Asiakirjasovelma Vaadin-sovelluskehyksellä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulun opinnäytetyö

Tietotekniikan koulutusohjelma

Riihimäki, syksy 2016

Jermu Toiviainen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Riihimäki
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tekijä Jermu Toiviainen **Vuosi** 2016

Työn nimi Asiakirjasovelma Vaadin-sovelluskehyksellä

TIIVISTELMÄ

Nykyaikana sovelluskehitys on suureksi osaksi siirtynyt pois työasemakohtaisista ohjelmista web-sovelluksiin. Web-sovelluksien etuna on käyttäjäystävällisyys, enää käyttäjän ei tarvitse huolehtia latauksesta, asennuksesta ja päivityksestä. Web-sovelluksien ansiosta yritykset voivat myydä ohjelmistojaan palveluina.

Opinnäytetyön aiheena oli evaluoida Vaadin-sovelluskehystä ja rakentaa sillä Liferay-portaaliin sovelma. Sovelman oli tarkoitus hakea sopimus metaluokalla merkittyjä asiakirjoja Tweb-järjestelmästä. Työn toimeksiantajana toimi Triplan Oy.

Opinnäytetyö käsittelee Liferaysta ja Vaatimesta työn kannalta oleellisimmat tekniikat ja käsitteet. Kävin läpi sovelman arkkitehtuurin, käytön ja kehitysmenetelmän. Lopuksi tein yhteenvedon projektista, jossa pohdin missä projektissa onnistuttiin ja missä oli parannettavaa. Lisäksi kävin läpi mahdolliset jatkokehitysehdotukset.

Lopputuloksena saatiin aikaan toimiva sovelma, joka vastasi sille asetettuja tavoitteita. Työstä voidaan sanoa että Vaadin vastaa hyvin sille asetettuihin tavoitteisiin. Sillä saadaan aikaiseksi joustavalla tavalla erilaisia web-sovelluksia, jotka toimivat muun muassa Liferay-portaalissa portlettina. Tutkimuksesta saatiin kuitenkin selville, että uusi Liferay 7:n kokonaan muutettu OSGi-pohjainen modulaarinen kehitysmalli asettaa omat haasteet Liferayn vanhemmille alustoille kehitettyjen portlettien päivittämisen uuteen versioon.

Avainsanat Vaadin, Liferay, web-sovelluskehitys, Java

Sivut 27 s. + liitteet 1 s.

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Riihimäki
Degree Programme in Information Technology
Software technology

Author Jermu Toiviainen **Year** 2016

Subject of Bachelor's thesis Document portlet with Vaadin framework

ABSTRACT

In today's world software development has largely moved away from desktop programs to web development. The advantage of web applications is that they are more user-friendly. No more does the user have to worry about downloading, installing and updating the software. Now companies can sell their Web applications as a service.

The subject of this thesis was to evaluate Vaadin framework and build a Liferay portlet with it. The portlet was meant to search for documents that had contract metadata in them. The portlet was produced for the use of Triplan Oy.

This thesis covers the technologies and essential terms of Liferay and Vaadin. It also covers the portlet's architecture, usage and development method. Finally there is a brief synopsis of the project in which I cover the parts of the project that were successful and the things that could be improved upon. In addition, I will go through the further development plans.

The result was a working portlet that met the goals that were set to it. Vaadin also worked well for what it was meant to do. It's flexible enough that it suits the needs of web development and a Liferay portlet. The research shows that the new OSGi based modular development method in Liferay 7 adds new challenges for converting older portlets to the new Liferay version.

Keywords Vaadin, Liferay, web development, Java

Pages 27 p. + appendices 1 p.

SISÄLLYS

1 JOHDANTO

Ohjelmistot ovat jo pitemmän aikaa tehneet siirtymää työpöytäsovelluksista selainpohjaisille sovelluksille. Selainpohjaiset sovellukset ovat yleisesti helpompia käyttää että ylläpitää. Organisaatiot haluavat usein sisäisen ratkaisun niiden asianhallinnalle. Silloin Tweb ja räätälöity portaaliratkaisu tulee esiin.

Liferay on yhdysvaltalainen portaali-valmistaja ja se on otettu alustaksi Triplan portaalille. Liferay-portaali on yleisesti käytössä organisaatioilla sisäisesti esimerkiksi intrana. Portaaliin on mahdollista luoda erilaisia käyttäjäryhmiä. Näiden avulla on helppo määrittää mihin sovelmiin ja mihin toimintoihin henkilöllä on oikeuksia.

Opinnäytetyön tavoitteena oli toteuttaa Liferay-portaalille asiakirjojen käsittely sovelma Vaadin-sovelluskehyksellä. Työn tarkoituksena oli demota Vaadin-sovelluskehyksellä rakennettua sovelmaa Triplan-portaalissa. Työssä käytettiin hyödyksi Tweb-asianhallintajärjestelmää. Tweb-järjestelmään tallennettuja asiakirjoja haettiin rajapinnan kautta sovelmaan näytettäväksi. Opinnäytetyön toimeksiantaja toimi Triplan Oy.

Aikaisemmin tehdyissä sovelmissa on tavallisesti käytetty Bootstrap-sovelluskehystä yhdessä JSP sivujen kanssa. Tarkoitukseen sopivia sovelluskehyksiä on lukuisia muitakin. Työhön kuitenkin valittiin Vaadin. Yksi Vaadin-sovelluskehyksen tuomista eduista on, ettei sen kanssa tarvitse kirjoittaa erikseen HTML ja Javascript koodia vaan kaikki tapahtuu Java koodin kautta, mikä helpottaa ja nopeuttaa kehitystyötä.

Opinnäytetyö painottuu enemmän Vaadin-sovelluskehyksen tutkimiseen Liferayta enemmän, koska Liferay toimi työssä vain sovelman alustana. Itse asiakirjasovelma on kuitenkin työn pääosassa. Sovelmasta käydään läpi muun muassa sen toiminta, arkkitehtuuri ja kehitysmenetelmä. Myös Twebin perusteita ja dynaamisia metatietoja selvitetään.

Työ on toiminnallinen, joten tavanomaisille tutkimusmenetelmille ei ollut juuri käyttöä. Toiminnallisuus perustuu sovelluksen ohjelmointiin ja testaukseen. Tutkimusosuus perustuu työssä käytettävien teknologioiden opiskeluun, jotta niitä osasi käytännössä hyödyntää. Tämän lisäksi tutustuin muiden kehittäjien ohjelmakoodiin.

2 PROJEKTIN TAVOITTEET JA TEHTÄVÄT

Maaliskuussa sain ohjelmistosuunnittelijan harjoittelijapaikan Triplanilta (Kuva 1). Triplan Oy on vuonna 1991 perustettu ohjelmistotalo, joka on erikoistunut asian- ja dokumentin hallintaan. Triplanin tärkein tuote on nimeltään Tweb. Se on selainkäyttöinen asian- ja asiakirjanhallinnan työväline. Triplanin asiakkaat koostuvat pääasiassa julkishallinnon organisaatioista, joita ovat muun muassa kaupungit, koulut, sairaanhoitopiirit, ministeriöt, valtiohallinnon virastot ja laitokset.

Projektin tavoitteena oli luoda asiakirjasovelma, joka mahdollistaisi organisaation Tweb järjestelmään tallennettujen sopimusten käsittelyn Triplan-portaalin kautta.

Kuva 1. Triplan Oy:n logo

Opinnäytetyö lähti liikkeelle toimeksiantajan kiinnostuksesta Vaadin-sovelluskehystä kohtaan. Se halusi testata Vaadin-sovelluskehyksellä tehtyä portlettia Liferay-pohjaisessa portaalissa. Minulla ei ollut aikaisempaa kokemusta Vaatimesta tai Liferaysta. Reilu kuukausi meni pitkälti kyseisten tekniikoiden tutkimiseen ja opiskeluun. Luin aiheesta kirjallisuutta, tein pieniä harjoitusprojekteja ja seurasin muita kokeneempia kehittäjiä. Samalla tutustuin WS-rajapinnan käyttöön, jonka kautta saisin haettua dataa Tweb sovelluksen tietokannasta sovelmaa varten.

Huhtikuussa pidetyssä palaverissa käytiin yksityiskohtaisemmin projektia läpi. Sovelman tarkoitus oli hakea Tweb järjestelmästä sinne tallennetut sopimukset niiden metatietoihin perustuen, jotka käyttäjä voisi sitten ladata itselleen. Käyttäjä voisi selata ja rajata sopimuksia niiden metatietojen avulla. Sovelman etuna olisi yksinkertaisempi tapa selata sopimuksia Tweb järjestelmän sijaan. Palaverissa sain sovelman alustavan vaatimusmäärityksen (Kuva 2 ja Kuva 3). Se sisälsi muun muassa tiedot käytettävistä metaluokista ja sarakkeiden nimistä.

1. Sovelmassa käytettävät Sopimus-metaluokan metatiedot

Sopimus-metaluokka, mcid=1001

Sopimuksen tila, maid=2004
arvot: Ei valintaa, mid=0
Vireillä, mid=1
Voimassa, mid=2
Päättynyt, mid=3

Sopimuksen voimassaolo, maid=2068
arvot: Ei valintaa, mid=0
Määräajan, mid=1
Toistaiseksi, mid=2
Kertaluontoinen, mid=3

Sopimusnumero, maid=2001

Sopimuskumppani, maid=2016

Vastuuhenkilö/organisaatio, maid=2096

Kuva 2. Sovelmassa käytettävät metatiedot.

Sarake 1: Asiakirjan nimike
Sarakeotsikko: Nimike

Sarake 2: Sopimusnumero, maid=2001
Sarakeotsikko: Nro

Sarake 3: Vastuuhenkilö/organisaatio, maid=2096
Sarakeotsikko: Vastuuhenkilö/organisaatio

Sarake 4: Sopimuksen tila, maid=2004
Sarakeotsikko: Tila

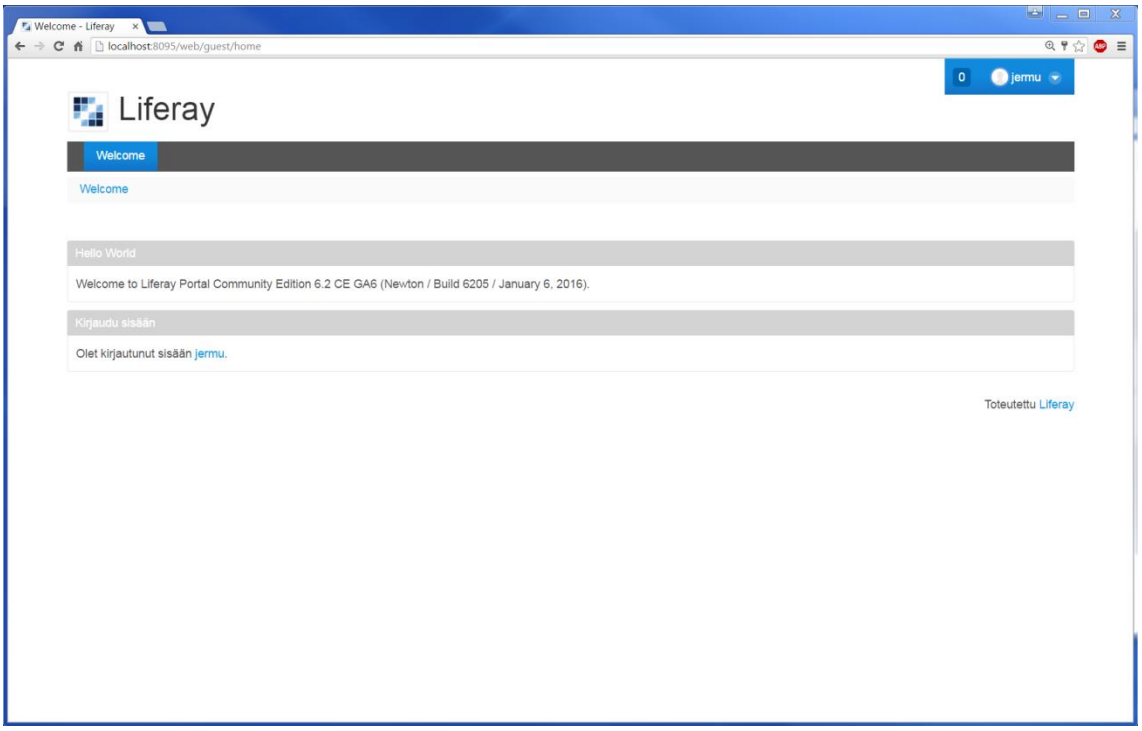
Sarake 5: Sopimuksen voimassaolo, maid=2068
Sarakeotsikko: Voimassaolo

Kuva 3. Taulukossa esitettävien sarakkeiden otsikot.

3 LIFERAY-PORTAALI

Lifery on avoimeen lähdekoodiin perustuva portaalialusta, joka on kirjoitettu Javalla. Liferayn perusti Brian Chan vuonna 2000. Liferaylla on tarjolla kaksi eri versiota tuotteestaan. Ensimmäinen on Liferay Portal Enteprise Edition, joka on tarkoitettu kaupalliseen käyttöön. Siihen kuuluvat päivitykset sekä täysi tuki. Liferay Portal Community Edition on ilmainen versio ja sitä tuetaan yhteisön voimin. (Valkeapää 2014; Wikipedia 2016a.)

Portaali koostuu useista yksittäisistä portleteista (Kuva 4). Portletti on käyttöliittymään liitettävä sovellus, joka näytetään portaalissa. Liferay tarjoaa paljon valmiita portletteja eri toiminnallisuuksilla, kuten sisällönjulkaisija, blogi ja kalenteri, mutta myös omien portlettien teko onnistuu. Liferayssa on erilaisia toiminnallisuuksia kuten käyttäjä– ja roolienhallinta. (Valkeapää 2014.)



Kuva 4. Testi-portaalin oletusnäkymä.

3.1 Keskeiset työkalut ja käsitteet

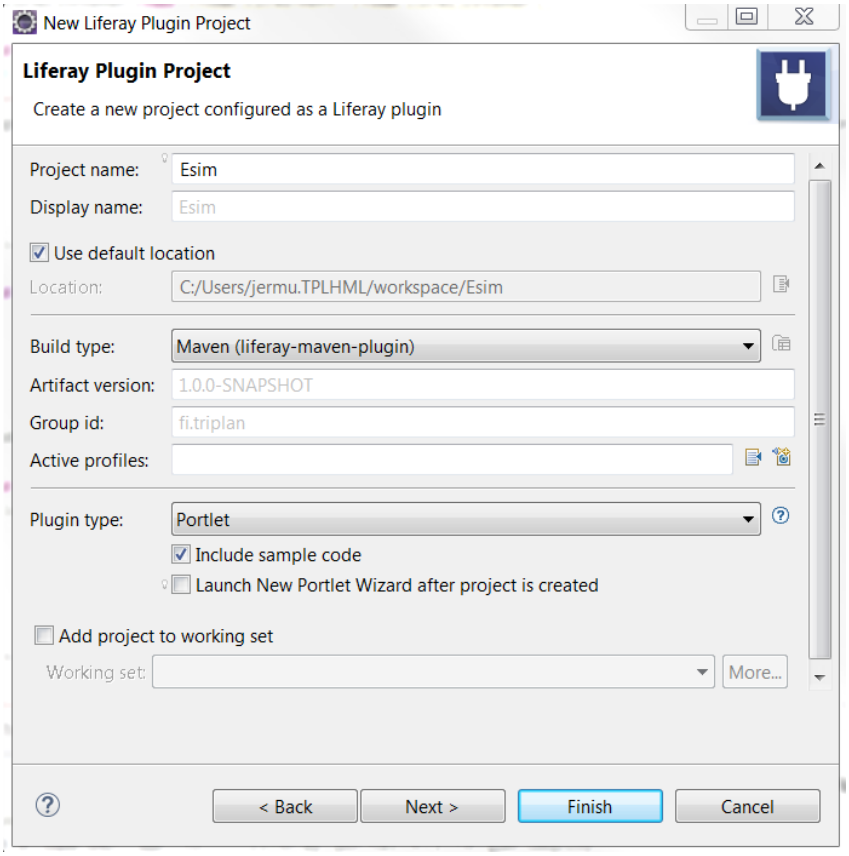
Käyn läpi Liferayhin liittyviä keskeisiä käsitteitä. Käsitteet ovat oleellisia ymmärtää projektin kulun kannalta ja ovat muutenkin alalla yleisesti käytettyjä normeja.

Valitsin seuraavat käsitteet sillä perusteella, että ne liittyvät oleellisesti tekemääni opinnäytetyöhön. Rajaan kuitenkin Liferayn omat ulkoasut ulos esittelystä, koska ulkoasu toteutetaan Vaatimen avulla.

3.1.1 Liferay IDE

Liferay IDE (Integrated Development Environment) on kehitysympäristöön asennettava lisäosa, joka helpottaa työskentelyä Liferay-projektien kanssa. Liferay IDE:llä on tuki Plugins SDK:ta varten, minkä ansiosta projektin voi luoda ilman komentorivin käyttöä. (Sezov Richard Jr. 2012, 305–307.)

Liferay IDE tarjoaa projektinluonti velho-työkalu, joka antaa käyttäjän myös valita projektin tyypin kuten portletti, teema tai layout. Siihen määritellään projektin tiedot, kuten projektin nimi, ryhmän id ja liitännäisen tyyppi (Kuva 4). (Sezov Richard Jr. 2012, 305–307.)



Kuva 5. Projektinluonti Liferay IDE:n avulla.

3.1.2 Portlet

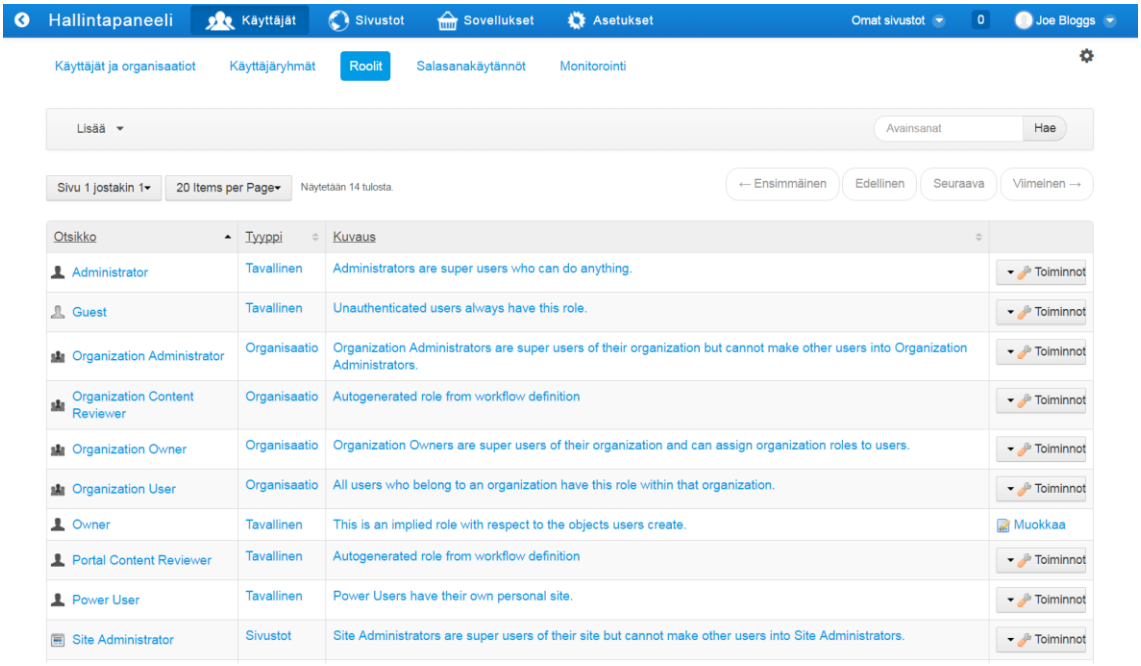
Kävin jo Liferayn alkukappaleessa lyhyesti läpi mikä on portletti, mutta käyn sitä nyt vielä tarkemmin läpi. Portletti on yleisin käytetty Liferay liitännäinen. Yksinkertaistettuna portletti on komponentti joista Liferay-portaali rakentuu. Portletteihin voidaan esimerkiksi rakentaa jonkinlaista toiminnallisuutta, tai se voi toimia sisällön näyttäjänä. Kuka tahansa pystyy rakentamaan oman portletin vastaamaan omia tarpeita, jos Liferayn valmiit ratkaisut eivät sovi. Halutessaan portletin ulkoasun voi tehdä vastaamaan Liferayn omaa standardia, jotta käyttäjä ei erota sitä Liferayn omasta portletista. (Liferay 2016a.)

Liferay antaa kehittäjälle hyvät mahdollisuudet valita, mitä sovelluskehystä käyttää. Vaihtoehtoihin lukeutuvat muun muassa Struts, Spring MVC, JSF ja Vaadin. Portletti projekti luodaan Liferay IDE:n avulla. Liferay IDE myös tarjoaa hot deploy toiminnon, jonka avulla portletin pystyy päivittämään portaaliin ilman uudelleen käynnistystä. (Liferay 2016.)

3.1.3 Käyttöoikeudet

Liferayssa käyttäjät jaetaan erilaisiin ryhmiin, joille voidaan jakaa rooleja. Rooli on käytännössä kokoelma erilaisia oikeuksia. Hallintapaneeliin käyttäjät välilehden alta löytyy roolit osuus, josta voi muokata eri roolien oi-

keuksia (Kuva 6). Roolien oikeuksia pystyy asettamaan portletti-tasolla. (Liferay 2016b.)



Kuva 6. Roolien hallintapaneeli

Toiminnot nappi avaa alasvetovalikon, josta voi muuttaa roolin oikeuksia. Roolit voidaan asettaa portaali, sivu tai organisaatio -tasolle. Tavallisimmat roolit ovat guest, user ja administrator. (Liferay 2016b.)

3.2 Apache Tomcat

Apache Tomcat on Apache Software Foundation ryhmän vuonna 1999 kehittämä avoimen lähdekoodin web-palvelin. Sitä ylläpitää ja kehittää avoin yhteisö. Apache Tomcat pohjautuu Javaan ja sen avulla pystyy muun muassa ajamaan Java servlettejä ja JSP (Java Server Page) sivuja. Tomcat vaatii toimiakseen vähintään JRE (Java Runtime Environment) ympäristön. (The Apache Software Foundation.)

Projektissani Liferay oli asennettu Tomcat palvelimen päälle. Samalla Vaadin myös käyttää Tomcat alustaa sen servlettien kanssa. Esimerkiksi projektissani Liferay oli asetettu porttiin 8095 ja se löytyi osoitteesta http://localhost:8095/.

4 VAADIN

Vaadin on Suomessa vuonna 2000 perustettu ohjelmistoalan yritys (kuva 7). Tuote on Java sovelluskehys, joka on tarkoitettu modernien rikkaiden Internet web-sovellusten kehittämiseen. Rikkaat Internet-sovellukset termi on käännös englanninkielisestä sanasta ”Rich Internet Applications” ja se tarkoittaa web-sovellusta, joka muistuttaa ominaisuuksiltaan työpöytäso-

vellusta (Wikipedia 2016b). Vaadin perustuu avoimeen lähdekoodiin ja se toimii Apache 2 lisenssin alaisuudessa. (Vaadin 2016a.)

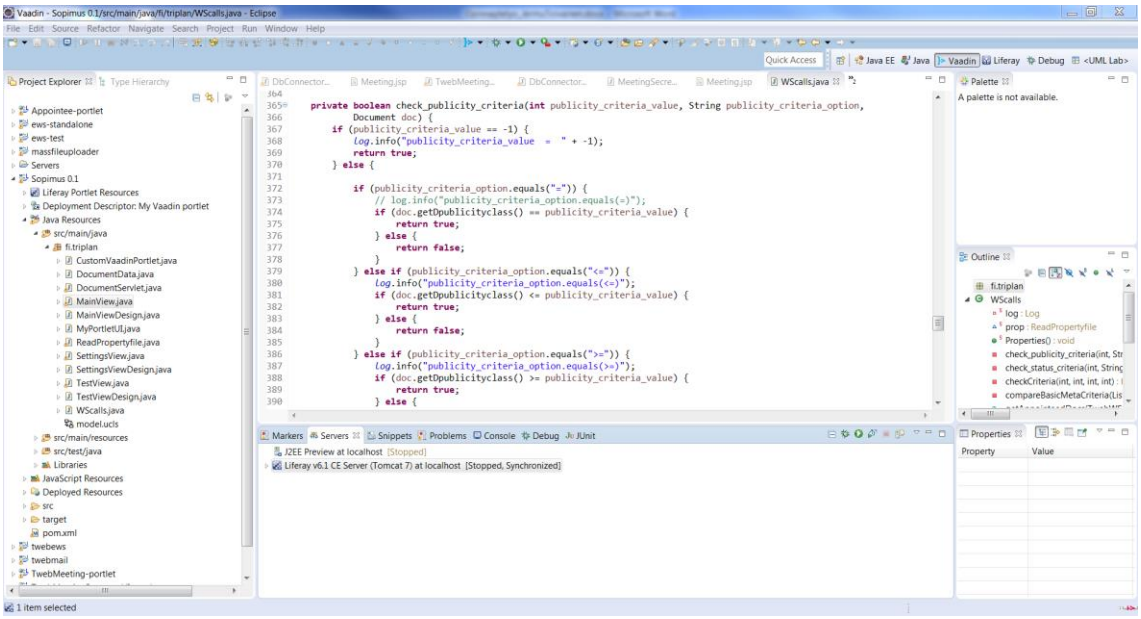
Kuva 7. Vaadin logo.

Vaatimella kehittäminen on tehokkaampaa verrattuna perinteiseen web-kehittämiseen HTML:llä ja JavaScriptillä. Se antaa kehittäjän keskittyä sovelluksen logiikkaan käyttöliittymän sijasta. Sovelluksen kehittäminen muistuttaa pitkälti Java-työpöytäsovelluksen kehittämistä AWT, Swing tai SWT Java toolkitillä. Palvelinpuolen sovelluskehys pitää huolen käyttöliittymästä selaimessa. Vaadin sovellus toimii Java servlettinä palvelimella. (Vaadin 2016a.)

4.1 Eclipse

Vaikka Eclipse ei suoranaisesti liity Vaadin-sovelluskehykseen otin sen käsiteltäväksi tähän yhteyteen, koska käytin projektissa paljon Vaadin-sovelluskehyksen omaa Eclipse näkymää. Eclipse on alusta sovelluskehitystä varten. Yksinään Eclipsestä ei ole paljoa iloa kehittäjälle, vaan se tarjoaa alustan erilaisille liitännäisille. Nämä liitännäiset luovat toiminnallisuuden Eclipseen. Liitännäisen tehtäviin voi esimerkiksi kuulua ohjelmakoodin kääntäminen, debuggaus tai julkaisu. Avoimen arkkitehtuurin ansiosta kuka tahansa pystyy luomaan liitännäisiä. (Eclipse Foundation 2016.)

Käytännössä Eclipse helpottaa ohjelmointia muun muassa koodin automaattisella täydennyksellä, virheentunnistuksella ja koodin korjausehdotuksilla. Projektissa käytin Vaatimen integroitua ohjelmointiympäristöä (kuva 8).



Kuva 8. Esimerkki kuva Eclipse ympäristöstäni.

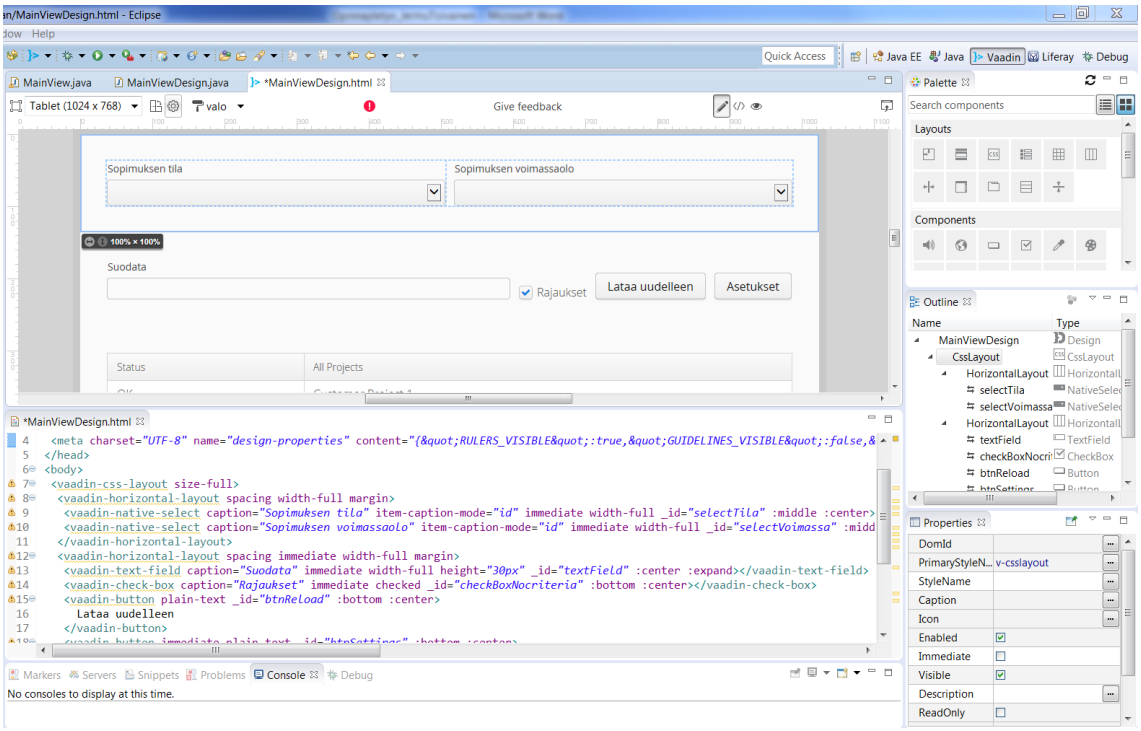
4.2 Vaadin IDE

Vaadin IDE on Vaatimen oma liitännäinen, joka on saatavilla useille eri ohjelmointiympäristöille kuten Eclipse, NetBeans ja IntelliJ IDEA. Liitännäinen tarjoaa useita eri toimintoja sovelluskehityksen avuksi. Sen projektinluontityökalulla pystyy luomaan uuden Vaadin projektin, teeman tai widgetin. (Vaadin 2016b.)

Liitännäisen erikoisuuksiin kuuluu sen tuoma tuki Vaadin Designerille. Sen avulla pystyy luomaan käyttöliittymän helposti raahaa ja liitä -tyylillä. Projektin pystyy julkaisemaan muun muassa servlettinä tai portlettina. (Vaadin 2016b.)

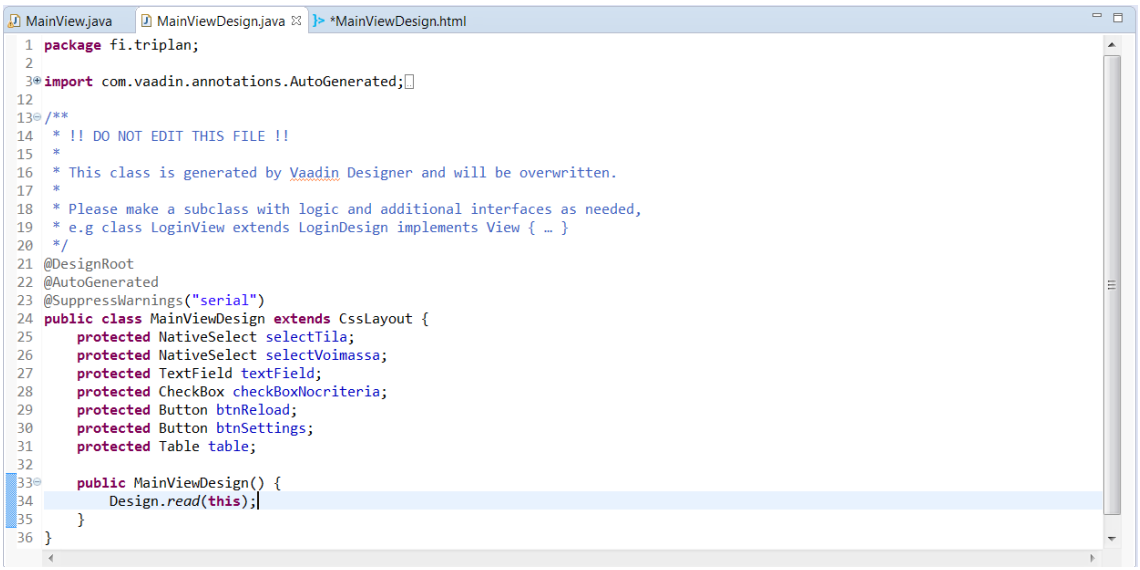
4.3 Vaadin Designer

Vaatimen oma graafinen editori on erinomainen työkalu sovelluksen käyttöliittymää rakentaessa (Kuva 9). Siinä on paljon hyödyllisiä ominaisuuksia, jotka nopeuttavat ja helpottavat sovelluksen tekoa. Käyttöliittymän rakentaminen onnistuu raahaamalla ja tiputtamalla haluttu komponentti paikalleen. Alkuun valitaan pohjaksi haluttu layout. Layotteja voi myös yhdistellä keskenään. (Vaadin 2016d.)



Kuva 9. Vaadin designer käytännössä.

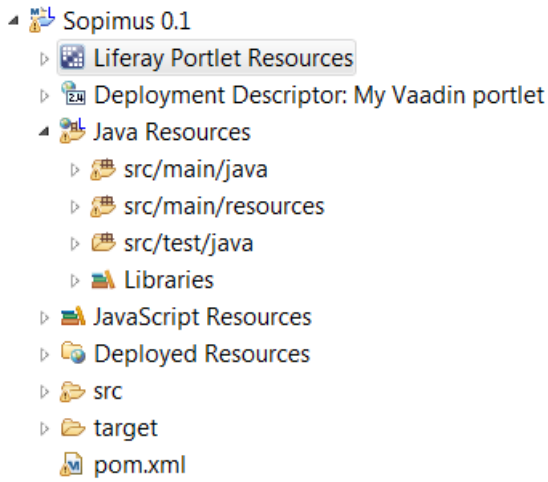
Designer luo käyttöliittymästä kaksi Java tiedostoa. Deklaratiivinen tiedosto, määrittää UI:n. Tätä tiedostoa kutsutaan designiksi ja Vaadin designer automaattisesti ylläpitää sitä, mutta sitä on myös mahdollista muokata käsin. Toista tiedostoa kutsutaan kumppani tiedostoksi. Se on Java tiedosto, joka yhdistää UI komponentit Java logiikkaan (Kuva 10). Tätä luokkaa ei saa itse muokata vaan designer huolehtii sen päivityksestä. Lopuksi jää kehittäjän oma Java koodi johon kirjoitetaan ohjelman logiikka. (Vaadin 2016d.)



Kuva 10. Designin kumppani tiedosto.

4.4 Maven

Maven on työkalu, jota käytetään Java projektien rakentamiseen ja hallintaan (Kuva 11). Maven tähtää olemaan parhaan käytännön kehitystyökalu, eli alalla yleisesti käytetty standardi. Ammattimaisissa Java projekteissa se on pitkälti tämän aseman saavuttanut. Projektin määrittely, toteutus ja yksikkötestit ovat osa normaalia Maven kehityksen elinkaarta. (Maven.)

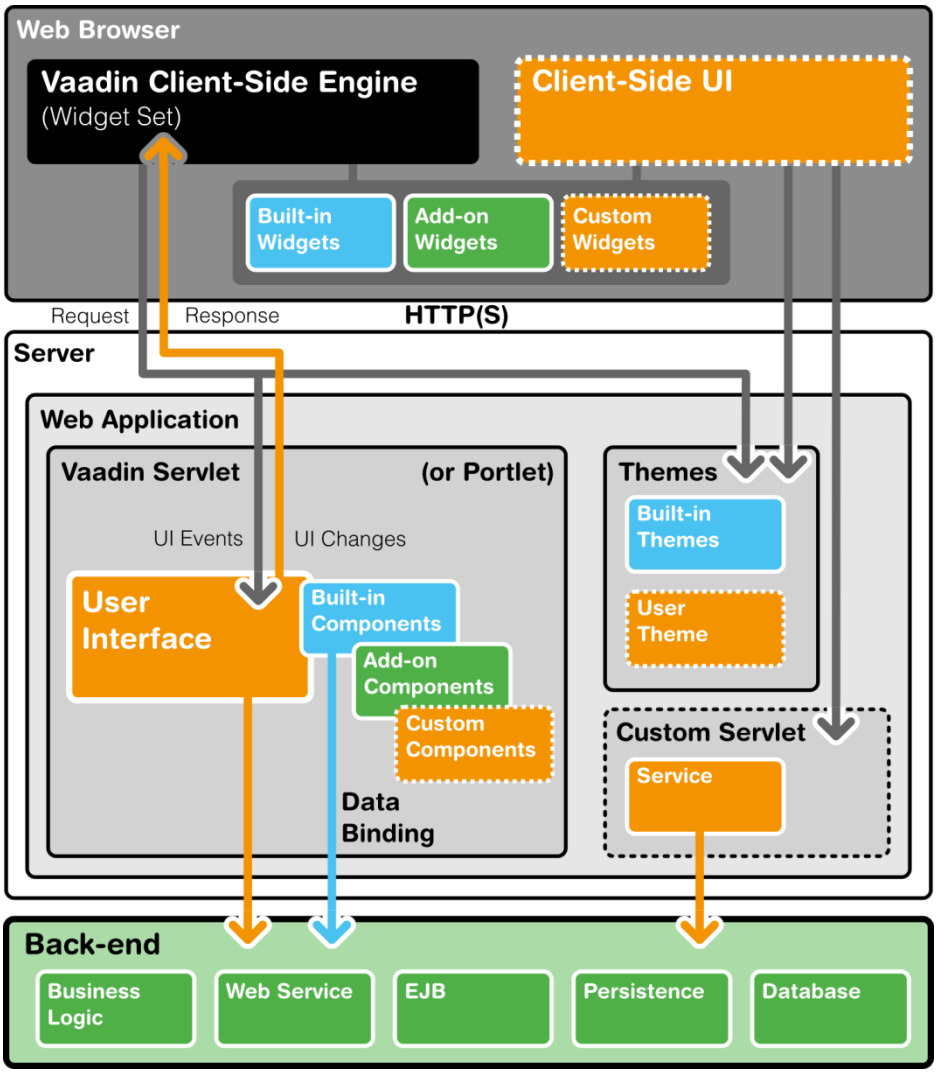


Kuva 11. Maven projektin rakenne.

Maven projektit määritellään POM (project object model) xml tiedostoon, josta löytyy kaikki yksittäisen projektin asetukset. Tavallisessa projektissa POM tiedostosta löytyy projektin nimi, omistaja, liitännäiset ja sen liitännäisyydet muihin moduuleihin. Maven lataa dynaamisesti tarvittavat kirjastot ja Maven liitännäiset yhdestä tai useammasta pakettivarastosta ja tallentaa ne projektin välimuistiin. (Maven.)

4.5 Vaadin arkkitehtuuri

Vaadin mahdollistaa kahden eri ohjelmointimallin käytön (Kuva 12). Palvelinpuolen (Server side) malli on yleisemmin käytössä, koska se on tehokkaampi. Sitten on web puolelta tuttu asiakaspuolella (Client side) tapahtuva malli, mikä tarkoittaa että ohjelmakoodi suoritetaan käyttäjän selaimella. (Vaadin 2016c.)



Kuva 12. Kuvaus Vaadin arkkitehtuurista.

Vaadin-sovelluskehystä käyttäessä koko ohjelma kirjoitetaan Javalla. Kehittäjän ei tarvitse kirjoittaa HTML:ää tai Javascriptiä. Asiakaspuolella voidaan käyttää Javalla kirjoitettuja widgettejä, jotka käännetään selaimella suoritettavaksi JavaScriptiksi. Molempia malleja on mahdollista käyttää sekaisin samassa projektissa. Ajax kutsut huolehtivat palvelimen ja selaimen välisestä tiedonsiirrosta. (Vaadin 2016c.)

4.5.1 Ajax

Ajax tulee sanoista ”Asynchronous JavaScript And XML”. Se on ryhmä erilaisia tekniikoita, joilla viitataan palvelimen ja verkkosivun väliseen keskusteluun. Nykyisin XML on harvemmin enää käytössä vaan JSON on korvannut sen. Web-sovellus ja palvelin vaihtavat pieniä määriä dataa taustalla mahdollistaen web-sovelluksen päivittämisen ilman sivun uudelleen lataamista. Tämä saa sovelluksen muistuttamaan enemmän tavallista työpöytäsovellusta. (Mozilla developer network.)

Vaadin käyttää Ajax tekniikkaa ohjaamaan sillä luotuja sovelluksia. Vaadin sovellukset eivät lataa sivua uudestaan näkymää vaihdettaessa. Tavallisesti tämä tehtäisiin lataamalla uusi HTML-sivu.

4.5.2 GWT

GWT on lyhenne sanoista Google Web Toolkit. Se on Googlen kehittämä avoimen lähdekoodin Java sovelluskehys, joka mahdollistaa monimutkaisten selainpohjaisten sovellusten kehittämisen. GWT:tä käytetään käyttöliittymän kehittämiseen. Käytännössä GWT kääntää Javalla kirjoitetun koodin optimoiduksi JavaScript koodiksi selainta varten. (GWT Project.)

Käyttöliittymä koostuu pitkälti yksittäisistä Widgeteistä. Widget on Javalla koodattu, käyttöliittymän komponentti johon on rakennettu erilaisia toiminnallisuuksia. Esimerkkinä tekstilaatikko, jolle on tehty valmiita metodeja kuten getText() ja setText(). GWT kääntää tämän komponentin selaimelle käytettävään muotoon eli JavaScriptiksi.

GWT on käytössä Googlella useissa eri projekteissa kuten AdWords, AdSense, Flights, Hotel Finder, Offers, Wallet ja Blogger. GWT:n käyttöön vaaditaan GWT SDK paketti. GWT on laajasti käytössä maailmalla ja sitä käyttävät tuhannet kehittäjät. Vaadin asiakaspuolen sovelluskehys on rakennettu GWT:n päälle. (GWT Project.)

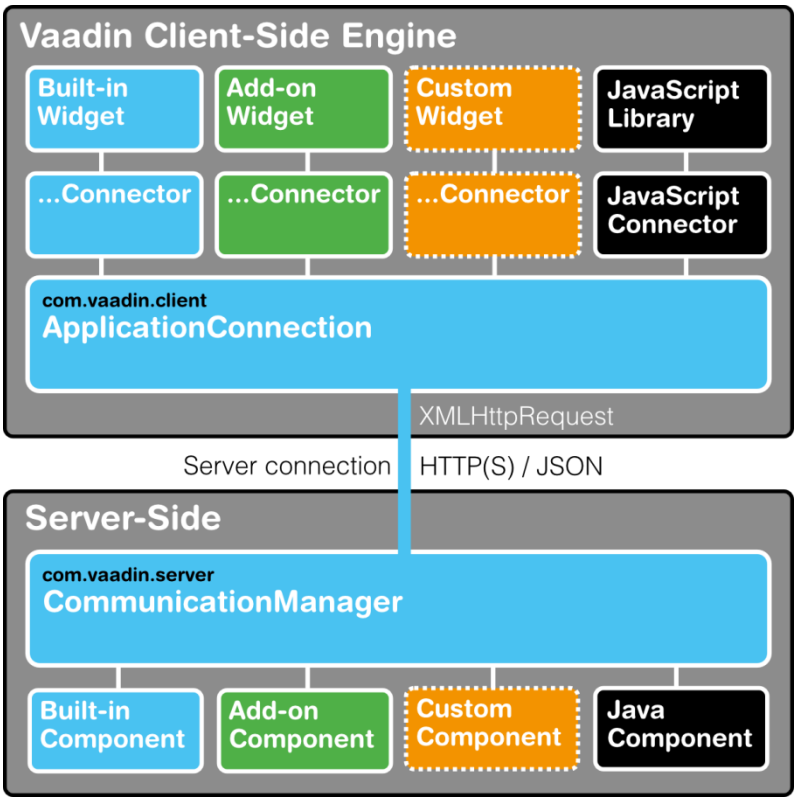
4.5.3 Java servlet

Java servlet on luokka, joka laajentaa web-palvelimen toiminnallisuutta. Yleensä sillä lisätään web-sovellukseen staattista sisältöä (HTML) tai dynaamista sisältöä (JSP). Selain on yhteydessä web-palvelimeen HTTP:n tai HTTPS:n välityksellä. (Oracle.)

Vaadin web-sovellus pakataan tavallisesti WAR (web application archive) tiedostoon. Ne ovat ZIP pakattuja Java JAR paketteja. Web-sovellus on määritetlty `WEB-INF/web.xml`, mikä määrittää URL polut servletteihin ja servlettien luokat. Palvelinpuolen Vaadin käyttöliittymä (UI) toimii juuri servlettinä. Se on VaadinServlet nimisen servletti luokan sisällä, mikä pitää huolen muun muassa sessiosta. Palvelinpuolen käyttöliittymä löytyy UI luokasta. (Vaadin 2016e.)

4.5.4 Asiakaspuolen kehys

Asiakaspuolen kehystä käytetään renderöimään palvelinpuolelta tuleva käyttöliittymä. Javalla kirjoitettu koodi käännetään selaimessa GWT:n avulla JavaScriptiksi. Käyttöliittymän komponenttia kutsutaan widget:ksi. Renderöinti tapahtuu ApplicationConnection tasolla. Se ottaa vastaan CommunicationManager:lta Ajax:n kautta tulevan datan (Kuva 13). (Vaadin 2016f.)

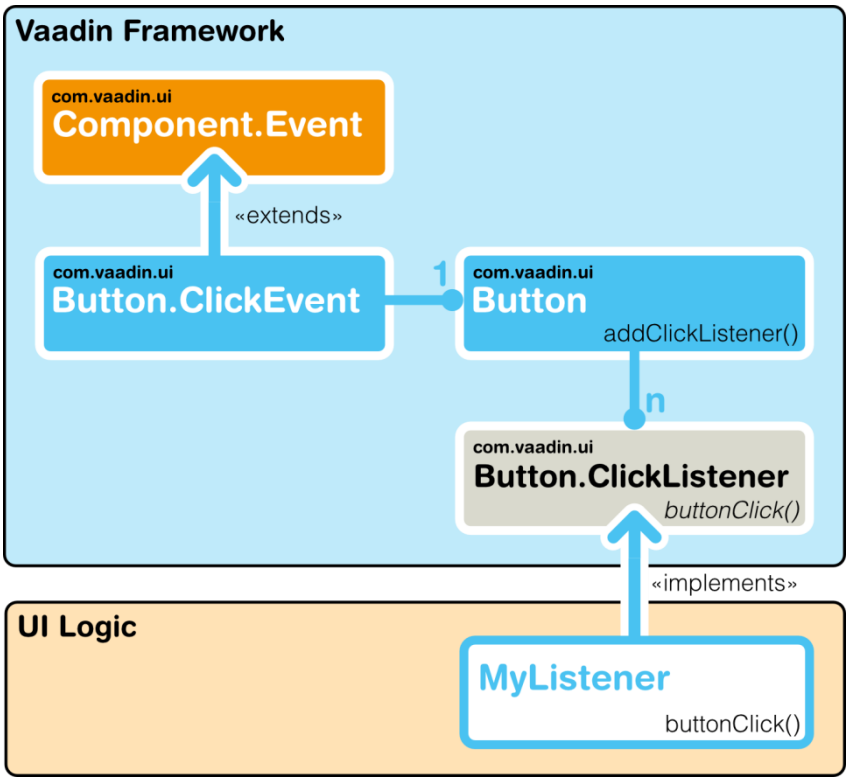


Kuva 13. Vaadin asiakaspuolen kehys

Tarjolla on Googlen ja Vaatimen omat versiot widgeteteistä, jotka muistuttavat paljon toisiaan. Widgetejä pystyy myös luomaan itse jos olemassa olevat eivät kelpaa. (Vaadin 2016f.)

4.5.5 Tapahtumankäsittely

Tapahtumankäsittelyllä tarkoitetaan tilannetta jossa käyttäjä tekee jotakin, kuten painaa nappia on sovelluksen tiedettävä siitä (Kuva 14). Vaadin-sovelluskehyksessä tapahtumienkäsittely tapahtuu tavallisen Java tapahtuman kuuntelijan (Event-listener pattern) mukaisesti. Siinä asetetaan jollekin tietylle komponentille kuuntelija (event listener). Komponenttia kutsuttaessa se laukaisee tapahtuman johon tapahtumankuuntelijat reagoivat. Java 8:ssa tapahtumien käsittely on mahdollista tehdä lambda lausekkeilla, mutta rajaan sen pois tästä työstä, koska sovelma on toteutettu Java 7:llä. (Vaadin 2016e.)



Kuva 14. Diagrammi napin tapahtuman käsittelystä

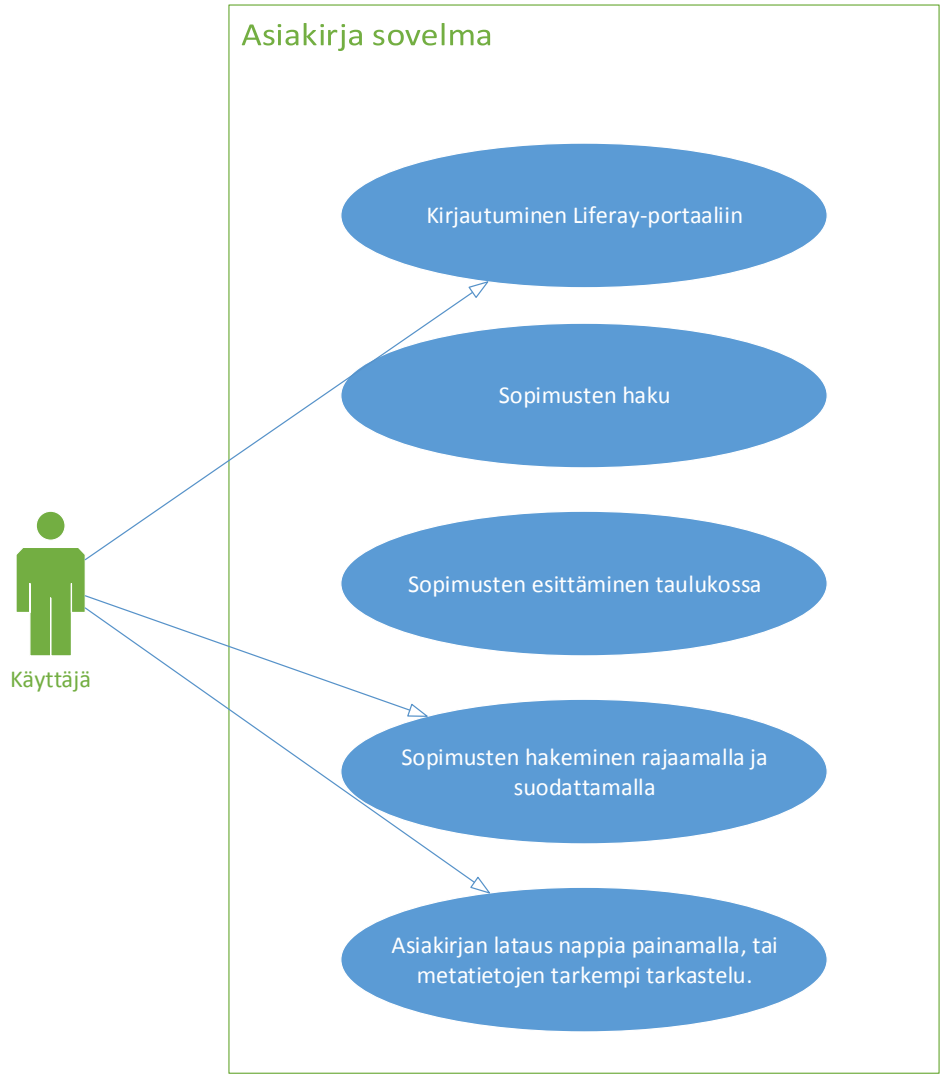
Seuraava esimerkki näyttää miten Vaadin tapahtumankäsittely käytännössä tapahtuu (Kuva 15). Nappiin ”btnDefaultParams”, jonka Designer on jo automaattisesti määrittänyt, liitetään uusi tapahtuman kuuntelija (ClickListener). Kun nappia painetaan, se laukaisee kuuntelija-metodin (buttonClick). Sen sisään voidaan kirjoittaa koodi toiminnolle mikä halutaan napin painalluksen suorittavan.

```
btnDefaultParams.addClickListener(new Button.ClickListener() {
    @Override
    public void buttonClick(ClickEvent event) {
        setDefaultParams();
    }
});
```

Kuva 15. Vaadin tapahtuman käsittelijä.

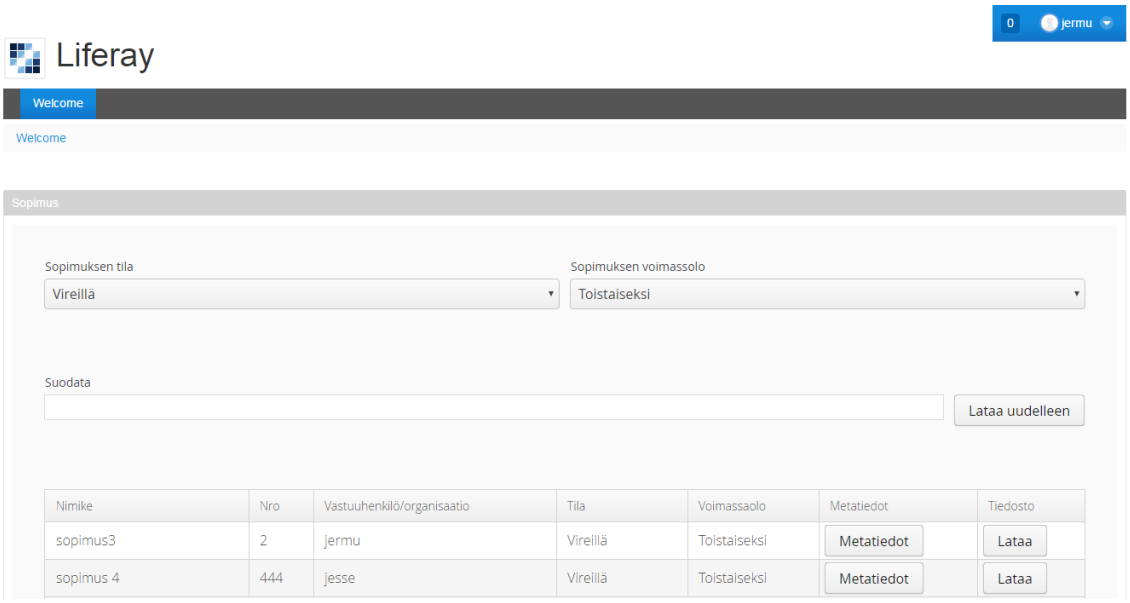
5 ASIAKIRJASOVELMA

Toimeksiantajalle tehtävä portletti toteutetaan Vaadin-sovelluskehyksellä Liferay-portaali alustalle. Portletissa on kaksi näkymää: päänäkymä ja asetukset näkymä. Käyttötapauskaavio kuvaa kuinka tavallinen käyttäjä käyttää sovelmaa (Kuva 16).



Kuva 16. Sovelman käyttötapauskaavio.

Päänäkymä on tavalliselle käyttäjälle ainoa käytettävissä oleva näkymä (Kuva 17). Tauluun ladataan Tweb:stä asianmukaiset sopimukset. Taulu näyttää oleelliset tiedot sopimukset sarakkeissa. Sarakkeiden mukaan on mahdollista järjestää sopimukset aakkosjärjestykseen. Taulusta pystyy lataamaan valitun sopimuksen tiedoston. Lisäksi on mahdollista lukea osa kyseisen sopimuksen dynaamisista metatiedoista. Metatiedot nappi avaa ponnahdusikkunan mistä näkee sopimuksen tärkeimmät dynaamiset metatiedot. Sopimuksia rajataan kahden alasvetovalikon avulla. Ensimmäinen valintalista sisältää vaihtoehdot sopimuksen tilasta ja toinen sopimuksen voimassaolosta. Alasvetovalikosta valitaan halutut arvot tai ne voidaan jättää myös tyhjäksi milloin sovelma hakee kaikki sopimukset voimassaolosta ja tilasta huolimatta. Päänäkymässä on myös erillinen tekstikenttä mihin kirjoittamalla voidaan suodattaa sopimuksia niiden nimikkeen mukaan. WS-rajapinnan kuormitusta vähentääkseen sovellus lataa session ajaksi kaikki sopimukset kerralla muistiin. ”Lataa uudelleen” -napilla sopimukset voidaan kuitenkin ladata tietokannasta uudestaan, jos Tweb-järjestelmään lisätään uusia sopimuksia session aikana, jotka halutaan nähdä sovelluksessa heti. Asetukset nappi on piilotettu tavallisilta käyttäjiltä.



Kuva 17. Portletin etusivu.

Asetukset näkymä on tarkoitettu vain Liferay-portaalin admin käyttäjälle. Siellä voi määrittää portletin yleisiä hakukriteereitä, hakuehtoja metaluokkiin perustuen. Lisäksi taulukon sarakkeiden nimet voi vaihtaa ja palauttaa oletusasetukset.

5.1 Kehitysmenetelmä

Projektissa käytettiin prototyyppimenetelmää. Prototyyppimenetelmässä ohjelmaa kehitetään spiraalimaisesti. Työn eri vaiheet jakaantuvat erilaisille kehille, kuten liiketoimintakerros eri bisneslogiikka ja tietokantoja koskeva kerros. Työ aloitetaan usein käyttöliittymän rakentamisesta, josta työ laajenee muille osa-alueille. (Xtract.)

Prototyyppimallissa näkee jo hyvin varhaisessa vaiheessa millainen sovelluksesta on tulossa. Asiakkaan on helppo tehdä prototyyppiin kehitysehdotuksia, kun sen voi nähdä käytännössä. Prototyyppimallin etuihin kuuluu että siihen on helppoa tehdä muutoksia, kesken kehityksen. Täytyy myös muistaa että ensimmäisen prototyyppi on vielä karkea versio lopputuotteesta. (Xtract.)

Projekti lähtikin liikkeelle käyttöliittymän hahmottelemisesta. Alkuun käyttöliittymästä piti saada näkyviin taulu sopimusten näyttämistä varten ja hakua rajaavat kentät. Tämän jälkeen lähdin pikkuhiljaa rakentamaan WS rajapintaa vasten koodia, joka hakisi asiakirjoja. Ensimmäinen vaihe oli saada haettua kaikki sopimukset, minkä jälkeen tehtävänä oli kirjoittaa koodia, joka erottelee halutut sopimukset parametri kerrallaan. Käytännössä lisäsin ohjelmakoodiin aina yhden metodin kerrallaan, joka kävi tietyn parametrin läpi. Samalla kehitin käyttöliittymää sitä mukaa kun toiminallisuutta tuli lisää.

5.2 Sovelman arkkitehtuuri

Sovelman arkkitehtuuri on jaettu useisiin eri Java luokkiin (Liite 1). MyPorletUI luokassa alustetaan Liferay-portletti ja asetetaan haluttu näkymä. Päänäkymä ja asetukset näkymä rakentuvat kukin kahdesta omasta Java luokasta. View päätteinen luokka sisältää kyseiseen näkymään liittyvän logiikan. Siellä alustetaan käyttöliittymä ja komponenttien toiminnallisuus, kuten tapahtumien kuuntelijat. ViewDesign päätteinen Java luokka on Vaatimen automaattisesti generoima. ViewDesign luokka tulee Vaatimen omalla käyttöliittymärakentajalla tehdystä käyttöliittymästä ja se sisältää Designerissa asetetut käyttöliittymäkomponentit. View luokka laajentaa ViewDesign luokkaa.

Liiketoiminnallinen kerros eli bisneslogiikka sijaitsee WScalls.java nimisessä luokassa. WScalls luokassa sijaitsee Twebin Web Services rajapinnan kutsut. Alkuun alustetaan TwebWSClient, jota käytetään Web Services rajapinnan kanssa (Kuva 18). Käytännössä tässä luokassa tapahtuu sopimusten haku WS-rajapinnan kautta ja sen jälkeen ne järjestellään rajausehtojen mukaan. WScalls Java luokka kutsuu myös instanssin ReadPropertyfile luokasta TwebWSClientin alustamista varten. ReadPropertyfile luokka hakee nimensä mukaisesti properties tiedostosta sinne määritetyt parametrit systemname, systemkey ja wsurl WS-rajapinnan clientin alustusta varten. DocumentData luokkaa käytetään sopimusten tietojen tallentamiseen. Luokasta löytyy sopimuksesta tallennettavat tiedot ja niille Javalle ominaiset get ja set metodit.

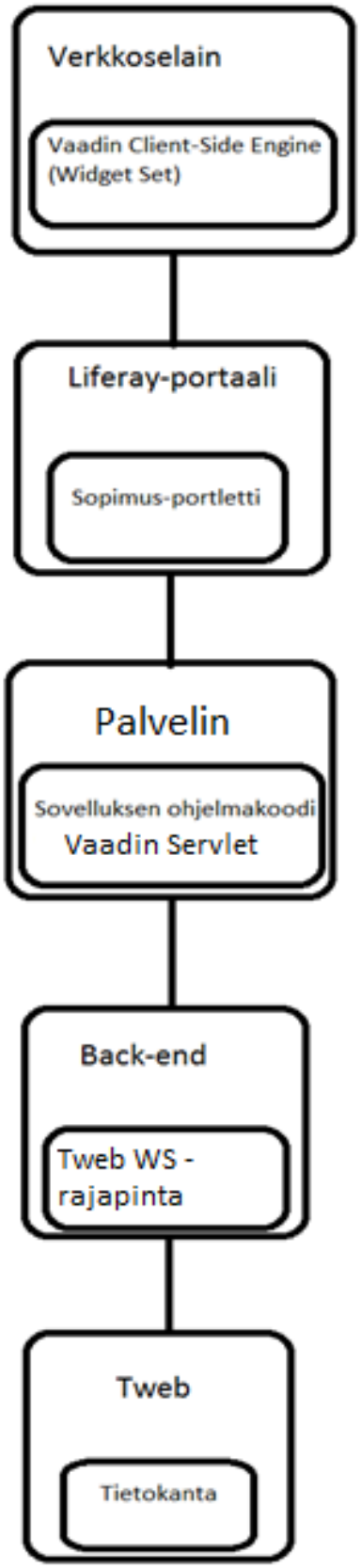
```
public TwebWSClient initializeWSClient(String user) {
    try {
        log.info("Alustetaan WSClientti.");

        TwebWSClient client = new TwebWSClient();
        log.info("WS url: " + prop.getWsurl());
        client.initConnection(prop.getWsurl());

        log.info("WS kirjautumisyritys. sysname: " + prop.getSystemname() + " syskey: " + prop.getSystemkey()
                + " user: " + user);
        client.authenticate(prop.getSystemname(), prop.getSystemkey(), user, "");
        log.info("WS " + client.getTwebWSVersion());
        log.info("WS client " + client.getVersion());
        return client;
    } catch (Exception e) {
        log.error("Ei pystytty alustamaan TwebWSClienttia. " + e.getMessage() + " CausedBy: " + e.getCause(), e);
    }
    return null;
}
```

Kuva 18. TwebWSClientin alustaminen

Sovelman arkkitehtuuri koostuu useista eri osa-alueista (Kuva 19). Organisaatiolla tulee jo olla Tweb asianhallintajärjestelmä käytössään. Lisäosaksi tarvitaan vielä Liferay portaali integraatio. Tämän jälkeen voidaan ottaa käyttöön asiakirjasovelma. Sovelma asennetaan palvelimella sijaitsevaan Liferay-portaaliin. Käyttäjän selaimella sovelma renderöidään Vaadin Client-Side Enginellä. Vaadin ohjelma toimii Java Web palvelimella Vaadin servlettinä. Selain ja palvelin kommunikoivat keskenään AJAX kutsujen välityksellä. WS rajapinta hakee Tweb järjestelmästä asiakirjoja sovelmaa varten.



Kuva 19. Sovelman arkkitehtuuri.

5.3 Twebin web services –rajapinta

Triplanin asian- ja asiakirjanhallinnan Tweb sovellusta varten on rakennettu palvelurajapinta, jota lyhyesti kutsutaan TwebWS. Rajapinnan tarkoitus on tarjota ulkopuolisille sovelluksille mahdollisuus hyödyntää Tweb-järjestelmää. TwebWS rajapinnan kautta saa siis käytännössä haettua ja syötettyä dataa Tweb sovelluksen tietokantaan.

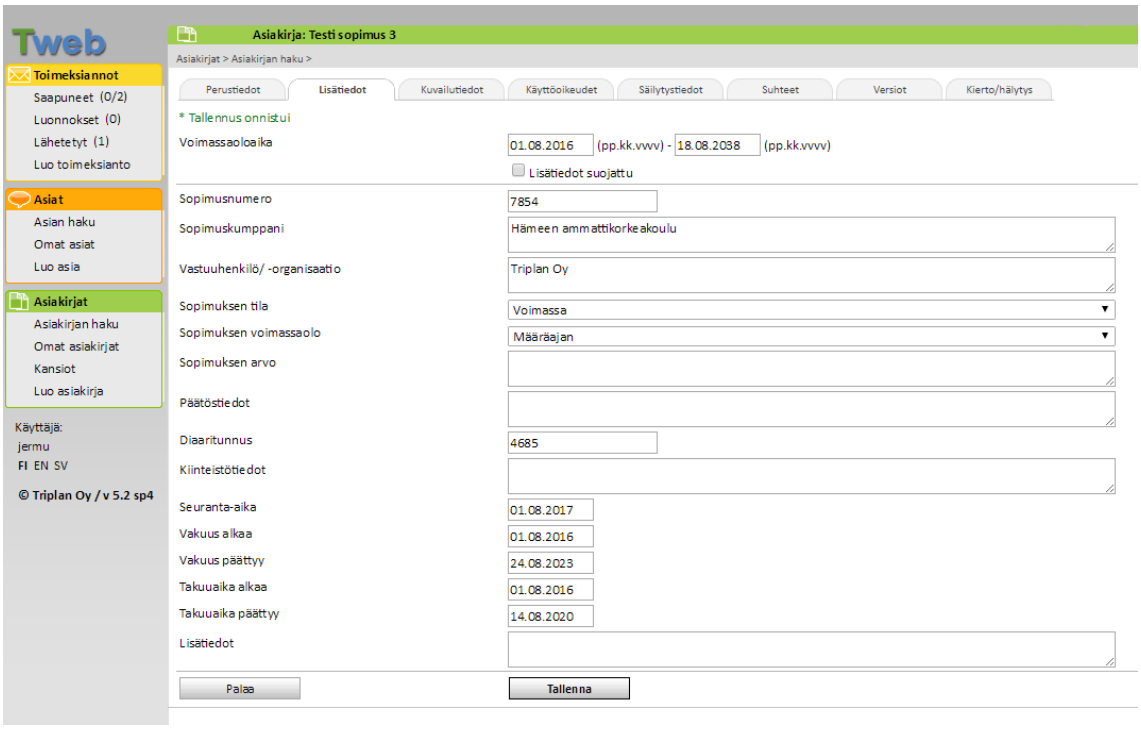
Rajapinta toimii SOAP (Simple Object Access Protocol) tekniikalla. SOAP on XML-pohjainen tietoliikenneprotokolla datan siirtoon. Esimerkkinä client lähettää SOAP kutsun palvelimelle, jossa Web-palvelut ovat käytössä. Tietokannasta haetaan hakuparametrien mukaiset tulokset. Palvelin palauttaa XML-pohjaisen tiedoston clientille, josta data on helppo lukea.

Sovelluksessa käytetään TwebWS-rajapintaa Tweb:n ja sovelluksen väliseen datan siirtoon. Rajapinnan kautta sovellus hakee sopimukset. Toimiakseen sovellus tarvitsee TwebWSClientin. Clientin avulla tehdään haku, jolla haetaan asiakirjat. Hakuehdoksi laitetaan dynaamisen metatiedon arvo.

5.4 Dynaamiset metatiedot

Tweb:ssä kaikilla asiakirjoilla on vakio metatiedot. Niihin määritetään asiakirjojen perustietoja, kuten nimike, laatimisaika ja asiakirjan tiedosto. Asiakirjoja on kuitenkin mahdollista luokitella tarkemmin niille lisättävien dynaamisten metatietojen avulla.

Dynaamiset metatiedot ovat tietyn id luokan alle tallennettuja asiakirjan lisätietoja (Kuva 20). Niiden avulla on helppo luoda aliluokkia asiakirjoille, kuten sopimukset, viranhaltijapäätökset ja niin edelleen. Sovelluksessa haetaan asiakirjoja joille on valittu juuri tällä id:llä asetettu dynaaminen metaluokka. Lisätietoihin asetettujen tietojen mukaan sovellus rajaa näytettäviä sopimuksia.



Kuva 20. Esimerkki sopimuksen dynaamisista metatiedoista.

5.5 Kirjautuminen

Portletin käyttöoikeudet on hoidettu Liferay-portaalin avulla. Päästäkseen portlettiin käsiksi käyttäjällä on aluksi oltava tunnukset oman organisaationsa Triplan-portaaliin, jossa ovat myös muut organisaation käytössä olevat portletit. Portaalin käyttöoikeudet on suoraan sidottu Tweb järjestelmään. Näin pystytään varmistamaan käyttäjä Twebin puolella jotta saadaan oikeat asiakirjat.

Portletti taas hakee Liferay-portaalista sisäänkirjautuneen henkilön käyttäjätunnuksen. Tällä käyttäjätunnuksella WS-rajapinta hakee Tweb järjestelmästä oikeat asiakirjat tarkistettuaan sen oikeudet.

5.6 Asiakirjojen käsittely

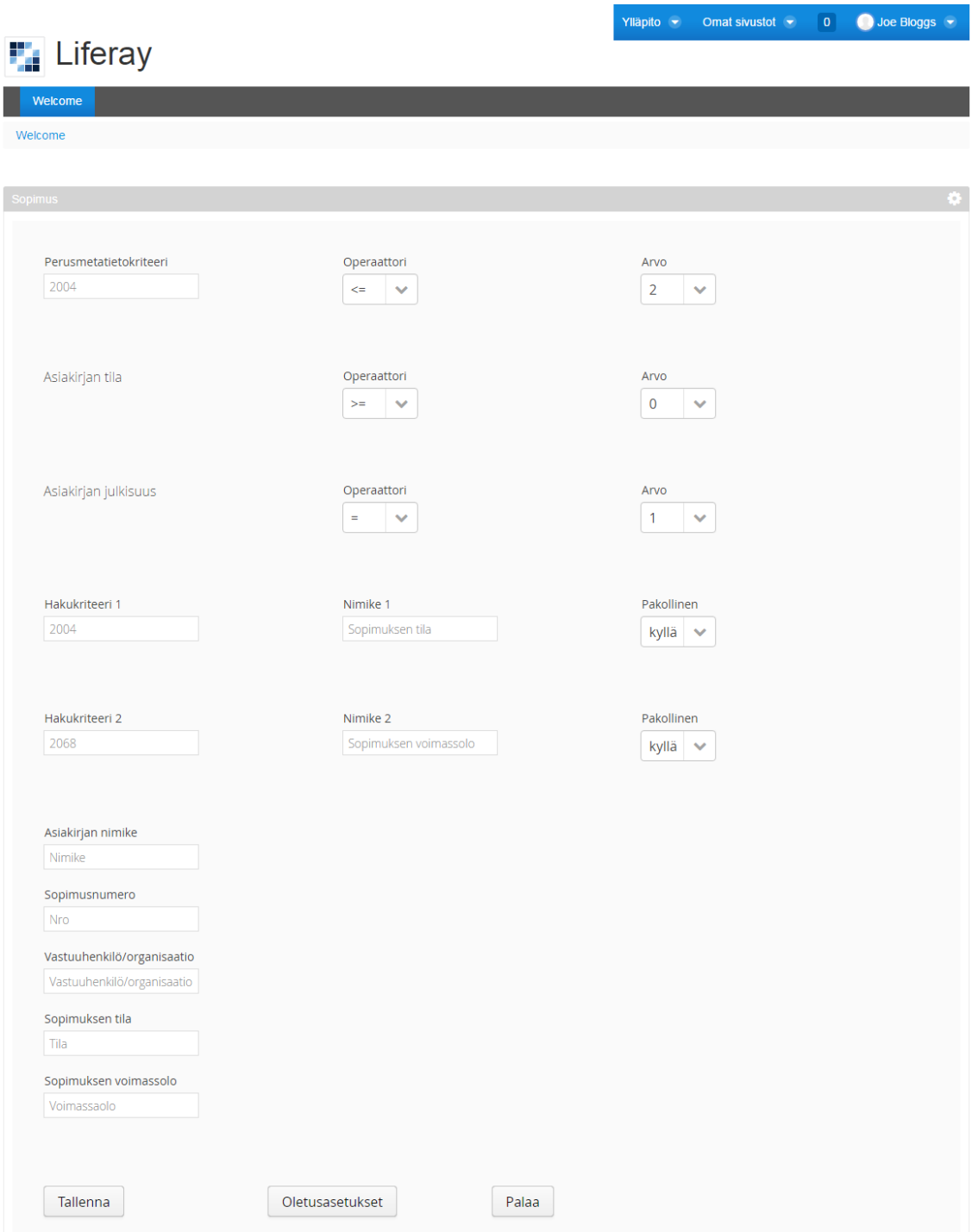
Asiakirjojen hakemiseen käytetään WS-rajapintaa. WScalls nimisessä java luokassa tapahtuu asiakirjojen käsittely. Haku aloitetaan alustamalla WS-client. Alustamisen jälkeen tehdään haku sopimusten dynaamiseen metaluokkaan perustuen. Haku hakee kaikki asiakirjat joilla on sopimuksen dynaaminen metaluokka. Tämän jälkeen käydään sopimukset yksittäin läpi. Aluksi tarkistetaan asiakirjan julkisuusaste ja tila. Jos sopimus läpäisee nämä kriteerit, tallennetaan sen dynaamiset metatiedot listaan. Dynaamisista metatiedoista tarkistetaan, että sopimus vastaa sovelmaan asetettua perusmetatietokriteeriä, joka on tässä tapauksessa sopimuksen tila. Esimerkiksi jos ei haluta antaa käyttäjälle mahdollisuutta nähdä päättyneitä sopimuksia niin ne rajataan pois asetukset sivulla.

Käyttäjän on kuitenkin mahdollista vielä käyttöliittymän kautta tehdä rajauksia näytettäviin sopimuksiin. Käyttöliittymässä tapahtuva sopimusten rajaus kahden alasvetovalikon hakuehdoilla on toteutettu aiemmin luodun listan avulla, johon kaikki hyväksytyt sopimukset on jo tallennettu. WS-rajapinnan kuormituksen vähentämiseksi sopimuksia ei ladata uudestaan saman session aikana. Näin säästetään paljon WS-rajapinnan resursseja, kun ei ladata aina kaikkia sopimuksia uudestaan käyttäjän vaihdettua rajausehtoa. Sovelluksessa on toinen lista johon tallennetaan näytettävät sopimukset. Hakuehtojen muuttuessa käydään läpi kaikki sopimukset sisältävä lista kahden muuttuvan hakukriteerin osalta. Sitten hakuehtoja vastaavat sopimukset lisätään toiseen listaan. Listassa olevat sopimukset näytetään sovelman taulussa. Tapahtuman toistetaan aina hakuehtojen muuttuessa ilman tarvetta tehdä uusia WS-rajapinta kutsuja. Jos saman session aikana kuitenkin halutaan Tweb:n kannasta uudet sopimukset, voidaan ne ladata ”Lataa uudestaan” napin avulla.

5.7 Asetukset

Liferay-portaali asennetaan aina Tweb järjestelmän päälle organisaatio kohtaisesti. Portletin asetukset sivu on tarkoitettu Liferay-portaalin admin käyttäjälle. Siellä on mahdollista muuttaa portletin asetuksia portaalikohtaisesti.

Sarakkeiden nimikkeet on mahdollista räätälöidä organisaatiokohtaisesti (Kuva 21). Portletin oletushaku arvot perusmetatietokriteeri, asiakirjan tila ja asiakirjan julkisuus arvot on myös mahdollista konfiguroida, jos jostain syystä organisaation Tweb järjestelmän id:t eroavat vakioista. Asetukset sivulta löytyy Oletusasetukset nappi, joka palauttaa alkuperäiset asetukset.



Kuva 21. Kuva asetuksista.

5.8 Testaus

Projektia varten pystytettiin kaksi testiympäristöä. Ensimmäinen testiympäristö asennettiin työkoneelleni ja toinen ympäristö asennettiin yrityksen verkossa sijaitsevalle virtuaalikoneelle. Testiympäristön toimintaan saaminen vaati paljon työtä. Ensiksi tarvittiin asentaa Tweb järjestelmä ja sen toimintaa ohjaava Web-Arkki. Vasta sen jälkeen asennettiin itse Liferay-portaali.

Omassa testiympäristössä testasin sovelman toimintaa kehityksen aikana ja kyseistä testiympäristöä tuli aina päivitettyä monta kertaa päivässä. Uuden version ajaminen testiympäristöön kävi melko nopeasti. Liferayn hot deployment toiminnolla sovelman pystyi ajamaan portaaliin suoraan Eclipsestä deploy nappia painamalla. Päivitys ei vaatinut edes Tomcatin uudelleen käynnistystä vaan parin minuutin kulutta sivu tuli ladata uudestaan sovelman päivittämiseksi. Koodissa tapahtuvien muutosten testaaminen oli näin käytännössä helppoa.

6 LOPPUYHTEENVETO

Opinnäytetyön aiheen saadessani en ollut koskaan aikaisemmin kuullut Vaadin sovelluskehyksestä. Aiemmat projektini olivat toteutettu suureksi osaksi tavallisten web-teknologioiden avulla (HTML ja JavaScript). Alkuun pääseminen oli projektin haastavin osuus. Kehitysympäristön kuntoon laitto aiheutti enemmän päänvaivaa, kuin olisi toivonut. Vaadin-sovelluskehyksellä rakennetusta Liferay-portletista löytyi toivottua vähemmän hyvää dokumentaatiota, koska kummatkin yritykset keskittyivät ohjeissaan enemmän omien natiivien ratkaisuiden esittelemiseen. suurin osa käytännön ratkaisuista löytyikin yleisiltä keskustelufoorumeilta.

Alkuun pääsemisen jälkeen kuitenkin Vaadin-sovelluskehyksen tehokkuuden huomasi heti. Vaadin designer editorin avulla sai alustavan käyttöliittymän nopeasti kasaan. Layouttien kanssa oli pientä hienosäätöä, koska niitä piti yhdistellä halutun tuloksen saamiseksi esim. pohjaksi horisontaalinen layout ja sitten siihen piti lisätä vielä vertikaalisia layouteja halutun tuloksen saamiseksi. Vaatimesta löytyi hyvin dokumentaatiota heidän virallisilta ohjesivuilta ”Vaadin Docs”. Vaadin-sovelluskehys sopii hyvin henkilöille joilla on aikaisempaa kokemusta Javasta ja haluavat yksinkertaistaa ja nopeuttaa käyttöliittymän teko prosessia. Projektissa päästiin suurimmaksi osaksi asetettuihin tavoitteisiin. Sovelma saatiin toimimaan halutulla tavalla.

6.1 Jatkokehitys

Sain tuotepäälliköltä jatkokehitystoiveita sovelmaa koskien. Sovimme kuitenkin, että tämän hetkinen tilanne täytti opinnäytetyölle määritetyt kriteerit. Suurin jatkokehitystoive oli jakaa sopimuksia näyttävä taulukko omille sivuilleen. Se tekisi sovelmasta siistimmän näköisen ja helpottaisi sen käyttöä. Yksi mahdollinen tapa toteuttaa tämä olisi esimerkiksi PagedTable nimisen lisäosan käyttö. Vaatimella on nimittäin tarjolla paljon käyttäjien itse tekemiä lisäosia.

Kun projekti aloitettiin keväällä, uusin saatavilla oleva Liferay versio oli 6.2 ja sillä oli virallinen tuki Vaadin portleteille. Kuluvan kesänä aikana kuitenkin julkaistiin uusi Liferay 7.0, jossa asiat muuttuivat. Nyt suositellaan käytettäväksi modulaarista OSGi kehystä, tekemään portleteista kevyempiä yhdistämällä portlettien päällekkäisiä Java paketteja. Eli sovel-

maa joutuu muokkaamaan aika paljon, että sen saa Liferay 7 yhteensopivaksi.

Sovelman oikeuksia olisi myös hyvä jatkossa tarkastella. Tällä hetkellä ainoastaan portaalin adminilla on oikeudet vaihtaa portaalin asetuksia. Tämän oikeuden voisi tulevaisuudessa antaa myös tavalliselle portaalin ylläpitäjälle. Lisäksi asetuksien arvojen numerot olisi hyvä vaihtaa niitä vastaavaan teksti selosteeseen, käytön helpottamiseksi. Nämä ominaisuudet eivät kuitenkaan olleet vielä testivaiheessa tärkeimmät prioriteetit, koska se ei vaikuttanut tavallisen käyttäjän toimintaan. Jälkikäteen ajateltuna yksikkötestien käyttö projektissa olisi ollut suotavaa. Se olisi loppupelissä helpottanut testausta, kun ei olisi tarvinnut aina käsin käydä testaamassa portaalin puolella. Alussa kuitenkin tuntui helpommalta käydä konkreettisesti testaamassa muutoksia käyttöliittymän puolelta.

LÄHTEET

Alejandro 2013. Vaadin 7 UI Design By Example, Birmingham: Packt Publishing Ltd.

Eclipse Foundation. Eclipse documentation. Viitattu 22.6.2016. http://help.eclipse.org/mars/index.jsp?nav=%2F0

GWT Project. Overview. Viitattu 12.7.2016 http://www.gwtproject.org/overview.html

Jaroslav & Kvasnovsky Ondrej 2013. Vaadin 7 Cookbook, Birmingham: Packt Publishing Ltd.

Liferay Developer Network 2016a. WWW-dokumentti. Viitattu 14.6.2016 https://dev.liferay.com/develop/tutorials/-/knowledge_base/6-2/developing-jsp-portlets-using-liferay-mvc

Liferay Develepor Network 2016b. WWW-dokumentti. Viitattu 13.9.2016 https://dev.liferay.com/discover/portal/-/knowledge_base/6-2/roles-and-permissions

Maven. What is Maven. WWW-dokumentti. Viitattu 4.8.2016. https://maven.apache.org/what-is-maven.html

Mozilla developer network. Ajax. Getting started. WWW-dokumentti Viitattu 20.7.2016. https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started

Oracle. Java Servlet Technology Overview. WWW-dokumentti. Viitattu 28.7.2016. http://www.oracle.com/technetwork/java/overview-137084.html

Richard Jr. 2012. Liferay In Action. Shelter Island, NY: Manning Publications Co.

The Apache Software Foundation. Apache Tomcat 2016. WWW-dokumentti. Viitattu 8.7.2016 http://tomcat.apache.org/

Vaadin 2016a. Vaadin Introduction. Overview. WWW-dokumentti. Viitattu 16.6.2016 https://vaadin.com/docs/-/part/framework/introduction/intro-overview.html

Vaadin 2016b. Vaadin Plug-in for Eclipse. WWW-dokumentti. Viitattu 28.6.2016. https://vaadin.com/eclipse#visual-designer

Vaadin 2016c. Vaadin Architecture. Overview. WWW-dokumentti. Viitattu 17.6.2016. https://vaadin.com/docs/-/part/framework/architecture/architecture-overview.html

Vaadin 2016d. Vaadin Designer. Overview. WWW-dokumentti. Viitattu 13.7.2016. https://vaadin.com/docs/-/part/designer/designer-overview.html

Vaadin 2016e. Vaadin Framework. Vaadin server-side applications. WWW-dokumentti. Viitattu 13.7.2016. https://vaadin.com/docs/-/part/framework/application/application-events.html

Vaadin 2016f. Vaadin Framework. Vaadin Architecture. Client-side engine. WWW-dokumentti. Viitattu 25.7.2016. https://vaadin.com/docs/-/part/framework/architecture/architecture-client-side.html

Vaadin 2016e. Vaadin Framework. Vaadin Architecture. Technological Background. WWW-dokumentti. Viitattu 28.7.2016. https://vaadin.com/docs/-/part/framework/architecture/architecture-technology.html

Valkeapää, H. 2014. Liferay palveluportaalin alustana. Karelia ammattikorkeakoulu. Tietojenkäsittelyn koulutusohjelma. Opinnäytetyö.

Wikipedia 2016a. WWW-dokumentti. Viitattu 7.6.2016. https://en.wikipedia.org/wiki/Liferay

Wikipedia 2016b. WWW-dokumentti. Viitattu 17.6.2016. https://en.wikipedia.org/wiki/Rich_Internet_application

Xtract 2016. WWW-dokumentti. Viitattu 7.10.2016. http://xtract.fi/prototyyppimenetelma-ohjelmistotuotannon-vaihejakomallina

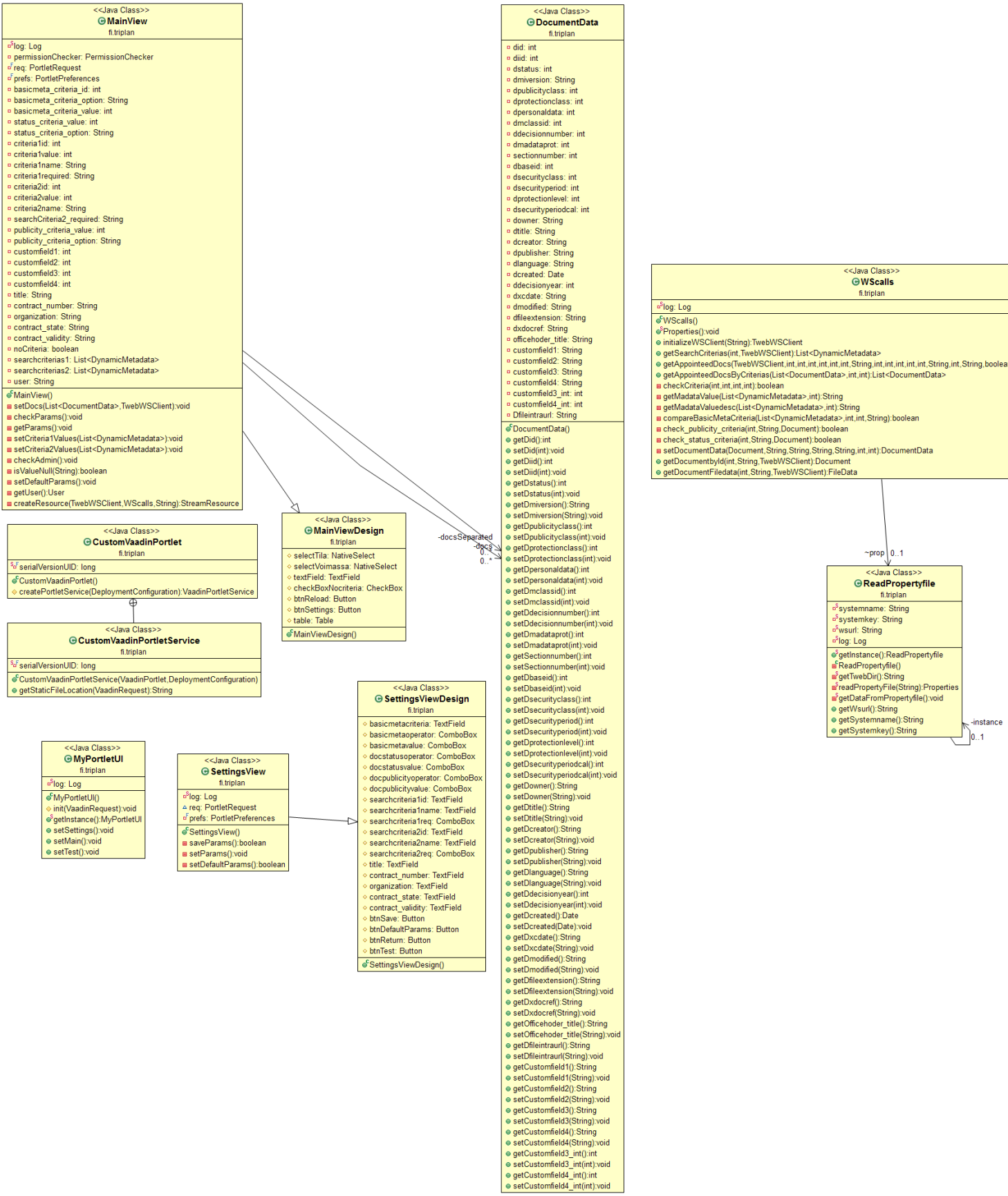
Kuva 12: Vaadin. Vaadin Architecture Overview. Viitattu 17.6.2016. https://vaadin.com/docs/-/part/framework/architecture/architecture-overview.html

Kuva 13: Vaadin. Vaadin Architecture Client-Side Engine. Viitattu 25.7.2016.

https://vaadin.com/docs/-/part/framework/architecture/architecture-client-side.html

Kuva 14: Vaadin. Vaadin Architecture - Events and Listeners. Viitattu 28.7.2016. https://vaadin.com/docs/-/part/framework/architecture/architecture-events.html

Liite 1


<<Java Class>>
MainView
fi.triplan
log: Log
permissionChecker: PermissionChecker
req: PortletRequest
prefs: PortletPreferences
basicmeta_criteria_id: int
basicmeta_criteria_option: String
basicmeta_criteria_value: int
status_criteria_value: int
status_criteria_option: String
criteria1id: int
criteria1value: int
criteria1name: String
criteria1required: String
criteria2id: int
criteria2value: int
criteria2name: String
searchCriteria2_required: String
publicity_criteria_value: int
publicity_criteria_option: String
customfield1: int
customfield2: int
customfield3: int
customfield4: int
title: String
contract_number: String
organization: String
contract_state: String
contract_validity: String
noCriteria: boolean
searchcriterias1: List<DynamicMetadata>
searchcriterias2: List<DynamicMetadata>
user: String
MainView()
setDocs(List<DocumentData>,TwebWSClient):void
checkParams():void
getParams():void
setCriteria1Values(List<DynamicMetadata>):void
setCriteria2Values(List<DynamicMetadata>):void
checkAdmin():void
isValueNull(String):boolean
setDefaultParams():void
getUser():User
createResource(TwebWSClient,WScalls,String):StreamResource
<<Java Class>>
DocumentData
fi.triplan
did: int
diid: int
dstatus: int
dmiversion: String
dpublicityclass: int
dprotectionclass: int
dpersonaldata: int
dmclassid: int
ddecisionnumber: int
dmadataprot: int
sectionnumber: int
dbaseid: int
dsecurityclass: int
dsecurityperiod: int
dprotectionlevel: int
dsecurityperiodcal: int
downer: String
dtitle: String
dcreator: String
dpublisher: String
dlanguage: String
dcreated: Date
ddecisionyear: int
dxcdate: String
dmodified: String
dfileextension: String
dxdocref: String
officehoder_title: String
customfield1: String
customfield2: String
customfield3: String
customfield4: String
customfield3_int: int
customfield4_int: int
Dfileintraurl: String
DocumentData()
getDid():int
setDid(int):void
getDiid():int
setDiid(int):void
getDstatus():int
setDstatus(int):void
getDmiversion():String
setDmiversion(String):void
getDpublicityclass():int
setDpublicityclass(int):void
getDprotectionclass():int
setDprotectionclass(int):void
getDpersonaldata():int
setDpersonaldata(int):void
getDmclassid():int
setDmclassid(int):void
getDdecisionnumber():int
setDdecisionnumber(int):void
getDmadataprot():int
setDmadataprot(int):void
getSectionnumber():int
setSectionnumber(int):void
getDbaseid():int
setDbaseid(int):void
getDsecurityclass():int
setDsecurityclass(int):void
getDsecurityperiod():int
setDsecurityperiod(int):void
getDprotectionlevel():int
setDprotectionlevel(int):void
getDsecurityperiodcal():int
setDsecurityperiodcal(int):void
getDowner():String
setDowner(String):void
getDtitle():String
setDtitle(String):void
getDcreator():String
setDcreator(String):void
getDpublisher():String
setDpublisher(String):void
getDlanguage():String
setDlanguage(String):void
getDdecisionyear():int
setDdecisionyear(int):void
getDcreated():Date
setDcreated(Date):void
getDxcdate():String
setDxcdate(String):void
getDmodified():String
setDmodified(String):void
getDfileextension():String
setDfileextension(String):void
getDxdocref():String
setDxdocref(String):void
getOfficehoder_title():String
setOfficehoder_title(String):void
getDfileintraurl():String
setDfileintraurl(String):void
getCustomfield1():String
setCustomfield1(String):void
getCustomfield2():String
setCustomfield2(String):void
getCustomfield3():String
setCustomfield3(String):void
getCustomfield4():String
setCustomfield4(String):void
getCustomfield3_int():int
setCustomfield3_int(int):void
getCustomfield4_int():int
setCustomfield4_int(int):void
<<Java Class>>
WScalls
fi.triplan
log: Log
WScalls()
Properties():void
initializeWSClient(String):TwebWSClient
getSearchCriterias(int,TwebWSClient):List<DynamicMetadata>
getAppointeedDocs(TwebWSClient,int,int,int,int,int,int,String,int,int,int,int,int,String,int,String,boolean):List<DocumentData>
getAppointeedDocsByCriterias(List<DocumentData>,int,int):List<DocumentData>
checkCriteria(int,int,int,int):boolean
getMadataValue(List<DynamicMetadata>,int):String
getMadataValuedesc(List<DynamicMetadata>,int):String
compareBasicMetaCriteria(List<DynamicMetadata>,int,int,String):boolean
check_publicity_criteria(int,String,Document):boolean
check_status_criteria(int,String,Document):boolean
setDocumentData(Document,String,String,String,String,int,int):DocumentData
getDocumentbyId(int,String,TwebWSClient):Document
getDocumentFiledata(int,String,TwebWSClient):FileData
<<Java Class>>
MainViewDesign
fi.triplan
selectTila: NativeSelect
selectVoimassa: NativeSelect
textField: TextField
checkBoxNocriteria: CheckBox
btnReload: Button
btnSettings: Button
table: Table
MainViewDesign()
-docsSeparated
-docs
0..*
<<Java Class>>
CustomVaadinPortlet
fi.triplan
serialVersionUID: long
CustomVaadinPortlet()
createPortletService(DeploymentConfiguration):VaadinPortletService
<<Java Class>>
CustomVaadinPortletService
fi.triplan
serialVersionUID: long
CustomVaadinPortletService(VaadinPortlet,DeploymentConfiguration)
getStaticFileLocation(VaadinRequest):String
~prop
0..1
<<Java Class>>
ReadPropertyfile
fi.triplan
systemname: String
systemkey: String
wsurl: String
log: Log
getInstance():ReadPropertyfile
ReadPropertyfile()
getTwebDir():String
readPropertyFile(String):Properties
getDataFromPropertyfile():void
getWsurl():String
getSystemname():String
getSystemkey():String
-instance
0..1
<<Java Class>>
SettingsViewDesign
fi.triplan
basicmetacriteria: TextField
basicmetaoperator: ComboBox
basicmetavalue: ComboBox
docstatusoperator: ComboBox
docstatusvalue: ComboBox
docpublicityoperator: ComboBox
docpublicityvalue: ComboBox
searchcriteria1id: TextField
searchcriteria1name: TextField
searchcriteria1req: ComboBox
searchcriteria2id: TextField
searchcriteria2name: TextField
searchcriteria2req: ComboBox
title: TextField
contract_number: TextField
organization: TextField
contract_state: TextField
contract_validity: TextField
btnSave: Button
btnDefaultParams: Button
btnReturn: Button
btnTest: Button
SettingsViewDesign()
<<Java Class>>
MyPortletUI
fi.triplan
log: Log
MyPortletUI()
init(VaadinRequest):void
getInstance():MyPortletUI
setSettings():void
setMain():void
setTest():void
<<Java Class>>
SettingsView
fi.triplan
log: Log
req: PortletRequest
prefs: PortletPreferences
SettingsView()
saveParams():boolean
setParams():void
setDefaultParams():boolean