LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

Web-sovelluskehitys Vaadin-sovelluskehyksellä

Tamminen, Vesa

2015 Leppävaara

Laurea-ammattikorkeakoulu
Laurea Leppävaara

Web-sovelluskehitys Vaadin-sovelluskehyksellä

Tamminen Vesa
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Helmikuu, 2015

Laurea-ammattikorkeakoulu
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

Tamminen, Vesa

Web-sovelluskehitys Vaadin-sovelluskehyksellä

Vuosi 2015 Sivumäärä 37

Tämän toiminnallisen opinnäytetyön tavoitteena oli kehittää toimeksiantajan asiakasprojektiin rajattu kokonaisuus web-sovelluksesta, joka toteutettiin käyttämällä suomalaista Vaadin-sovelluskehystä. Opinnäytetyönä toteutettiin yksi keskeinen näkymä kyseisessä web-sovelluksessa sekä ulkoasun että back-end –toiminnallisuuden osalta.

Toimeksiantajayritys opinnäytetyölle oli Elisa Appelsiini Oy. Appelsiini Finland Oy perustettiin vuonna 1999. Elisa osti Appelsiinin vuonna 2010. Yrityksen nimi muuttui Elisa Appelsiiniksi vuonna 2014.

Toimeksiantaja toivoi, että opinnäytetyö olisi konkreettista kehitystyötä asiakasprojektiin. Toimeksiannossa luotava web-sovelluksen osa meni tuotantoon vuoden 2014 lopussa.

Opinnäytetyössä havainnollistettiin projektissa kehitettyä ominaisuutta toimeksiannon tarkan kuvauksen lisäksi kuvakaappauksilla, Vaadin-komponenttien avaamisella sekä varsinaisella Java-ohjelmakoodilla.

Opinnäytetyön teoriaosuudessa tarkasteltiin myös Vaadin-sovelluskehyksen rakennetta ja käyttöä. Sovelluskehystä avaamalla pyritään taustoittamaan sitä, miksi erityisesti Vaadin-sovelluskehys sopii tällaiseen projektiin. Tämän lisäksi avattiin myös muita Java EE –projektin web-sovelluskehitykseen liittyviä moduuleja, joiden toiminnasta opinnäytetyöprojekti oli riippuvainen.

Asiasanat: Vaadin, sovelluskehys, web-sovellus, ohjelmistokehitys, Java

Laurea University of Applied Sciences Abstract
Laurea Leppävaara
Bachelor's Degree Programme in Business Information Technology

Tamminen, Vesa

Web Application Development with Vaadin Framework

Year 2015 Pages 37

The goal of the operational part of this thesis was to design and develop a web application view with some essential user interaction features for a company. The functional view included the layout, the user interface components and the back-end functionality. The web application was developed with a Finnish framework called Vaadin for Java Enterprise programming language.

The web application was made for a client of the commissioner of this thesis. The commissioner wished that the thesis would be actual software development for their client project thus the nature of this thesis is functional. The developed end-product of this thesis went into production for the commissioner's client to use just at the end of the year 2014.

The commissioner of this thesis was Elisa Appelsiini. Appelsiini Finland Oy was founded in the year 1999. A big telecommunications company Elisa bought Appelsiini in 2010 and in 2014 the name of the client company was changed to Elisa Appelsiini.

In addition to demonstrating the actual view, the purpose was also to explain the different Vaadin-components used in the view by thorough explanation, by screenshots and by actual Java code.

In the theory section of the thesis Vaadin framework was examined in more detail and the criteria of using it was demonstrated. The other modules of a Java EE project used in the client project were also clarified.

Keywords: Vaadin, framework, web application, software development, Java

Sisällys

1 Johdanto

Tämän opinnäytetyön tavoitteena oli kehittää Java-ohjelmointikielen Vaadin-sovelluskehyksellä keskeinen ominaisuus toimeksiantajayrityksen asiakasprojektiin, joka on selaimella käytettävä, asiakkaan henkilöstölle sekä heidän asiakkailleen tarkoitettu web-sovellus.

Opinnäytetyö on toiminnallinen. Toiminnallisuus muodostuu sovelluskehityksestä toimeksiantajayrityksen olemassa olevassa asiakasprojektissa. Työ koostui konkreettisesta Java-ohjelmakoodista, jolla asiakkaan haluama toiminnallisuus saavutettiin sekä kehitysympäristön ja –ratkaisujen taustoittamisesta. Taustoittaminen perustui kirjallisiin lähteisiin sekä omaan kokemusperäiseen tietopohjaan ja kokemuksiin. Keskeinen tavoite oli tarkastella ja esitellä eritoten Vaadin-sovelluskehystä tämän kehitystyön puitteissa ja yleisellä tasolla. Sen lisäksi tarkasteltiin lyhyesti Java-pohjaiseen web-sovelluskehitykseen ja tähän projektiin liittyviä teknisiä kokonaisuuksia.

Vaadin on kotimainen Java-sovelluskehys ja mahdollisesti yksi tekijä Java-pohjaisen web-sovelluskehityksen suosion kasvamisessa. Lyhyesti ilmaistuna sillä pyritään selkiyttämään ja yksinkertaistamaan web-sovelluksissa käyttöliittymän ja liiketoimintalogiikan toimintaa niin, että molemmat voidaan ohjelmoida lähes kokonaan pelkällä Java-ohjelmointikielellä.

Toimeksiannossa asiakkaalle tarkoitettuun web-sovellukseen tuli kehittää keskeinen toiminto, jossa asiakasyrityksen henkilöstö voi luoda vapaamuotoisia, tiettyyn tilattuun palveluun tai toimeksiantoon liittyviä tehtäviä yrityksen henkilöstölle yksilöllisesti tai kokonaiselle käyttäjäryhmälle. Ominaisuus voi kuulostaa näin ilmaistuna pieneltä ja rajatulta, mutta lopulta siihen sisältyi laaja kirjo Vaadin-sovelluskehyksen ominaisuuksien sekä muiden Java EE –ohjelmistokehitykseen liittyvien teknisten kokonaisuuksien käyttämistä. Näitä niin kutsuttuja Java EE –projektin moduuleja avattiin tässä opinnäytetyössä käytännön ja teorian kautta. Tähän ryhmään kuuluu esimerkiksi sellaisia tyypillisiä yrityssovelluskehitykseen liittyviä ohjelmia ja Java-kirjastoja kuten Maven, WildFly, Hibernate, JPA ja EJB.

Tässä opinnäytetyössä ei käytetty konkreettisia tutkimusmenetelmiä, kuten haastatteluja, koska sellaisten käyttö ei olisi mitenkään edesauttanut tavoitetoiminnallisuuden saavuttamista. Tarvittava tieto halutun ominaisuuden yksityiskohdista tuli suoraan ja tarkasti määriteltynä ohjelmointiryhmän johtajalta eli ohjelmistoarkkitehdilta. Nämä yksityiskohdat oli sovittu ja käyty läpi aiemmin asiakkaan kanssa. Työn luonne oli siis ennalta määritelty ja suorittava, mutta itse ohjelmakoodin luomiseen annettiin suhteellisen vapaat kädet.

2 Tavoitteet ja tehtävät

Toimeksiantajayritys opinnäytetyölle oli Elisa-Appelsiini Oy. Yritys perustettiin alun perin Appelsiini Finland Oy –nimisenä vuonna 1999. Elisa osti yrityksen vuonna 2010 ja yrityksen nimi muuttui vuonna 2014 Elisa Appelsiini Oy:ksi. Loppuvuodesta myös yrityksen logo muuttui uuden nimen ja ilmeen mukaiseksi.

Kuva 1: Elisa Appelsiini Oy:n uusi logo

Yritys tuottaa erilaisia IT-palveluja pääasiassa keskikokoisille yrityksille. Palveluihin kuuluvat pääasiassa perustietotekniikka, sovelluspalvelut, työn tuottavuus –palvelut, konsultointi- sekä pilvipalvelut.

Yritys on yksi Suomen nopeimmin kasvavista IT-yrityksistä. Elisa-Appelsiini kasvaa edelleen nopealla ja kiihtyvällä tahdilla vastoin alan yleistä trendiä, joka synkän taloustilanteen ja epävarmojen ulkopoliittisten ja taloudellisten tulevaisuudennäkymien johdosta on

laskujohteinen. Toimeksiantajayrityksen asiakkaan nimeä ei tässä opinnäytetyössä saatu tietoturvasyistä mainita.

Tähän opinnäytetyöhön rajautuneen kehitystyön tarkoitus oli tuottaa Java-ohjelmointikielen Vaadin-sovelluskehyksen erilaisia komponentteja sekä ominaisuuksia käyttäen asiakasyritykselle tuotettavaan web-sovellukseen yksi, suhteellisen keskeinen ominaisuus.

Toimiakseen tämä ominaisuus vaatii tiedonhakua sovelluksen taustalla toimivasta tietokannasta, käyttöliittymäkomponentteihin syötetyn tiedon käsittelyä, usean erityyppisen käyttöliittymäkomponentin yhteistoimintaa sekä kyseisen ominaisuuden käyttöliittymän tapahtumien vaikuttamista web-sovelluksen muihin näkymiin sekä tietokantaan niin, että web-sovelluksen liiketoimintalogiikka pysyy kauttaaltaan eheänä.

Toimeksiantotehtävä annettiin nimekkeellä ”yleisen tehtävän lisääminen asiakasyrityksen henkilöstölle.” Toimeksiantajan asiakkaalle räätälöidyn web-sovelluksen keskeisimpiä käyttöliittymänäkymiä on Tehtävät-sivu, jolla asiakasyrityksen työntekijät näkevät heille ilmaantuneet uudet tehtävät ja voivat niiden perusteella priorisoida ja hoitaa joka päiväisiä työtehtäviään. Osa tälle sivulle päivittyvistä tehtävistä saa tehtäväkuvauksensa, tehtäväään liittyvät asiakirjat sekä tehtävän vastuukäyttäjän tai -käyttäjäryhmän automaattisesti tehtävän tyypistä riippuen. Tehtävätyyppejä voi olla esimerkiksi ”Hyväksy asiakasta koskeva päätös” tai ”Postita loppulausunto asiakkaalle”. Edellä mainitut tehtävät muodostuvat automaattisesti tehtävätyypille asetettujen parametrien mukaisesti.

Yleinen tehtävä tarkoittaa sitä, että tehtävän kuvaus, tehtäväkohtaiset asiakirjat ja vastuuhenkilö tai –ryhmä ovat vapaavalintaisia, jolloin ne eivät siis muodostu automaattisesti vaan kaikki tehtävän kentät luodaan manuaalisesti näkymässä, joka tässä toimeksiannossa tuli toteuttaa. Tämä tapa antaa tehtäviä sopii tilanteisiin, jossa tehtävä ei sovi mihinkään ennaltamääriteltyyn tehtävätyyppiin. Voidaan siis luoda tehtävä jollekin käyttäjälle tai käyttäjäryhmälle, liittää siihen mikä tahansa asiakastoimeksiantoon aiemmin liitetty asiakirja ja kirjoittaa tehtävälle vapaa kuvaus, jolloin kaikki tuo tieto näkyy vastaanottajakäyttäjien Tehtävät-näkymässä yhteen tehtäväelementtiin sisältyvinä kohtina. Tehtäväelementissä myös näkyy mihin toimeksiantoon ja mahdollisesti mihin toimeksiannolle lisättyyn palveluun yleinen tehtävä liittyy.

Kokonaisuudessaan asiakasyrityksen edustajan vaatimus, jonka perusteella toimeksianto määriteltiin, oli seuraava: ”Käyttäjänä haluan lisätä toimeksiantoon tai jollekin sille tilatulle palvelulle yleisen tehtävän.” Näkymän vaatimusmäärittelyssä listattiin joukko toimintoja, joita web-sovelluksen käyttäjien pitäisi voida tässä näkymässä suorittaa. Näitä olivat yleisen tehtävän lisääminen toimeksiannolle, yleisen tehtävän lisääminen liittyen tiettyyn

toimeksiannon tilattuun palveluun, asiakirjojen liittäminen yleiseen tehtävään drag & drop –ominaisuutta käyttäen, vastuuhenkilön valitseminen ja asettaminen tehtävälle, vastuuroolin valitseminen ja asettaminen tehtävälle, kuvauksen kirjoittaminen tehtävälle ja listan viimeisenä kohtana kaikkien näkymässä luotujen tehtävien ilmestyminen yleinen tehtävä –otsikolla Tehtävät-näkymään niin, että vastuuhenkilö voi itselleen osoitetusta tehtävästä tarkastella yhdellä kertaa sen kuvausta ja nähdä listan siihen liitetyistä asiakirjoista sekä tilatun palvelun, johon tehtävä on mahdollisesti liitetty.

Ominaisuus oli luonteeltaan sellainen, että se ei ollut asiakkaan liiketoiminnan eikä web-sovelluksen toiminnan kannalta kriittinen, mutta kuitenkin tärkeä, mahdollisimman pian kriittisten ominaisuuksien jälkeen kehitettävä toiminnallisuus. Projektitiimin ja asiakkaan kanssa keskusteltiin ja sovittiin, että tämä ominaisuus voisi sopia työharjoittelijalle opinnäytetyön aiheeksi ja kehitettäväksi erillään sillä aikaa, kun muu, kokeneempi osa tiimiä keskittyisi asiakasyrityksen liiketoiminnan kannalta kriittisempien ominaisuuksien ja korjauksien kehittämiseen.

Toimeksiantajayritys Elisa-Appelsiini Oy ilmaisi hyvin aikaisessa vaiheessa, että ideaalitilanteessa opinnäytetyö olisi konkreettista ohjelmointia asiakasprojektiin. Tämä oli merkittävin yksittäinen tekijä toimeksiannon aihetta toimeksiantajan kanssa mietittäessä. Asiakkaalta saatiin hyväksyntä opinnäytetyön aiheelle keskustelemalla toimeksiantajayrityksen, tämän projektin projektipäällikön ja asiakasyrityksen edustajan kanssa.

3 Tutkimus

Kuten mainittua, varsinaisia tutkimusmenetelmiä ei toimeksiannon tavoitteiden saavuttamiseksi ollut mielekästä käyttää, joten sellaisten sisällyttäminen työhön olisi ollut jopa haitallista projektin etenemiselle ja tarpeetonta toimeksiantajan ja asiakasyrityksen henkilöstön kuormittamista.

Tutkimuksellista otetta vaativaksi työn osa-alueeksi voidaan toki yleisellä tasolla luonnehtia haluttuun ominaisuuteen liittyvien teknisten kokonaisuuksien opiskeleminen kirjallisesta materiaalista ja sovelluskehitysryhmän muiden työntekijöiden luomasta ohjelmakoodista. Ominaisuuden ulkoisen ja toiminnallisen rakenteen yhdenmukaistaminen web-sovelluksen muiden osien kanssa myös vaati useita työpäiviä vaativaa havainnointia käyttöliittymä- ja ohjelmakooditasolla, jota toki tehtiin koko projektissa työskentelyn ajan jo ennen varsinaista toimeksiannon työprosessin aloittamista, joten tähän ei tarvinnut käyttää toimeksiannon alussa kovin paljoa aikaa.

Ominaisuuden erilaisista yksityiskohdista piti luonnollisesti olla jatkuvassa keskusteluyhteydessä ohjelmointitiimin kanssa. Projektitiimin työskentelykäytännön takia asiakkaaseen oli kerrallaan yhteydessä vain yksi henkilö, projektin sovelluskehitystä ohjaava ohjelmistoarkkitehti. Toisinaan tuli tarvetta muuttaa jotain jo toteutettua yksityiskohtaa tai haluttiin lisätä tai poistaa vielä piirteitä, joita ei alkuperäisessä toimeksiantomäärityksessä ollut mainittu. Tällainen oli esimerkiksi käyttöliittymäkomponentin luominen tehtävän vastuuryhmän valitsemiseksi pelkän yksittäisen henkilön lisäksi.

4 Toteutus vaiheittain

Ennen toimeksiantoa opinnäytetyön tekijä oli työskennellyt asiakasprojektin parissa hieman yli kolme kuukautta. Projektin ohjelmakoodin rakenne oli siis tullut monilta osin jo tutuksi. Kun toimeksianto annettiin, niin oli jonkinlainen käsitys jo siitä miten ja missä Java-luokissa kyseisen toiminnallisuuden front-endiin ja back-endiin liittyviä ominaisuuksia voi luoda ja muokata.

Haluttuun näkymään liittyi kuitenkin ominaisuuksia, jotka vaativat perusteellista uuden opiskelua. Näistä esimerkkinä drag & drop –ominaisuus, jossa asiakirjoja voi liikuttaa tietokoneen hiirellä asiakirjataulusta alueelle, jota kutsutaan dropzoneksi. Toistaiseksi tällainen ominaisuus oli toteutettu vain hieman erilaiseen näkymään ja erilaisin vaatimusmäärittelyin ja sen oli luonut hyvin kokenut konsultti, joka halusi käyttää Javan uusimman version luokkia ja metodeja, mikä aiheutti huomattavaa tarvetta itseopiskelulle, jotta vastaava ominaisuus saataisiin toimimaan toimeksiannossa työstettävässä näkymässä.

Ensimmäinen työvaihe ennen ohjelmakoodin kirjoittamisen aloittamista oli siis ominaisuuteen vaadittavan koodin rakenteen kartoittaminen. Oli tutkittava projektissa jo olemassaolevia rakenteita perusteellisesti, koska jokaiseen näkymään ja ominaisuuteen halutaan yhteneväisyyttä niin paljon kuin mahdollista. Tämä mahdollistaa sujuvan sovellushallinnan jatkossa sekä sulavan liiketoimintalogiikan läpi koko web-sovelluksen.

Toiseksi tuli kehittää hahmotelma näkymän ulkoasusta. Mitä painikkeita käyttäen ja mistä käsin käyttäjä pääsisi uuteen näkymään, miten eri komponentit tulisi asetella ikkunaan ja mitä muita mahdollisia lisäominasuuksia käyttöliittymäpuolelle mahdollisesti tulisi. Tällaisia ovat esimerkiksi varmistusdialogit eri toimintojen kohdalla sekä mahdollinen aakkosjärjestys vastuuhenkilöiden ja –roolien valinnassa.

Kahteen ensimmäiseen työvaiheeseen resursoitiin kaksi työpäivää. Tämän jälkeen alkoi ohjelmakoodin varsinainen kirjoittaminen. Ohjelmakoodin kirjoittamista ei suoraan ohjattu projektitiimin johdon puolesta, koska haluttiin myös nähdä, miten uusi työharjoittelija

lähestyy uuden kokonaisuuden ohjelmointia olemassa olevan kokemuksen ja osaamisensa pohjalta. Toiminnon ollessa tässä vaiheessa vielä asiakasyrityksen liiketoiminnan jatkuvuuden kannalta toissijainen, ei ollut myöskään kriittistä puuttua toteutukseen liiemmin vielä tässä vaiheessa, kun se tultaisiin joka tapauksessa katselmoimaan koodin osalta ennen seuraavaan versioon lisäämistä, kuten projektiryhmässä on käytäntönä.

Koodin kirjoittamisessa pyrittiin ensin luomaan painikkeet, joilla päästään uuteen, modaaliin ikkunaan, jossa uuteen ominaisuuteen liittyvät komponentit ovat. Tässä projektijohto kysyi toteuttajan mielipidettä ulkoasun osalta. Näkemykset olivat yhtenevät ja lopulta loogisia, sujuvan käyttökokemuksen kannalta mielekkäitä vaihtoehtoja painikkeiden sijainnille ei käytännössä ollutkaan. Vaadin-sovelluskehys mahdollistaa hyvin nopeasti kyseisten painikkeiden luomisen ja tästä voitiin helposti edetä varsinaisten ikkunoiden luomiseen. Molemmat painikkeet vievät ikkunan otsikkoa lukuun ottamatta identtisiin ikkunanäkymiin. Ikkunaa luodessa tuli pääasiassa vain määritellä ikkunan koko, sijainti ruudulla, sen modaalisuus ja se, miten käyttäjä mahdollisesti voi muokata ikkunaa.

Painike- ja ikkunaluokkien jälkeen päästiin toteuttamaan omassa Java-luokassaan (lähdekoodiliite OppariversioTehtavaSelector.java) varsinainen ikkunan sisältö, johon tulisi kaikki tarvittavat layout-komponentit ja interaktiiviset käyttöliittymäkomponentit, kuten painikkeet, tekstikentät, valitsimet, asiakirjataulu ja asiakirja-dropzone. Tämän luokan työstämiseen kului valtaosa koko toimeksiantoon käytetyistä työtunneista ja koko prosessista.

Kun kaikki näkymät olivat ulkoisesti valmiit ja komponenttien toiminnallisuus koodin osalta sen näköisiä, että ominaisuutta voidaan alkaa kokeilla, oli aika siirtyä testaamaan luotua kokonaisuutta. Testaaminen suoritettiin virtuaalikoneella omassa Microsoft SQL Server 2012 –testitietokannassa. Tässä työvaiheessa tuli kokeilla luoda erilaisia tehtäviä tehdyllä käyttöliittymällä ja katsoa, että tietokannasta haetaan oikea tieto ja että komponenttien toiminta tekee oikeat muutokset tietokantaan.

Kun eri testitapaukset voitiin todeta onnistuneiksi ja vaatimusmäärittelyn kaikki kohdat toteuttaviksi, voitiin ominaisuus lisätä web-sovelluksen kehitysversioon ja asettaa koodi muun projektitiimin tarkasteltavaksi. Tätä vaihetta kutsutaan ohjelmakoodin asettamiseksi katselmointitilaan. Katselmoinnissa ohjelmistoarkkitehti lukee lisätyn koodin läpi ja testaa vielä kertaalleen ominaisuutta.

Ohjelmistoarkkitehti totesi ominaisuuden olevan riittävän toimiva tässä vaiheessa ja sen kokeilua päätettiin jatkaa kehitysversiossa vielä kolme viikkoa, joka oli Web-sovelluksen tuotantoon menemiseen jäljellä oleva aika. Ominaisuus päätettiin ottaa mukaan

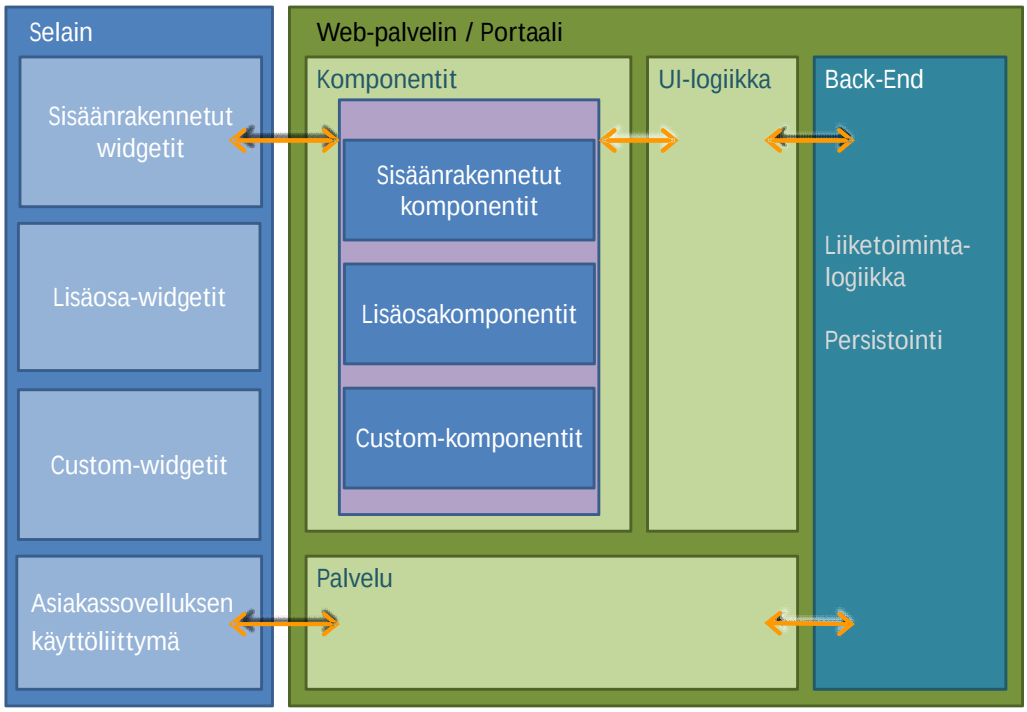
ensimmäiseen tuotantoversioon ja asiakkaan testattavaksi. Asiakas antaisi palautetta, mikäli ominaisuuden toiminnassa olisi jotain puutteita.

5 Vaadin-sovelluskehys

Marko Grönroos kertoo Vaadinin olevan web-sovellusten kehittämiseen suunniteltu sovelluskehys, jonka tarkoitus on tehdä korkealaatuisten, web-pohjaisten käyttöliittymien rakentaminen ja ylläpito helpoksi ja suoraviivaiseksi. Vaadin-sovelluskehyksen yksi perusajatuksista on se, että se tukee kahta eri ohjelmointimallia: palvelinpuolen (server-side) ja asiakasohjelmapuolen (client-side) ohjelmakoodia. Kantavana ideana tässä on se, että Vaadin mahdollistaa web-sovelluksen käyttöliittymän ohjelmoimisen palvelinpuolen ohjelmakoodilla ja varsinaisia, pääasiassa web-selaimille tarkoitettuja ohjelmointikieliä ja -tekniikoita, kuten JavaScriptia, CSS:ää ja HTML:ää ei tarvitse manuaalisesti käyttää lainkaan. Vaadin-sovelluskehys huolehtii web-puolen koodista käyttäjän luodessa pelkkää Java-koodia. (2013, 3)

Vaadin-sovelluskehys toki mahdollistaa myös mainittujen selaintekniikoiden monipuolisen käyttämisen, jos haluaa luoda esimerkiksi omia ”widgettejä” tai muokata käyttöliittymää tavoilla, jotka pelkällä server-side Java-koodilla ei ole mahdollisia. Widgetit ovat käyttöliittymäkomponentteja. Ohjelmointitiimin itse luomista widgeteistä käytetään termiä custom widget.

Yrityssovellusmaailmassa kuitenkin tosiasiassa useissa projekteissa, kuten tämän opinnäytetyön kohdeprojektissa, vähintään oman css-tyylittelyn kirjoittamiselle on jossain määrin tarvetta, kun halutaan vastata asiakkaan tarkkoihin vaatimuksiin jonkin käyttöliittymäkomponentin ulkonäön ja toiminnan osalta. Selaintekniikoihin liittyvän ohjelmakoodin luominen on määrällisesti kuitenkin suhteellisen vähäistä ja usein riittää että tiimissä yksi henkilö on erikoistunut siihen osa-alueeseen. Toki on myös mielekästä, että kaikilla projektitiimin jäsenillä olisi edes perusymmärrys kyseisistä tekniikoista.



Kuva 2: Vaadin-sovelluksen arkkitehtuuri

5.1 Vaadin-sovelluksen rakenne

Vaadin-dokumentoinnissa arkkitehtuurin kerrotaan muodostuvan palvelinpuolen sovelluskehyksestä ja asiakassovelluspuolen ohjelmistomoottorista, jota ajetaan web-selaimessa ja joka kuvantaa sovelluksen käyttöliittymän käyttäjälle ja tuo käyttäjäinteraktion palvelimelle. Web applikaation käyttöliittymä ajetaan Java Servlet –sessiona Java-ohjelmistopalvelimella ja asiakassovelluksen sovellusmoottori ajetaan JavaScriptinä. (Grönroos 2013, 4)

Vaadinilla tehtyjen web-sovellusten ajamiseen ei tarvita Javaan liittyviä selainliitännäisiä, koska asiakassovelluspuolen moottori ajetaan JavaScriptinä selaimessa. Tämä on suuri etu, kun verrataan Vaadinia sovelluskehyksiin, jotka perustuvat Flashiin, Java Appletteihin tai muihin selainliitännäisiin. Vaadin pohjautuu GWT:n (Google Web Toolkit) tukeen kaikissa käytetyimmissä selaimissa niin, että sovelluskehittäjien ei tarvitse huolehtia selaintuesta.

5.2 Layoutit

Oliopohjaisen ajattelumallin kannalta on hyvin selkeää, että Vaadin-käyttöliittymät koostuvat layout-objekteista, joihin lisätään komponentteja addComponent-metodikutsulla. Layouteja voi yhdessä näkymässä olla useita ja niitä voi myös lisätä sisäkkäin.
Layout-tyyppejä on useita erilaisia. Tyypistä riippuen Vaadin-komponentit järjestyvät kyseiselle layoutille esimerkiksi allekkain, vierekkäin tai jollain muulla kehittäjän määrittelemällä tavalla.

Vaadin-käyttöliittymäkomponentit voidaan jakaa kahteen isoon ryhmään: komponentit, joiden kanssa käyttäjä on suorassa vuorovaikutuksessa ja layout-komponentit, joihin edellä mainitun ryhmän komponentteja sijoitellaan käyttäjän nähtäväksi. (Grönroos 2013, 224)

Layout-komponenttien laajan metodivalikoiman ansiosta voidaan jo pelkällä Java-ohjelmointikielellä määritellä hyvin tarkasti se, miten komponentit tulevat varsinaisessa web-sovelluksessa käyttäjän selaimessa näkymään.

Käyttöliittymässä kaikkien layout-komponenttien ja sitä kautta muidenkin komponenttien pohjana on niin sanottu sisältö-layout. Siihen lisätään tarpeen mukaan muita layouteja ja lopulta käyttäjän käytettävissä olevat komponentit, kuten valikot, painikkeet ja tekstikentät lisätään päällimmäisiin layouteihin.

Esimerkiksi tämän opinnäytetyön toimeksiantona olleessa käyttöliittymäikkunassa pohjana on sisältö-layout, johon on lisätty ikkunan keskeltä vaakatasossa jakava HorizontalSplitPanel-komponentti, johon on lisätty VerticalLayout-tyyppiset layoutit, jotka puolestaan järjestävät niille lisätyt käyttöliittymäkomponentit allekkain.

Lähes kaikkia layout-komponentteja voi hienosäätää niille tehtyjen metodien avulla. Voidaan säätää esimerkiksi layoutien kokoa, niille asetettavien komponenttien kohdistamista, riviväliä ja niin edelleen. Jos kuitenkin on niin, että haluttua ulkoasua ja asettelua ei voida saavuttaa valmiiden layout-komponenttien omia metodeja hyödyntämällä, voidaan käyttää CustomLayout-komponenttia, joka mahdollistaa suoran HTML-koodilla toteutettavien pohjien käytön. Tämä kuitenkin asettaa tiettyjä rajoitteita sille, miten kyseistä layoutia voi käsitellä Java-ohjelmakoodin kautta suhteessa muihin layout-komponentteihin. (Grönroos 2013, 226)

5.3 Käyttöliittymäkomponentit

Vaadin-sovelluskehys sisältää hyvin kattavan valikoiman käyttöliittymäkomponentteja. Vaadinin käyttöliittymäkomponenttikirjasto sisältää kaikki web-sovelluksissa yleisimmin käytetyt komponentit sekä niiden lisäksi useita erikoisuuksiakin, kuten drag & drop –

toiminnallisuuteen liittyviä komponentteja, joita on hyödynnetty toimeksianto-ominaisuuden käyttöliittymässäkin.

Rajapintahierarkian huipulla on Component-rajapinta, jonka jokainen käyttöliittymäkomponentti toteuttaa. Luokkahierarkian huipulla taas puolestaan AbstractComponent-luokka, jonka jokainen käyttöliittymäkomponentti perii. Yllä mainittu rajapinta ja yliluokka sisältää runsaasti metodeja, jotka löytyvät täten kaikilta komponenteilta. Esimerkiksi tavallisen tekstikentän ja AbstractComponent-luokan väliin mahtuu pitkä lista luokkahierarkiaa ja useita rajapintoja.

Mikäli kehittäjä haluaa luoda täysin uuden komponentin, tulee kyseisen Java-luokan periä AbstractComponent-yliluokka. (Grönroos 2013, 108)

5.4 Yleisiä Vaadin-komponenttien ominaisuuksia

Kuten mainittua, komponenttien yliluokat ja rajapinnat tarjoavat jokaiselle komponentille suuren valikoiman perusominaisuuksia, jotka liittyvät esimerkiksi komponenttien datakäyttäytymiseen, sijaintiin käyttöliittymässä, kokoon, otsikkoon, oletusarvoihin, näkyvyyteen, kuvakkeisiin ja niin edelleen. Sovelluskehittäjä voi siis luottaa siihen, että lähes kaikista komponenteista löytyy nämä yleisimmät ominaisuudet metodikutsujen takaa.

5.4.1 Otsikko

Suurella osalla käyttöliittymäkomponentteja on otsikko-ominaisuus (Caption). Vaadin-komponentin otsikko sijaitsee käyttöliittymäkomponentin yllä, alla, sivulla tai komponentin sisällä näkyvä teksti, jolla pyritään kuvaamaan mihin käyttäjän tulee komponenttia käyttää. (Grönroos 2013, 108)

Otsikon asettaminen on suoraviivaisuudessaan Vaadin-sovelluskehyksen käytön yksinkertaisuutta hyvin kuvaavaa. Jos halutaan luoda vaikkapa painike, voidaan sille asettaa otsikko yhdellä setCaption-metodikutsulla. Joissain komponenteissa otsikon voi antaa suoraan parametrina. Esimerkiksi oletetaan, että kehittäjä haluaa luoda käyttöliittymään tekstikentän, johon käyttäjän tulisi kirjoittaa etunimensä. Komponentin caption-teksti annetaan parametrina TextField-luokan constructorille ja tämän jälkeen komponentti on valmis käyttöliittymään lisättäväksi addComponent-metodikutsulla.

```
TextField etunimi = new TextField("Etunimi");
layout.addComponent(etunimi);
```

Kuva 3: Caption-esimerkki

5.4.2 Komponentin käyttöönotto ja näkyvyys

Lähes jokaisen Vaadin-komponentin säädettäviin ominaisuuksiin kuuluu käyttöönotto (enabled) ja näkyvyys (visibility). Metodikutsuilla setEnabled(true) ja setVisible(true), voidaan asettaa käyttöliittymäkomponentti käyttäjän käyttöön ja nähtäväksi tai vastavuoroisesti asettaa ns. ”harmautettuun” eli disabled-tilaan ja myös piilottaa näkyvistä antamalla mainittuihin metodikutsuihin parametriksi boolean-arvo false.

Tätä Vaadin-sovelluskehyksen ominaisuutta käytetään hyvin paljon yrityssovelluksia tehtäessä, koska on usein niin, että yrityksen henkilöstöllä ja sen asiakkaillakin voi olla useita eri käyttäjärooleja, joiden perusteella tietyt ominaisuudet ovat käytettävissä tai näkyvillä. Esimerkiksi yrityksen pääkäyttäjä voi mennä napin painalluksella muokkaamaan muiden käyttäjien tietoja, mutta tavalliselta asiakaskäyttäjältä on luonnollisesti syytä estää kyseiseen näkymään pääsy. Asiakasprojektissa käyttäjillä on yli kymmenen eri roolia, joten käyttöliittymäkomponenttien näkyvyyden säätely oli huomattava osa jokaisen käyttöliittymänäkymän ohjelmointia. Komponentin käytettävyyden ja näkyvyyden asettaminen on niin ikään suoraviivaista.

```
etunimi.setEnabled(false);
etunimi.setVisible(true);
```

Kuva 4: Enabled ja Visible -esimerkki

5.4.3 Komponenttien koon sääteleminen

Käyttöliittymäkomponenttien koon sääteleminen on hyvin suoraviivaista. Kaikkien käyttöliittymäkomponenttien kokoa voi säätää. Koon voi määrittää pikseleinä tai prosentteina, jolloin voidaan vaikkapa määrittää, että komponentin leveys on esimerkiksi viisikymmentä prosenttia sen layoutin koosta, johon komponentti on sijoitettu. Komponentin voi asettaa viemään koko layoutin tilan metodikutsulla setSizeFull(). (Duarte 2013, 102)

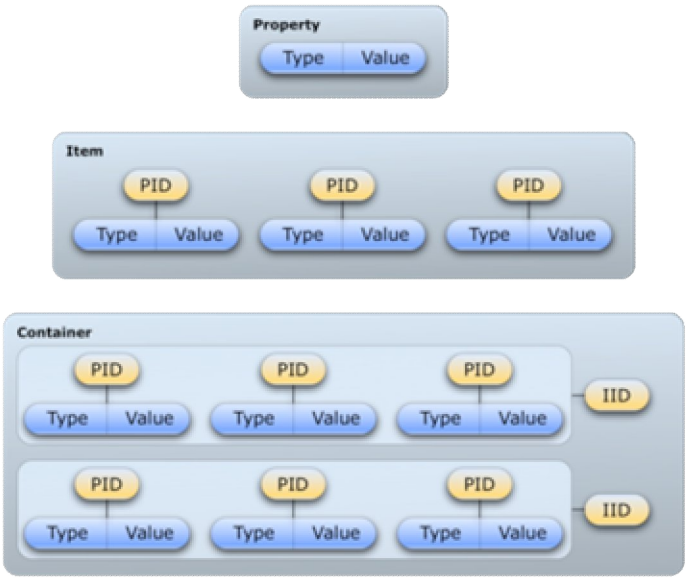
5.5 Vaadin-komponenttien periminen

Asiakasprojektissa on noudatettu yleistä kehitystapaa Vaadin-sovelluskehyksellä rakennetuissa projekteissa, joissa luodaan omia komponenttiluokkia, jotka periytyvät Vaadin-sovelluskehyksen komponenttiluokista. Esimerkiksi jos halutaan luoda taulukomponentti, niin hyvän käytännön mukaista olisi luoda vaikkapa luokka OmaTable, joka perii Vaadinin oman

komponenttiluokan Table. Tämä on hyvä tapa luoda myös oma ikkunaluokka, jollaisesta luodaan kaikki projektin ikkunaobjektit niin, että niillä on esimerkiksi aina yhtenevä css-tyyli ja perustoiminnallisuus, eikä niitä tarvitse erikseen asettaa jokaiselle ikkunalle. Tällöin myös halutut css-tyylimuutokset saadaan kaikille ikkunoille kerrallaan.

5.6 Vaadinin tietomalli

Vaadinin tietomallin hierarkian muodostaa kolme rajapintaa: Container, Item ja Property. Jos näiden tasojen välistä suhdetta ajateltaisiin vaikkapa relaatiotietokannan tauluna, niin container vastaisi taulua, item taulun riviä, ja property saraketta. (Grönroos 2013, 312)



Kuva 5: Vaadinin tietomalli

Komponentteja voi sitoa suoraan tietokantaan. Käytännössä tämä tarkoittaa sitä, että komponentti voi näyttää suoraan tietokannasta saatavan arvon tai kenttäkomponenttiin syötetty arvo tallentuu suoraan tietokantaan. Tavallisimmin komponenttien sisältämiä syöttökenttiä tai valikkoja sidotaan tietokantaan niin, että niissä tapahtuvat muutokset ja niihin syötetty tieto menee suoraan komponentista tietokantaan. Tämä voidaan tehdä niin, että transaktioon tarvitaan erillinen toiminto käyttäjältä tai niin, että tietoa liikkuu automaattisesti saman tien, kun tieto on syötetty kenttään. (Holan, Kvasnovsky 2013, 241)

6 Asiakasyrityksen web-sovelluksen toiminta ja rakenne

Asiakasyritykselle kehitettävä web-sovellus luodaan Vaadin-sovelluskehystä käyttäen. Menemättä asiakasyrityksen tietoturvaa vaarantavalle tasolle asiakkaalle tuotettavan web-sovelluksen yksityiskohdista puhuttaessa, voidaan sovellusta kuvailla yleisluontoisemmin.

Asiakasyritykselle kehitettävässä käyttöliittymässä kirjaudutaan aluksi sisään käyttäjätunnuksella ja salasanalla. Käyttäjärooleja on erilaisia, kuten toimisto-, esimies-, ylläpito- ja suunnittelijaroolit.

Etusivulla aukeaa näkymä toimeksiannoista. Toimeksiannot liittyvät johonkin asiakkuuteen ja ensimmäinen ja oleellisin tieto näkymässä kunkin toimeksiannon kohdalla onkin kyseiseen toimeksiantoon liittyvän asiakkaan suku- ja etunimi. Sen jälkeen tila, jossa toimeksianto parhaillaan on. Toimeksiannon tiloja ovat muun muassa ”valmisteilla”, ”työskentely” ja ”vastaanotettu”.

Toimeksiantoa hiirellä painamalla pääsee Toimeksianto-näkymään. Näkymässä on tarkasteltavissa tarkempaa tietoa asiakastoimeksiannosta. Käyttäjätunnuksen rooleista riippuen näkymässä on erilaisia toimintoja, joilla asiakkaan ja toimeksiannon tietoja voi muokata. Juuri tähän näkymään tulee painikkeet, joista pääsee lisäämään yleisen tehtävän toimeksiannolle tai mihin tahansa sille lisätylle palvelulle.

Toimeksiannon tietojen alapuolella on näkymä toimeksiannolle lisätyistä palveluista. Kullekin palvelulle asetetaan vastuuhenkilö eli asiakasyrityksen työntekijä, joka palvelua tulee työstämään.

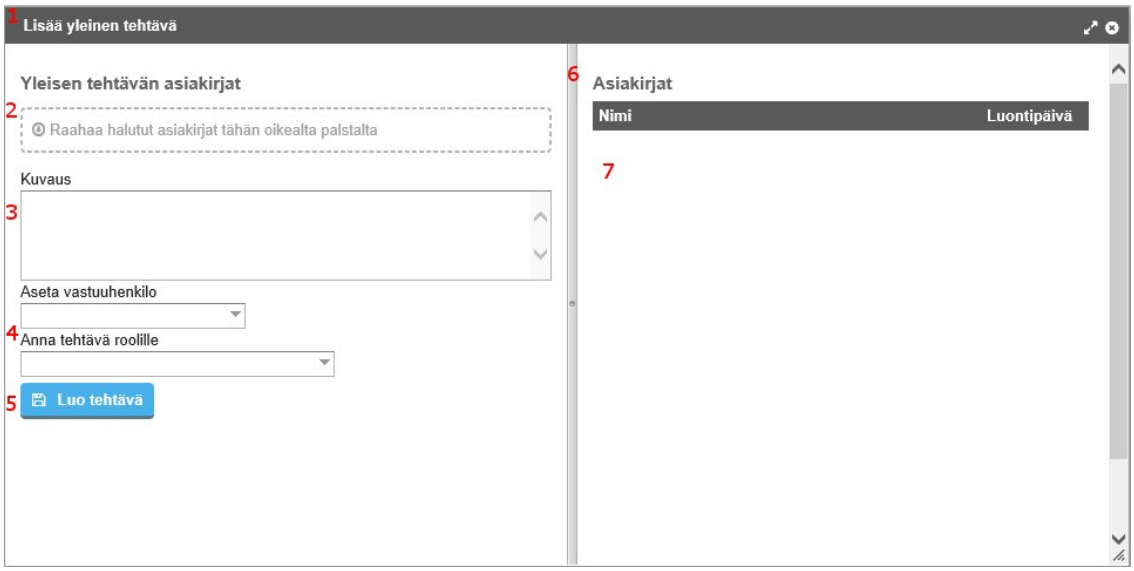
Opinnäytetyönä toteutetun ominaisuuden kannalta web-sovelluksen tärkeimmät näkyvät ovat siis Toimeksianto- ja Tehtävät-näkymä. Edellä mainitussa sijaitsee kyseiset napit ja jälkimmäiseen ilmestyy ominaisuudella luodut tehtävät.

7 Toimeksiantoon liittyvät käyttöliittymäkomponentit

Opinnäytetyön toimeksiantotoiminnallisuudessa eli yleisen tehtävän lisäämisessä asiakasyrityksen käyttäjätunnuksille, käytettiin useita eri Vaadin-sovelluskehyksen käyttöliittymäkomponentteja. Tässä kappaleessa on havainnollistettu kuvalla eri komponenttien ulkoasua sekä niiden käyttötarkoitusta. Kuva on suora ruutukaappaus toiminnassa olevasta, opinnäytetyön aiheena olevasta ominaisuudesta asiakkaan web-sovelluksessa. Kuvassa eri komponentit ja komponenttiryhmät on numeroitu ja niitä avataan tekstikappaleissa numerojärjestyksessä ensimmäisestä viimeiseen.

Erilaisia käyttöliittymäkomponentteja sisältyi runsaasti tähän yhteen ikkunaan, joka aukeaa modaalina eli niin, ettei taustalla näkyvää käyttöliittymää voi ikkunan näkyessä käyttää. Modaalisuudella pyritään siihen, että käyttäjä ei vahingossakaan muokkaisi ikkunan taustalla näkyvän käyttöliittymän komponentteja tämän ikkunan ollessa auki. Näin vältetään virhetilanteet esimerkiksi asiakastietoja käsitellessä.

Yksi projektin web-sovelluksen arkkitehtuurisista pääajatuksista oli, että käyttäjä ei voisi tehdä liian montaa asiaa yhtä aikaa samassa työtehtävässä (toimeksiannossa), vaan käyttäjän työskentely etenee tarkasti rajatuin vaihein. Tällä pyritään selkeyteen niin käyttöliittymässä, kuin sovelluksen back-endissä pyörivässä liiketoimintalogiikassakin. Jälkimmäinen huomio tarkoittaa käytännössä sitä, että tietokantaan ei näin tule ristiriitaisia syötteitä käyttöliittymästä. Liiketoimintalogiikkaa haittaavaa ristiriitaisuutta syntyisi, jos käyttäjä esimerkiksi tallentaisi asiakastiedot toisessa ikkunassa samalla, kun hänellä on saman asiakkaan tietoja käsittelevä pop-up ikkuna auki.



Kuva 6: Toimeksiannon käyttöliittymäkomponentteja

7.1 Kohta 1: Ikkunan ylätunniste

Ikkunan ylätunniste on ikkunan yläosassa näkyvä otsikko. Se kertoo mikä kyseisen ikkunan sisältämän toiminnallisuuden päätarkoitus on.

Ikkunan perusrakenne ja sen sisältö ovat kaksi erillistä Java-luokkaa. Ikkunan ylätunniste, modaalisuus, korkeus, leveys ja ikkunan sisällön muodostava objekti määritetään omassa luokassaan (lähdekoodiliite OppariVestioTehtavaWindow.java). Varsinainen sisältö

liiketoimintalogiikkaan suoraan vaikuttavine käyttöliittymäkomponentteineen määritellään toisessa luokassa (lähdekoodiliite OppariverioTehtavaSelector.java).

Ensiksi mainittu luokka perii projektin oman Window-luokan, josta se saa muunmuassa css-tyylinsä ja toiminnallisuutta, joka on haluttu toteuttaa hieman Vaadinin omasta ikkunatoteutuksesta poikkeavasti. Projektin oma Window-luokka puolestaan perii suoraan Vaadinin oman Window-luokan, joka puolestaan antaa ikkunalle kaikki oleelliset ikkunan käyttäytymiseen liittyvät ominaisuutensa, kuten koon muuttamisen, liikuttamisen, keskittämisen ja niin edelleen.

Vaadinin selkeys ja suoraviivaisuus komponenttien muotoiluun liittyen tulee hyvin esille ikkunan luomisessa. Ikkunan ylätunniste asetetaan metodilla setCaption(”ylätunnisteen teksti”), ikkuna asetetaan modaaliseksi metodilla setModal(true), leveyden ja korkeuden voi asettaa pikseleinä tai prosentteina suhteessa layoutin kokoon, johon komponentti luodaan. Parametreina annetaan siis ensin luku ja sen jälkeen Unit-enumeraation arvo. Esimerkiksi setWidth(50, Unit.PERCENTAGE) ja setHeight(50, Unit.PERCENTAGE).

Kuten mainittua, ikkunan täyttävä sisältö määritellään toisessa luokassa. Oletetaan, että sisällön määrittävän Java-luokan nimi on TehtavaSelector. Ensin määritellään ikkunaluokkaan objekti TehtavaSelector selector. Tämän jälkeen ikkunan sisältö yksinkertaisesti määritetään metodikutsulla setContent(selector).

Muotoilu tehdään siis palvelinpuolen ohjelmakoodina eli tavallisemmin ilmaistuna server-side –koodina, metodikutsuina muodossa: omalkkuna.metodiNimi(parametrit). Metodien nimet ovat hyvin itsestään selviä ja annettavien parametrien määrä on minimaalinen. Usein riittää yksi parametri, kuten yllä olevissa esimerkeissä.

7.2 Kohta 2: Dropzone asiakirjoille

Ikkunan sisällössä ensimmäisenä ylävasemmalla näkyy Label-tekstikomponentin alla katkoviivalla rajattu alue. Tätä aluetta kutsutaan dropzoneksi ja siihen raahataan oikealta, kuvassa tyhjältä, taululta asiakirjoja. Kyseiset asiakirjat tallennetaan Luo tehtävä –napin painalluksen yhteydessä ja liitetään tehtävään joka näkymässä luodaan. Myöhemmin kyseiset asiakirjat on nähtävissä listana, painettavina linkkeinä Tehtävät-näkymässä luodon tehtävän kohdalla.

Vaadin-sovelluskehyksessä on toteutus Drag and drop –toiminnallisuudelle. Drag and drop tarkoittaa tässä yhteydessä sitä, että voidaan raahata hiirellä ikkunan oikealla puolella olevasta taulusta asiakirjoja näkymän vasemmalla puolella olevalle ”dropzonelle” eli

käyttöliittymänäkymään rajatulle alueelle, johon asiakirjatiedostot tiputetaan ja josta ne voidaan liittää kyseiseen tehtävään ja liiketoimintalogiikkaan. Dropzone on toteutettu erillisessä Java-luokassa (lähdekoodiliite OppariversioTehtavaDropzone.java) ja sen toiminnallisuus on verrattain monimutkainen.

Tähän toiminnallisuuteen liittyvän dropzonen ohjelmakoodi on osittain jo olemassa olevaa koodia web-sovelluksen toisessa näkymässä olevasta dropzone-toteutuksesta. Siihen on kuitenkin tehty paljon näkymä- ja käyttötarkoituskohtaisia muutoksia.

Tiivistettynä tämä komponentti toimii tässä yhteydessä niin, että YleinenTehtavaDropzone perii Dropzone-luokan. Sille määritellään source table eli taulu, jolta halutaan liikuttaa rivejä eli tässä tapauksessa asiakirjoja, tälle kyseiselle dropzonelle. Koodissa määritellään tallennus- ja poistotoiminnot dropzonella sijaitseville dokumenteille. Suuri osa luokan koodista käsittelee css-muotoilua ja asiakirjojen tilojen tarkistusta ja asettamista. Lopussa metodi clearDropzone tyhjentää dropzonen ja valmistaa sen seuraavaa käyttökertaa varten.

7.3 Kohta 3: Kuvaus

Käyttöliittymänäkymän Kuvaus-kohdassa on tekstikenttä, jossa voidaan antaa vapaa kuvaus luotavalle tehtävälle. Tehtävän kuvaus on myöhemmin nähtävissä web-sovelluksen Tehtävät-näkymässä kyseisen tehtävän kohdalla.

Tämä komponentti on nimeltään TextArea ja se on suoraan Vaadinin komponenttikirjastosta. Omia muokkauksia ei tarvittu joten ei ole tarpeettomasti lähdetty luomaan omaa, TextArean perivää luokkaa. Kaikessa lyhykäisyydessään TextAreaa voisi luonnehtia isoksi tekstinsyöttökentäksi. Perinteinen, yhden rivin tekstikenttäkomponentti on puolestan TextField.

Kun ikkuna avataan, suoritetaan kentälle metodikutsu setValue(””). Tämä varmistaa että kenttä on tyhjä, mutta ei kuitenkaan sisällä null-arvoa. Muotoilussa komponentille kutsutaan metodi setWidth(”100%”). Tämä metodikutsu määrittää tekstikentän koko ikkunanäkymän vasemman puoliskon levyiseksi. Jos keskellä olevaa näkymänjakajaa liikutetaan, niin tekstikentän koko muuttu myös. Näkymänjakaja kuvaillaan kohdassa kuusi.

On huomionarvoista, että leveyden voi uusimmassa Vaadin-versiossa ilmaista myös tällä lyhyellä, yhden parametrin tavalla, jossa yksikkö määräytyy prosenttimerkillä Stringin sisällä heti integer-arvon perään.

Erilaisiin kenttäkomponentteihin syötetty tekstisisältö on komponentin arvo eli value. Kun arvoa halutaan käyttää jossain muualla, esimerkiksi tallennuksessa, niin se saadaan metodikutsulla getValue(). Arvoja siis asetetaan ja haetaan olio-ohjelmoinnille tyypillisellä getter- ja setter-mallilla.

7.4 Kohta 4: Vastuuhenkilö ja –roolit

Yleinen tehtävä –toiminnallisuudessa halutaan luoda tehtävä tietylle asiakasyrityksen työntekijälle tai kokonaiselle käyttäjäryhmälle käyttäjätunnusten roolien mukaan. Tässä kohdassa valitut arvot määrittävät ne käyttäjät, joiden Tehtävät-näkymään kyseinen tehtävä sen luomisen yhteydessä ilmestyy.

Tämä osa toiminnallisuudesta on toteutettu Vaadinin ComboBox-komponenteilla. ComboBox on tyhjä kenttä, jonka avaamalla saa näkyviin vaihtoehdot, joista voi klikkaamalla valita sen käyttäjän tai roolin, jolle tehtävä halutaan asettaa. Projektissa on luotu oma, tyylitelty ja muokattu ComboBox, joka perii Vaadinin ComboBox-luokan.

ComboBoxeihin ilmestyvät vaihtoehdot etsitään tietokannasta. Tietokannassa on käyttäjätunnukset ja sieltä tietyin kriteerein valikoituu ComboBoxeihin henkilöt, joille kyseiset käyttäjätunnukset kuuluvat. Roolit haetaan kannasta collectionina ja saatu collection lisätään komponenttiin esimerkiksi metodikutsulla selectVastuurooli.addItems(vastuuroolit). Tiettyjä metodikutsuja tarvitaan, että nimet näkyvät käyttöliittymässä oikein kokonimillä.

7.5 Kohta 5: Luo Tehtävä –nappi

Toimeksiannossa pyydetyn ominaisuuden kannalta tärkein komponentti ikkunassa on vasemmassa alakulmassa näkyvä Luo tehtävä –nappi. Sen painallus suorittaa kaikkiin muihin komponentteihin asetettujen arvojen tallentamisen tehtävään sekä tehtävän luomisen tietokantaan ja sitä kautta Tehtävät-näkymään.

Napin luokka perii Vaadinin Button-luokan ja on muokattu ja tyylitelty asiakasprojektin web-sovelluksen linjan mukaiseksi. Vaadinin Button-luokkiin sisältyy ClickListener niminen rajapinta, joka kuuntelee ja tunnistaa napinpainalluksia ja laukaisee siihen kirjoitetun metodin, kun painallus tapahtuu. Tässä tapauksessa napin klikkaaminen kutsuu processReadyToMail-metodin jossa tapahtuu asiakirjojen liittäminen ja kenttäkomponenttien arvojen ottaminen tehtävään. Tehtava-luokassa tehtavabuilderilla varsinaisesti luodaan tehtävä sille syötetyillä parametreilla, jotka ovat tämän näkymän komponenteista saatuja

arvoja. Painalluksen lopussa tämän ikkunan kentät tyhjennetään, päivitetään ja ikkuna suljetaan.

7.6 Kohta 6: Näkymän jakaja

Täsmälleen näkymän keskellä on Vaadinin komponentti HorizontalSplitPanel. Kyseinen komponentti on kätevä halkaisemaan näkymän keskeltä, niin että näkymän puoliskojen jako näkyy käyttäjälle selkeästi ja käyttäjä voi myös muokata halutessaan jakajan sijaintia.

HorizontalSplitPanelin sijainti asetetaan näkymää luodessa aivan näkymän keskelle metodikutsulla setSplitPosition(50, Unit.PERCENTAGE). Tämä tarkoittaa sitä, että ensimmäinen puolisko kattaa tasan viisikymmentä prosenttia näkymästä. SplitPanelin puoliskojen sisällöt asetetaan metodikutsuilla setFirstComponent ja setSecondComponent.

7.7 Kohta 7: Asiakirjataulu

Ikkunan oikealle puoliskolle ei tule muita komponentteja kuin taulu, johon ilmaantuu kaikki toimeksiannolle lisätyt asiakirjat. Taulussa näkyy omina sarakkeinaan asiakirjan nimi ja asiakirjan toimeksiantoon liittämisen ajankohta.

Näistä asiakirjoista voidaan valita tehtävään liitettävät asiakirjat aiemmin kuvaillulla Drag and drop –tekniikalla. Asiakirjat ilmestyvät tauluun riveinä, ja yksi rivi vodaan suoraan raahata hiirellä dropzonelle ja siten liittää tehtävään.

8 Toimeksianto ja kehitysympäristön muut teknologiat

Java EE:llä toteutettavat web-sovellus –projektit sisältävät tavallisesti useita eri teknologioita, jotka muodostavat eri työkalujen kanssa kehitysympäristön. Vaadin on suunniteltu toimimaan projekteissa, joissa hyödynnetään yleisimpiä Java EE –stackiin liittyviä teknologioita. Esimerkiksi eri sovelluspalvelinten kanssa kuten Apache Tomcat tai, kuten tämän projektin tapauksessa, JBoss WildFly8.

Tämän projektin kohdalla kehittäjien tuli ymmärtää, miten Vaadin-sovelluskehystä käytetään web-sovelluksen kehittämiseen EJB3:n, Mavenin, JBoss Weld CDI:n, Hibernaten, JBoss WildFly8:n ja SQL Server 2012:n kanssa.

8.1 Apache Maven

Maven on pääasiassa Java-projekteille kehitetty projektinhallintaan ja projektin rakentamisautomaatioon tarkoitettu työkalu. Sen avulla voidaan määritellä ja hallinnoida Java-projektin käyttämiä kirjastoja ja niiden versioita yksinkertaisesti yhdestä paikasta käsin. Maven-projektit sisältävät yhden tai useamman pom.xml-tiedoston, jossa projektin eri moduulien riippuvuudet ja versiot on selkeästi jaettu. Pelkkä versionumeron vaihtaminen Mavenin käyttämässä xml-tiedostossa päivittää kyseisen moduulin version projektissa. Mavenin avulla projekti pakataan muotoon, jolla se voidaan ajaa esimerkiksi JBoss WildFly8 -sovelluspalvelimella. (Wikipedia 2014.)

8.2 JBoss WildFly8

Java EE web applikaatiot ajetaan sovelluspalvelimella. Sellaisia ovat mm. Oracle GlassFish, Apache TomCat sekä JBoss WildFly8. Projektissa, jossa tämän opinnäytetön toimeksianto toteutettiin, käytettiin JBoss WildFly8 –sovelluspalvelinta. Kyseessä on JBossin sovelluspalvelinperheen, kirjoitushetkellä tuorein tulokas, joka oli projektin sovellusarkkitehdin mukaan helppo ottaa tähän projektiin käyttöön. Se antaa kehittäjien käyttöön CDI-referenssi-implementaation, Weldin. WildFly8:ssa on myös uusi web-palvelin –implementaatio, Undertow, joka on uusi ja tekniikaltaan joustava, koska se mahdollistaa web-palvelimen rakentamisen käyttäen useita eri moduuleja. (Marchioni 2014, 18)

8.3 EJB3.1

EJB eli Enterprise JavaBean on back-end –komponenttiarkkitehtuuri, jolla rakennetaan yrityssovelluksia modulaarisesti. Yksi Java-luokka voi olla Enterprise JavaBean. EJB-objekteja on pääasiassa kolmea erilaista. On SessionBean, joka voi olla tilaton tai tilallinen, ja joka esittää asiakasohjelman kutsumia, liiketoimintalogiikkaan liittyviä toimintoja. EntityBean-komponentit edustavat pysyvää tietoa, jota säilytetään esimerkiksi tietokannassa. Esimerkiksi toimeksianto-ominaisuuden käsittelemät vastuuhenkilöt, asiakirjat ja tehtävät ovat EntityBean-objekteja. Message Driven Bean –komponentteja kutsutaan asynkronisesti ulkoisten tapahtumien kautta. (Kodali, Rathod, Wetherbee, Zadrozny 2013, 22)

8.4 Hibernate

Hibernate on WildFly8-ohjelmistopalvelimella käytössä oleva Java-kirjasto. Hibernate tuo projektiin mukaan annotaatiot, joilla Java-objektit voidaan kartoittaa, tutummin ilmaistuna mapata, tietokantaan. Näin voidaan luoda Java-luokkia, jotka rakenteeltaan edustavat taulua tietokannassa ja sen sarakkeita. Hibernate-annotaatioiden avulla voidaan myös suhteellisen

yksinkertaisesti luoda yhteyksiä eri taulujen välissä. Esimerkiksi Asiakas-taulu yhdistetään Tilaukset-tauluun @OneToMany-annotaatiolla, joka ilmaisee, että yksi Asiakas-objekti voi liittyä useaan Tilaus-objektiin. (Linwood, Minter, Ottinger 2014, 9)

9 Yhteenveto

Ominaisuuden toteutus onnistui kaikilta osin hyvin ja suurin piirtein odotusten mukaisesti. Mitään täysin odottamattomia vastoinkäymisiä ei ilmennyt suunnittelu- eikä toteutusvaiheissa. Kehitysympäristön kokoaminen opinnäytetyötä varten oli suhteellisen monimutkainen prosessi ja vaati hieman puutteellisen dokumentoinnin lisäksi myös ohjausta projektitiimin sovellusarkkitehdilta. Java EE –projektin kehitysympäristöön kuuluu tavallisesti paljon asennettavaa. Tekninen ympäristö tuli kuitenkin tutuksi työnteon edetessä ja itseopiskelun myötä.

Lopulta opinnäytetyön toteutuspuoli tehtiin hyvin itsenäisesti. Oli hyvin opettavaista etsiä itse tietoa Java EE –projektin eri moduuleista ja miten ne toimivat käytännössä. Loppua kohden projektiryhmän kokeneemmat jäsenet saivat testattavaksi keskeneräisiä, joskin lähes valmiita versioita toiminnallisuudesta ja ulkoasuun tehtiin pientä hienosäätöä, jotta se olisi tyyliltään mahdollisimman yhtenevä muun käyttöliittymän ja näkymien kanssa. Osa muutoksista tuli myös asiakasyrityksen edustajien muuttaessa ja tarkentaessa ominaisuuden vaatimusmäärittelyä.

Opinnäytetyöhön liittyvän suunnittelun ja toteutuksen voidaan sanoa onnistuneen. Vaikka tarkkaa aikataulua ei varsinaisesti asetettu, toteutui työ suhteellisen järkevässä ja asiakkaalle sopivassa ajassa. Ominaisuus meni sellaisenaan tuotantoon toteutuksen ja testauksen jälkeen.

Lähteet

Duarte, A. 2013. Vaadin 7 UI Design By Example: Beginner's Guide. Birmingham: Packt Publishing.

Grönroos, M. 2013. Book of Vaadin: Vaadin 7 Edition. 2. painos. Turku: Vaadin.

Holan, J. & Kvasnovsky, O. 2013. Vaadin 7 Cookbook. Birmingham: Packt Publishing.

Kodali, R., Rathod, C., Wetherbee, J. & Zadrozny, P. 2013. Beginning EJB 3. 2. painos. New York City: Apress.

Linwood, J., Minter, D. & Ottinger, J. 2014. Beginning Hibernate. 3. painos. New York City: Apress.

Marchioni, F. 2014. Practical Java EE 7 Development on WildFly. Rome: ITBuzzPress.

Verkkolähteet

Wikipedia.org. Maven. Viitattu 2.2.2015.
http://en.wikipedia.org/wiki/Apache_Maven

Kuvat

Liitteet

Liite 1 OppariversioTehtavaSelector.java

```
// Paketin nimi ja importit piilotettu sekä asiakasyritykseen ja toimialaan viittaavat
// luokka-, objekti- ja metodinimet muutettu

@SuppressWarnings("serial")
@UIScoped
public class YleinenTehtavaSelector extends CssLayout {

    @Inject
    ValidatorGroup validationGroup;

    @Inject
    [Kayttajaoikeuksiin liittyvä palveluluokka]

    private List<Tunnari> vastuuhenkilot = new ArrayList<Tunnari>();
    private List<Roolit> vastuuroolit = new ArrayList<Roolit>();

    @Inject
    @Apply(method = VaadinSetters.ADD_STYLE_NAME, value = "h2")
    private PLabel searchHeader;

    @Inject
    @SetI18nCaption("hallinta.palvelu.v.taulukko.kuvaus")
    private TextArea kuvaus;

    @Inject
    @New
    private Table searchTable;

    @Inject
    @Apply(method = VaadinSetters.ADD_STYLE_NAME, value = "h2")
    private PLabel asiakirjaDropzoneHeader;

    @Inject
    @New
    private CssLayout outgoingPane;

    @Inject
    private PComboBox selectVastuuhenkilo;

    @Inject
    private PComboBox selectVastuurooli;

    @Inject
    @Apply(method = VaadinSetters.ADD_STYLE_NAME, value = "primary-button")
    @Apply(method = VaadinSetters.ADD_STYLE_NAME, value = "p-save")
    private PButton luoTehtavaButton;

    @Inject
    @Apply(method = VaadinSetters.ADD_STYLE_NAME, value = "p-secondary-button")
    @Apply(method = VaadinSetters.ADD_STYLE_NAME, value = "p-cancel")
    private PButton cancelButton;

    @Inject
    @New
    private HorizontalSplitPanel splitPanel;

    @Inject
    javax.enterprise.event.Event<CloseEvent> closeWindowEvent;

    @Inject
    javax.enterprise.event.Event<RefreshEvent> refreshEvent;

    @Inject
    private Identity identity;

    private Toimeksianto toimeksianto;

    private YleinenTehtavaDropzone dropzone;

    private Tunnari vastuuhenkilo;
```

```
private Roolit vastuurooli;

HashSet<Dokumentit> dropatutDokumentit = new HashSet<>();

public void setToimeksianto(@Observes Toimeksianto toimeksianto) {
    kuvaus.setValue("");
    this.toimeksianto = toimeksianto;
}

@PostConstruct
private void init() {

    setSizeFull();
    initializeSourceTable(null);
    buildSplitPanel();
}

private void buildSplitPanel() {

    splitPanel.setSizeFull();
    splitPanel.setImmediate(true);
    splitPanel.setSplitPosition(50, Unit.PERCENTAGE);
    splitPanel.setFirstComponent(buildLeftColumn());
    splitPanel.setSecondComponent(buildRightColumn());

    addComponent(splitPanel);
}

private CssLayout buildLeftColumn() {
    createOutgoingPane();
    cancelButton.addClickListener(clickEvent -> closeParentWindow());
    kuvaus.setWidth("100%");

    return CssLayoutBuilder.get()
            .addStyleName("p-left-col")
            .addComponent(CssLayoutBuilder.get()
                .addComponent(asiakirjaDropzoneHeader)
                .addComponent(outgoingPane)
                .addComponent(luoTehtavaButton)
                .build())
            .build();
    }

private void createOutgoingPane() {
    List<Roolit> Roolityypit = new ArrayList<Roolit>();
    Roolityypit.add(Roolit.rooli1);
    Roolityypit.add(Roolit.rooli2);
    Roolityypit.add(Roolit.rooli3);
    Roolityypit.add(Roolit.rooli4);

    vastuuhenkilot.addAll(kayttajatunnusService.findAllByRoleTypes(RooliTyypit));
    selectVastuuhenkilo.setItemCaptionMode(ItemCaptionMode.EXPLICIT);

    vastuuroolit.addAll(Arrays.asList(Roolit.getRoolityypit()));
    selectVastuurooli.addItems(vastuuroolit);

    for (Tunnari kt : vastuuhenkilot) {
        selectVastuuhenkilo.addItem(kt);
        selectVastuuhenkilo.setItemCaption(kt, kt.getFullName());
    }

    String text = "yleinentehtava.dialog.drag.placeholder";
    dropzone = createDropzone(searchTable, text, false);

    outgoingPane.addComponent(dropzone);
    outgoingPane.addComponent(kuvaus);
    outgoingPane.addComponent(selectVastuuhenkilo);
    outgoingPane.addComponent(selectVastuurooli);

    luoTehtavaButton.addClickListener(clickiEvent -> processReadyToMail(dropzone));
}

private void processReadyToMail(YleinenTehtavaDropzone dropzone) {

    Stream<Asiakirja> kasiteltavatDokumentit = dropatutDokumentit.stream();
```

```
            kasiteltavatDokumentit.forEach (
                ak -> {
            ak.setAsReadyToMailAndSave(AsiakirjaType.MUU_YKKOSKIRJEESSA);
            dropatutDokumentit.add(ak);
            ak.reload();
        });
        vastuuhenkilo = (Tunnari) selectVastuuhenkilo.getValue();
        vastuurooli = (Roolit) selectVastuurooli.getValue();


        Tehtava.createYleinenTehtava(_t("tehtava.name"), kuvaus.getValue(),
            dropatutDokumentit, toimeksianto, vastuuhenkilo, vastuurooli).save();

        toimeksianto.reload();
        refreshEvent.fire(new RefreshEvent());
        closeParentWindow();
    }

    private YleinenTehtavaDropzone createDropzone(final Table sourceTable,
final String placeholderKey, final boolean readOnly) {

        YleinenTehtavaDropzone zone =
        new YleinenTehtavaDropzone(sourceTable, placeholderKey, readOnly,
            dropped -> handleDrop((Asiakirja) dropped),
            deleted -> handleDelete((Asiakirja) deleted));
        return zone;
    }

    private void handleDrop(Asiakirja ak) {
        dropatutDokumentit.add(ak);
    }

    private void handleDelete(Asiakirja ka) {

        Dokumentti reloadedDokumentti = (Dokumentti) dokumentti.reload();
        reloadedDokumentti.setType(UNDEFINED);
        reloadedDokumentti.save();
        reloadedDokumentti.remove();

        populateSourceTable();
    }

    private CssLayout buildRightColumn() {
        searchTable.setWidth("99%");
        searchTable.setSelectable(true);
        searchTable.setMultiSelect(true);

        return CssLayoutBuilder.get()
            .addStyleName("p-right-col")
            .addComponent(searchHeader)
            .addComponent(searchTable)
            .build();
    }

    @Override
    public void attach() {
        super.attach();

        if (toimeksianto == null) {
            closeParentWindow();
            return;
        }
        update();
    }

    public void update() {
        populateSourceTable();
        dropzone.clearDropzone();
        dropatutDokumentit.clear();
        kuvaus.setValue("");
        selectVastuuhenkilo.setValue(null);
        selectVastuurooli.setValue(null);
    }

    @SuppressWarnings("unchecked")
```

```
    private void populateSourceTable() {
        BeanItemContainer<Asiakirja> container = (BeanItemContainer<Asiakirja>)
searchTable.getContainerDataSource();
        container.addAll(Asiakirja.findByToimeksianto(toimeksianto));
    }

    private void initializeSourceTable(final ClientSideCriterion acceptCriterion) {
        final BeanItemContainer<Asiakirja> tableContainer = new BeanItemContainer<>(
            Asiakirja.class);
        searchTable.setContainerDataSource(tableContainer);
        searchTable.setVisibleColumns("name", "dateCreated");
        searchTable.setColumnHeader("name", UILocalisation.localise("dokumentit.nimi"));
        searchTable.setColumnHeader("dateCreated",
UILocalisati   on.localise("dokumentit.luontipaivays"));

        searchTable.addGeneratedColumn("dateCreated",
new FileEntityPTable.ShortDateGenerator());

        searchTable.setDragMode(Table.TableDragMode.ROW);

        searchTable.setColumnExpandRatio("name", 4.0f);
        searchTable.setColumnExpandRatio("dateCreated", 1.0f);
    }

    private void closeParentWindow() {
        closeWindowEvent.fire(new YleinenTehtavaWindow.CloseEvent());
    }
}
```

Liite 2 TilatullePalvelulleYleinenTehtavaSelectorOppariversio.java

```
// Oleellinen osa Tilatulle palvelulle asetettavan yleisen tehtävän koodia,
// joka poikkeaa jokseenkin toimeksiannolle asetettavan yleisen tehtävän koodista
// Paketin nimi ja importit piilotettu sekä asiakasyritykseen ja toimialaan viittaavat
// luokka-, objekti- ja metodinimet muutettu

@UIScoped
public class TilatullePalvelulleYleinenTehtavaSelector extends CssLayout {



    private TilattuPalvelu tilattuPalvelu;

    private void processReadyToMail(YleinenTehtavaDropzone dropzone) {

        Stream<Asiakirja> kasiteltavatAsiakirjat = dropatutAsiakirjat.stream();
            kasiteltavatAsiakirjat.forEach (
                ak -> {
            ak.setAsReadyToMailAndSave(AsiakirjaType.MUU_YKKOSKIRJEESSA);
            dropatutAsiakirjat.add(ak);
            ak.reload();
        });
        vastuuhenkilo = (Kayttajatunnus) selectVastuuhenkilo.getValue();
        vastuurooli = (RooliTyyppi) selectVastuurooli.getValue();

        TilattuPalvelu temp = (TilattuPalvelu) this.tilattuPalvelu.reload();

        Tehtava.createTilatullePalvelulleYleinenTehtava(_t

             ("toimeksianto.yleinentehtava_tilatullepalvelulle.tehtava.name"),
                        kuvaus.getValue(), temp,
            dropatutAsiakirjat, toimeksianto, vastuuhenkilo, vastuurooli).save();

        toimeksianto.reload();
        refreshToimeksiantoEvent.fire(new RefreshToimeksiantoEvent());
        closeParentWindow();
    }

    public void setTilattuPalvelu(TilattuPalvelu tp) {
        this.tilattuPalvelu = (TilattuPalvelu) tp.reload();
        this.tilattuPalvelu.setToimeksianto(toimeksianto);
    }

    public TilattuPalvelu getTilattuPalvelu() {
        return tilattuPalvelu;
    }

//--------------
```

Liite 3 OppariversioTehtavaDropzone.java

```
// Paketin nimi ja importit piilotettu sekä asiakasyritykseen ja toimialaan viittaavat
// luokka-, objekti- ja metodinimet muutettu

@SuppressWarnings("serial")
public class YleinenTehtavaDropzone extends Dropzone {
    private static final Logger log = LoggerFactory
            .getLogger(YleinenTehtavaDropzone.class);
    private Table sourceTable;
    private DeleteHandler deleteHandler;
    private SaveHandler saveHandler;
    private Set<Asiakirja> oldItems = new HashSet<Asiakirja>();
    private Set<Asiakirja> newItems = new HashSet<Asiakirja>();

    public YleinenTehtavaDropzone(final Table sourceTable,
            final String placeHoldTextKey, final boolean readOnly,
            final SaveHandler saveHandler, final DeleteHandler deleteHandler) {
        this(new CssLayout(), sourceTable, placeHoldTextKey, readOnly,
                saveHandler, deleteHandler);
    }

    private YleinenTehtavaDropzone(final CssLayout root, final Table sourceTable,
            final String placeHoldTextKey, final boolean readOnly,
            final SaveHandler saveHandler, final DeleteHandler deleteHandler) {
        super(root);
        this.root = root;
        this.sourceTable = sourceTable;
        this.deleteHandler = deleteHandler;
        this.saveHandler = saveHandler;
        setReadOnly(false);
        addStyleName(DROPZONE_CSS_STYLE);

        placeholder = CssLayoutBuilder
                .get()
                .addComponent(
                        LabelBuilder
                                .get()
                                .with(FontAwesome.ARROW_CIRCLE_O_DOWN.getHtml())
                                .withStyle("p-drag-placeholder-icon")
                                .setSizeUndefined().asHTML().build())
                .addComponent(
                        LabelBuilder.get().with(_t(placeHoldTextKey))
                                .setSizeUndefined()
                                .withStyle("p-drag-placeholder-text").build())
                .build();
        root.addComponent(placeholder);

        setDropHandler(new DropHandler() {
            @Override
            @SuppressWarnings("unchecked")
            public void drop(final DragAndDropEvent event) {
                log.debug("drop() called, event" + event.toString());
                final DataBoundTransferable t = (DataBoundTransferable) event
                        .getTransferable();
                final Container sourceContainer = t.getSourceContainer();
                final Object sourceItemId = t.getItemId();
                final Asiakirja doc = (Asiakirja) sourceItemId;

                saveHandler.execute(doc);
                final CssLayout item = createItem(doc, deleteHandler, readOnly);

                newItems.add(doc);
                addComponent(item);
                sourceContainer.removeItem(sourceItemId);
            }

            @Override
            public AcceptCriterion getAcceptCriterion() {
                return new SourceIs(sourceTable);
            }
        });

    }
```

```
public CssLayout createItem(Asiakirja ka,
        final DeleteHandler deleteHandler, boolean readOnly) {
    log.debug("createItem() called, " + ka.getName());
    log.debug("State" + ka.getProcessState().getTag());
    String statusString = buildStatusString(ka);

    Asiakirja doc = ka;

    Label nameAndStatus = LabelBuilder
            .get()
            .with(String.format(
                    "<a href=\"%s\">%s</a><br>%s",
                    String.format("%s/%s/%s",
                            Configuration.FILES_BASE_URL_PATH,
                            Configuration.UPLOADS_DIR, doc.getUrlFilePath()),
                    doc.getName(), statusString))
            .withStyle("p-ykkoskirje-item-name").asHTML()
            .setSizeUndefined().build();

    Label icon = LabelBuilder
            .get()
            .with(FontAwesomeHelper.getByMediaType(doc.getMediaType())
                    .getHtml()).withStyle("p-ykkoskirje-item-icon")
            .asHTML().setSizeUndefined().build();

    CssLayout item = CssLayoutBuilder.get()
            .addStyleName("p-ykkoskirje-item").addComponent(icon)
            .addComponent(nameAndStatus).build();

    Button deleteButton = new Button(FontAwesome.TRASH_O.getHtml(),
            clickEvent -> {
                YleinenTehtavaDropzone.this.root.removeComponent(item);

                newItems.remove(ka);

                deleteHandler.execute(ka);
            });

    deleteButton.addStyleName(BaseTheme.BUTTON_LINK);
    deleteButton.setHtmlContentAllowed(true);

    if (!readOnly) {
        item.addComponent(deleteButton);
    }

    return item;
}

private String buildStatusString(final Asiakirja ak) {
    StringBuilder sb = new StringBuilder();

    Asiakirja doc = ak;

    Consumer<Date> buildSpan = (date) -> {
        sb.append("<span class=\"p-ykkoskirje-status-");
        sb.append(doc.getProcessState().name());
        sb.append("\">");
        sb.append(_t(doc.getProcessState().getTag()));
        sb.append(" ");
        sb.append(date);
        sb.append("</span>");
    };

    switch (doc.getProcessState()) {
    case LUONNOS:
        buildSpan.accept(doc.getLuontiPvm());
        break;
    case VALMIS_POSTITETTAVAKSI:
        buildSpan.accept(doc.getLuontiPvm());
        break;
    case POSTITETTU:
        buildSpan.accept(doc.getSentDate());
        break;
    case VASTAANOTETTU:
```

```
                buildSpan.accept(doc.getReceivedDate());
                break;
            }

            return sb.toString();
        }

        public Optional<Set<Asiakirja>> getAsiakirjaItems() {
            if (newItems.isEmpty() && oldItems.isEmpty()) {
                return Optional.empty();
            } else {
                Set<Asiakirja> allItems = new HashSet<>();
                allItems.addAll(oldItems);
                allItems.addAll(newItems);
                return Optional.of(allItems);
            }
        }

        public void clearDropzone() {
            root.removeAllComponents();
            root.addComponent(placeholder);
            oldItems.clear();
            newItems.clear();
        }

    }
```

Liite 4 OppariversioTehtavaDropzone.java

```
// Paketin nimi ja importit piilotettu sekä asiakasyritykseen ja toimialaan viittaavat
// luokka-, objekti- ja metodinimet muutettu

@UIScoped
public class YleinenTehtavaWindow extends PWindow {
    @Inject
    private YleinenTehtavaSelector selector;

    @PostConstruct
    public void init() {
        setCaption(_t("yleinentehtava.header"));
        setModal(true);
        setWidth(50, Unit.PERCENTAGE);
        setHeight(50, Unit.PERCENTAGE);
        setContent(selector);
    }

    public void setToimeksianto(Toimeksianto toimeksianto) {
        selector.setToimeksianto(toimeksianto);
    }

    public static class CloseEvent {
        public CloseEvent() {
        }
    }

    public void close(@Observes CloseEvent ce) {
        close();
    }

    public void show() {
        super.show();
    }

}
```