 5.8.2013.

Kokkarinen, Ilkka 2004. Java, Prolog ja Python: Tehokas näkökulma ohjelmointiin. Helsinki: Edita Prima Oy.

Lindley, Cody 2013. DOM Enlightenment. Sebastopol: O'Reilly Media, Inc.

McConnell, Steve 2004. Code Complete. Redmond: Microsoft Press.

Nielsen, Jakob 2000. WWW-suunnittelu. Helsinki: Oy Edita Ab.

Otero, Cesar, Gorkov, Alexei & Larsen, Rob 2012. Professional JQuery. Wrox. E-kirja. <http://site.ebrary.com.ezproxy.mikkeli.amk.fi>. Päivitetty 1.3.2012. Luettu 14.10.2013.

Python 2013. 2 PEPs 252 and 253: Type and Class Changes. Python. WWW-dokumentti. <http://docs.python.org/release/2.2.3/whatsnew/sect-rellinks.html>. Ei päivitystietoja. Luettu 5.11.2013.

Schafer, Steven M. 2010. HTML, XHTML, and CSS. Indianapolis: Wiley Publishing Inc.

Severance, Charles 2012. Java Script: Designing a Language in 10 Days. WWW-dokumentti. <http://www.computer.org/csdl/mags/co/2012/02/mco2012020007.html>. Päivitetty 1.2.2012. Luettu 31.7.2013.

Van Rossum, Guido 2009. First-class Everything. WWW-dokumentti. <http://python-history.blogspot.fi/2009/02/first-class-everything.html>. Päivitetty 27.2.2009. Luettu 2.9.2013.

W3C 2013 a. The HTML Syntax. W3C. WWW-dokumentti. <http://dev.w3.org/html5/spec-author-view/syntax.html>. Ei päivitystietoja. Luettu 7.8.2013.

W3C 2013 b. HTML 5.1 Nightly. W3C. WWW-dokumentti. <http://www.w3.org/html/wg/drafts/html/master/single-page.html>. Päivitetty 7.8.2013. Luettu 7.8.2013.

World Wide Web Consortium 2005. Document Object Model (DOM). WWW-dokumentti. <http://www.w3.org/DOM/>. Päivitetty 19.1.2005. Luettu 4.8.2013.

York, Richard & Pouncey, Ian 2011. *Beginning CSS: Cascading Style Sheets for Web Design* (3rd Edition). Wrox. E-kirja. <http://site.ebrary.com.ezproxy.mikkeli.amk.fi>. Päivitetty 1.5.2011. Luettu 14.10.2013.

Yhden näkymän Mako-templaattiriippuvuudet

