Tatu Hautamäki

SOVELLUS OPPISOPIMUSTEN HANKINTASOPIMUSTEN KÄSITTELYYN

Tietojenkäsittelyn koulutusohjelma

2016

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SOVELLUS OPPISOPIMUSTEN HANKINTASOPIMUSTEN KÄSITTELYYN

Hautamäki, Tatu
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Syyskuu 2016
Ohjaaja: Nieminen, Hans
Sivumäärä: 61
Liitteitä: 2

Asiasanat: ASP.NET MVC, Web, Sovelluskehitys, Oppisopimuskoulutus

---

Opinnäytetyön aiheena oli selainpohjaisen oppisopimusten hankintasopimusten käsittelysovelluksen toteuttaminen Länsirannikon koulutus Oy WinNovalle. Sovellus koostuu tietokannasta, palvelinpuolen logiikasta sekä itse käyttöliittymästä. Sovellus on jo opinnäytetyön kirjoittamisen hetkellä tuotannossa asiakasyrityksessä.

Lopullisesta sovelluksesta tuli alkuperäistä suunnitelmaa huomattavasti laajempi ja monimutkaisempi kokonaisuus. Projekti oli haastava harjoittelijalle, mutta se myös opetti vaatimusmäärittelyn, tietokannan tarkan suunnittelun sekä projektin ja koodimassojen hallinnan tärkeydestä.

Ennen sovelluksen toteuttamista oppisopimusten hankintasopimuksien käsittely ja laskutus suoritettiin kokonaan käsin. Nykyinen järjestelmä on selkeyttänyt ja tehostanut vanhaa, paperisiltojen varaan rakennettua prosessia.

AN APPLICATION FOR HANDLING SALES CONTRACTS FOR APPRENTICE CONTRACTS

Hautamäki, Tatu
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Business Information Systems
September 2016
Supervisor: Nieminen, Hans
Number of pages: 61
Appendices: 2

Keywords: ASP.NET MVC, Web, Software development, Apprentice contracts

---

The purpose of this thesis was to create a browser based application for handling sales contracts for apprentice contracts for Länsirannikon koulutus Oy WinNova. The application consists of database, server side logic and user interface. During the writing of this thesis the application is in daily use at the customer organization.

The final application became much more complex and extensive than the original plan suggested. The project was challenging for a trainee, but it also taught much about the importance of requirement specifications, thorough database design and both project and code management.

Before the application was launched the handling and billing of the sales contracts for apprentice contracts was done wholly by hand and pen. The system that is now in use has enhanced and clarified the old process that was built on paper bridges.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on oppisopimusten hankintasopimusten käsittelysovelluksen suunnittelu ja toteuttaminen. Sovelluksen asiakasyrityksenä ja toimeksiantajana toimii Länsirannikon koulutus Oy WinNova, joka on toisen asteen ammatillista nuorten sekä aikuisten koulutusta tarjoava voittoa tavoittelematon yritys. Suoritin tietojenkäsittelyn opintoihini kuuluvan harjoittelujaksoni samaisessa yrityksessä. Sovelluksen alustava suunnittelu aloitettiin jo aivan harjoittelujaksoni alkupuolella ja se otettiin käyttöön tammikuun 2016 lopulla. Sovelluksen kehitystyö jatkuu vielä opinnäytetyön kirjoittamisen hetkellä.

Hankintasopimusten käsittelyprosessiin kuuluu niiden luomisen ja muokkaamisen lisäksi myös hankintasopimuksen hintaliitteen mukainen laskutus. Vanhan prosessin mukainen laskutus tapahtui ainoastaan paperilaskuilla, joiden laatu vaihteli laidasta laitaan. Koko prosessi toimi suurimmaksi osaksi vain paperilomakkeilla ja laskutuskorteilla, joita arkistoitiin kansioihin sekä verkkoasemille. Erityisesti laskutusprosessin kaoottinen tila oli suurin motivaatio työn alkuunpanolle.

Työssä käydään läpi projektin elinkaari aina määrittelystä suunnitteluvaiheeseen ja lopulta toteutukseen ja käyttöönottoon asti. Lisäksi työ sivuaa hankintasopimusprosessin kehittymistä ja selkeytymistä projektin aikana.

2 YLEISKATSAUS PROSESSISTA

2.1 Yleiskuva prosessista

Oppisopimusten hankintasopimusten käsittelyprosessissa on kyse sekä itse hankintasopimuksen luomisesta että myöhemmin sen laskuttamisesta. Hankintasopimuksessa oppisopimuskoulutusta tarjoava yritys ostaa koulutuksen tarjoajalta koulutuspalveluja oppisopimusopiskelijoiden kouluttamista varten. Käytännössä oppisopimuskoulutuksen tarjoaja ostaa lähes aina koulutusta oman yrityksensä sisältä, jolloin rahat eivät karkaa yrityksen ulkopuolelle. Koulutuksen tarjoaja laskuttaa oppisopimuskoulutusta tarjoavaa yritystä oppisopimusopiskelijoiden kouluttamisesta.

Hankintasopimuksesta käy ilmi muun muassa hankittava koulutus, koulutuksen kesto, yhteyshenkilöiden nimet sekä opiskelijoiden nimet. Hankintasopimukseen liitetään myös liitteitä, joista tärkeimpiä ovat opiskelijoiden henkilökohtaistamisliitteet, yritysyhteystiedot sekä hintaliite. Henkilökohtaistamisliitteet sisältävät tarkempia tietoja opiskelijoista, yritysyhteystiedot puolestaan yrityksestä. Hintaliitteessä käydään läpi suoritettavat tutkinnonosat sekä niiden hinnat. Myöhemmin tapahtuva laskutus tehdään hintaliitteen perusteella. Koulutuksen tarjoaja laskuttaa oppisopimuskoulutuksen ostanutta yritystä joko tasaisesti opiskelijan edetessä koulutuksessa tai kerralla opiskelijan saadessa koulutuksensa tarjoajan osalta loppuun.

Työn kannalta kriittisimpiä toimijoita hankintasopimusprosessissa ovat oppisopimuskeskuksen sihteerit ja koulutusasiantuntijat sekä koulutuksen tarjoajan vastuusihteerit, opettaja sekä koulutuspäällikkö. Tämän työn tapauksessa sekä oppisopimuskeskus että koulutuksen tarjoaja työskentelevät asiakasyrityksen alaisuudessa. Oppisopimuskeskuksen sihteerit ovat vastuussa hankintasopimuksen täyttämisestä, tarkastamisesta ja lopullisesta kirjaamisesta. He myös täyttävät opiskelijakohtaisesti suoritettavat tutkinnonosat hintaliitteeseen. Tämän jälkeen hankintasopimuksesta vastaava koulutuspäällikkö täyttää tutkinnonosien hinnat ja määrät, jotka koulutusasiantuntija vielä tarkastaa ennen kuin hankintasopimus voidaan hyväksyä. Koulutuksen tarjoajan vastuusihteerit laskuttavat hintaliitteen mukaisesti hankintasopimusta. Koulutuspäällikön pitää vielä

tarkastaa lasku, ennen kuin oppisopimuskeskuksen sihteeri voi hyväksyä sen. Hyväksytty ja tarkastettu lasku lähtee eteenpäin laskutusjärjestelmille. Opettajan tehtäväksi jää varsinaisen koulutuksen antaminen sekä suoritteiden merkitseminen StudentaPlus-opintohallintojärjestelmään.

2.2 Vanha prosessi

Paperinen hankintasopimuspohja esitäytetään ja siihen liitetään hintaliite. Tämän jälkeen esitäytetty hankintasopimus lähetetään liitteineen koulutuspäällikölle, opettajalle sekä vastuusihteerille. Hankintasopimus pyydetään palauttamaan takaisin oppisopimuskeskukselle sovittuun päivään mennessä, tyypillisesti aikaa annetaan kahdesta kolmeen kuukautta. Palautuva hankintasopimus sisältää henkilökohtaistamissuunnitelmat ja hintaliitteen hintoineen ja määrineen. Tämän jälkeen se hyväksytetään vielä koulutusasiantuntijan kautta. Oppisopimuskeskuksen sihteerit kirjaavat hyväksytyn hankintasopimuksen SopPro-järjestelmään, jossa ylläpidetään tietoja oppisopimusopiskelijoista. Hankintasopimus skannataan liitteineen, siirretään verkkoasemalle ja linkitetään SopPro:n Sentraali-järjestelmän kautta luotuun hankintasopimukseen. Liittämisen jälkeen hyväksytystä hankintasopimuksesta lähetetään vielä kopio sähköpostilla koulutuspäällikölle. (Oppisopimuskeskus 2015.)

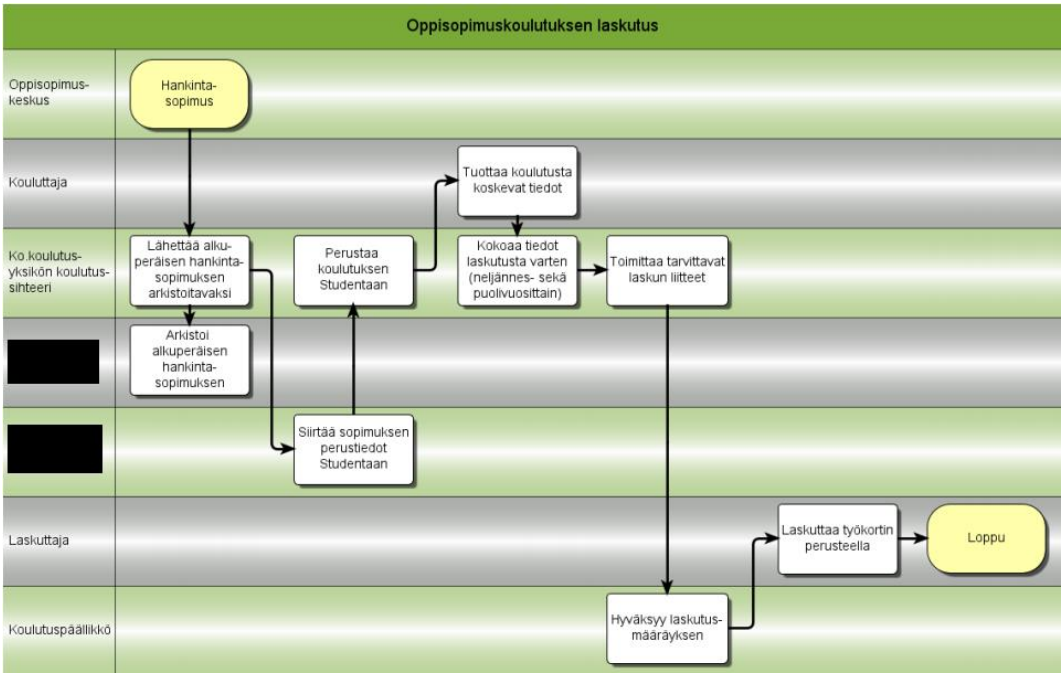
Koulutuksen järjestäjä laskuttaa hintaliitteen mukaan. Samaa hankintasopimusta voi laskuttaa useampikin sihteeri. Lasku voidaan tehdä silloin, kun opettaja on ilmoittanut tutkinnonosan suorittamisesta sihteerille. Laskuttavalla sihteerillä pitää olla hallussaan hankintasopimuksen hintaliite. Laskut tehdään laskutuskorteille, jotka ovat käytännössä Excel-taulukkoja. Laskuun merkitään hintaliitteen mukaisesti tutkinnonosittain hinnat sekä määrät. Valmiit laskutuskortit lähetetään fyysisinä oppisopimuskeskuksen sihteereille, jotka tarkastavat ja siirtävät ne eteenpäin laskutukseen sekä SopPro-järjestelmään. (Oppisopimuskeskus 2015.)

2.3 Vanhan prosessin ongelmat

Vanhan hankintasopimusprosessin ongelmat keskittyivät lähinnä hintaliitteen mukaiseen laskuttamiseen. Kaikki laskuttajat eivät tarkastelleet hintaliitettä laskua tehdessään, vaan joko heittivät summittaisesti hintoja satunnaisiin tutkinnonosiin tai ilmoittivat vain laskun kokonaissumman. Hinnat ja määrät saattoivat olla myös täysin ristiriidassa sovittujen summien kanssa. Tällaisissa tapauksissa laskutuskortin tarkastajalla ei ollut muuta vaihtoehtoa kuin ottaa suoraan yhteyttä laskuttajaan. Tämä kuormitti huomattavasti oppisopimuskeskuksen työntekijöitä ja hidasti rahojen saapumista koulutusaloille. (Oppisopimuskeskus 2015.)

Syitä hintaliitteen tarkastelemattomuudelle ei saatu koskaan selville. Todennäköisesti hintaliitteen hankala sijainti verkkoasemalla sekä vain oppisopimuskeskuksen käytössä olevalla SopPro-järjestelmässä hankaloitti hintaliitteen nopeaa tarkastelua laskuttajan päässä. Lisäksi prosessin kokonaiskuva oli monelle koulutuksen tarjoajalle epäselvä: miksi hintojen ilmoittaminen tutkinnonosittain oli niin tärkeää?

Laskutusprosessista ei myöskään löytynyt selkeää, oikean elämän tilannetta kuvaavaa kuvausta asiakasyrityksen dokumentinhallintajärjestelmästä. Seuraavan sivun kuva (Kuva 1) on ainoa saatavilla oleva kuvaus, joka epätoivoisesti yrittää kuvata vanhan prosessin tilaa. Prosessi etenee vasemmalta oikealle ja jokaisella prosessin toimijalla on oma ratansa.



Kuva 1. Oppisopimuskoulutuksen vanha laskutusprosessi. Huomaa laskutusprosessiin kuulumattomat osuudet, esimerkiksi ”Perustaa koulutuksen Studentaan”. (WinNova 2012.)

Laskuttamisen kaoottinen tila oli työn alkuun paneva voima. Varsinkin oppisopimuskeskuksen sihteerit, joiden tehtävänä on siis kirjata ja tarkastaa laskuja, olivat todella kyllästyneitä vanhaan laskutusprosessiin.

3 KÄYTETTÄVÄT TEKNIIKAT

3.1 ASP.NET

ASP.NET on Microsoftin web-tuotantoalusta, jonka avulla ohjelmistokehittäjä voi tuottaa web-sovelluksia suhteellisen vähällä määrällä koodia. Web-sovelluksia voidaan luoda millä tahansa CLR:n (Common Language Runtime) kanssa yhteensopivalla kielellä, esimerkiksi Microsoftin omilla C# tai Visual Basic –ohjelmointikielillä. (Microsoft, 2016a.)

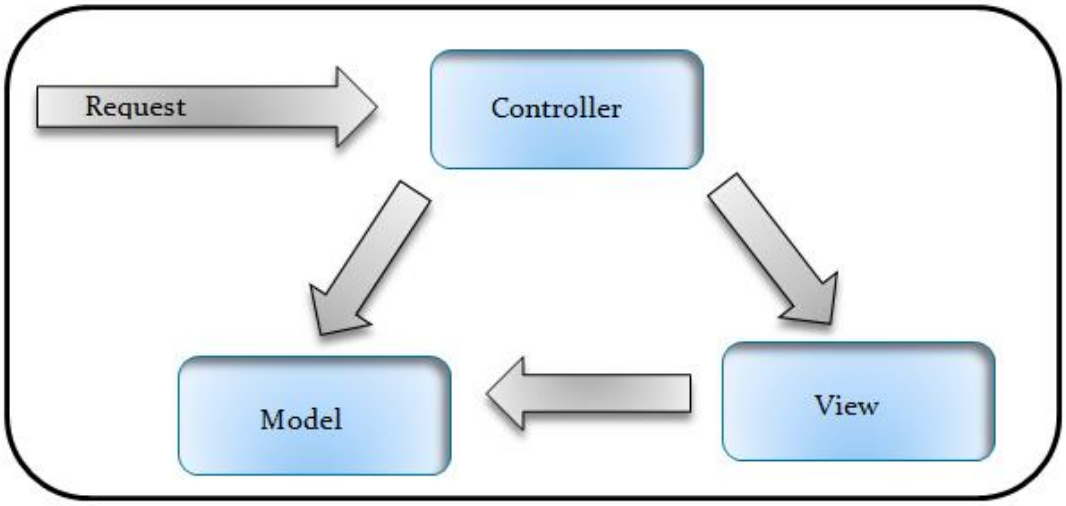
3.1.1 ASP.NET MVC

ASP.NET MVC on arkkitehtuurimalli, joka kuuluu Microsoftin ASP.NET web-tuotantoalustaperheeseen. Se perustuu norjalaisen Trygve Reenskaugin jo vuonna 1979 luomaan Model-View-Controller -malliin. MVC-arkkitehtuurimallissa pyritään eriyttämään käyttöliittymä, sovelluslogiikka ja -data jotta sovelluskokonaisuuden hallinta ja ylläpito helpottuisi. (Koskimies & Mikkonen 2005, 142.)

MVC jakautuu nimensä mukaisesti kolmeen osaan: malliin (model), näkymään (view) ja ohjaimeen (controller). Tietokantaa käyttävät mallit sisältävät sovelluksen logiikkaa, validointia ja toimintoja. Mallien tehtävänä on huolehtia niin tiedon hakemisesta kuin päivittämisestäkin tietokantaan. (Koskimies & Mikkonen 2005, 142; Microsoft, 2016b.)

Näkymät toimivat sovelluksen käyttöliittyminä ja ne luodaan yleensä niihin liitetyn mallin datan perusteella. ASP.NET MVC:n versiosta 3 eteenpäin näkymien piirtämiseen on käytetty Razor-näkymämoottoria. Itse näkymissä käytetään Razor-syntaksia, jonka avulla niihin voidaan kirjoittaa HTML-koodia dynaamisesti. Tämä koodi ajetaan palvelimella ennen kuin sivu piirretään loppukäyttäjälle. Lisäksi Razor tarjoaa useita HTML-apumetodeja kehittäjän käyttöön, joilla mallista generoitavaa HTML-sisältöä voidaan luoda minimaalisella määrällä koodia. Näkymämoottori mahdollistaa myös osittaisten näkymien (Partial View) luonnin, joita voidaan upottaa varsinaisiin näkymiin. (Microsoft, 2016b; Microsoft, 2016c.)

Ohjaimet käsittelevät käyttäjän syötteitä ja pyyntöjä ja määrittelevät, mikä näkymä käyttäjälle palautetaan tai miten käyttäjän lähettämä syöte käsitellään. Näkymän mahdollisesti tarvitsema malli luodaan ja liitetään palautettavaan näkymään ohjaimessa. Ohjaimien ei kuitenkaan ole pakko palauttaa vain näkymiä, vaan ne voivat myös esimerkiksi palauttaa osittaisia näkymiä tai JSON-dataa. (Microsoft, 2016b.) Seuraavan sivun kuvassa (Kuva 2) havainnollistetaan MVC:n toimintaperiaatetta.



Kuva 2. MVC:n toimintaperiaate. Käyttäjän pyyntö menee ensin ohjaimelle, joka ottaa yhteyttä oikeaan malliin. Malli palauttaa tarvitun tiedon, mikä näytetään käyttäjälle näkymässä. (Codeproject, 2016.)

3.1.2 Global.asax

Global.asax on myös muissa ASP.NET-perheen tuotteissa esiintyvä tiedosto, jossa määritellään sovelluksen tapahtumien hallintaa. Mahdollisia tapahtumia ovat esimerkiksi sovelluksen käynnistyminen (Application_Start) sekä käyttäjän pyynnön vastaanottaminen (Application_BeginRequest). Global.asax ei kuitenkaan ole pakollinen, vaan sen puuttuessa ASP.NET olettaa tapahtumien hallinnan puuttuvan kokonaan. (Microsoft, 2016d.)

ASP.NET MVC -sovelluksissa Global.asax-tiedostossa alustetaan sovelluksen reititys (routing), niputus (bundling) sekä suodatus (filtering).

3.1.3 Reititys

Reitityksen pääajatuksena on se, että web-sovelluksen URL-osoitteet eivät viittaisi suoraan tiedostoihin vaan ohjaimiin. URL-osoitteiden selkeytyminen on miellyttävämpää niin loppukäyttäjälle kuin kehittäjällekin. Reititys voidaan määrittää joko suoraan Global.asax–tiedostossa tai erillisessä RouteConfig.cs-tiedostossa minkä sisältö alustetaan reititystaulukoksi Global-asax-tiedostossa. (Microsoft, 2016e.)

Reititys toimii URL-kaavojen hyödyntämisen kautta. Kaavoissa voidaan käyttää joko suoraan ohjaimien, toimintojen tai tiedostojen nimiä, mutta tyypillisesti niissä käytetään paikanosoittajamääritteitä (placeholder). Esimerkiksi {controller}/{action} viittaa kaikkiin web-sovelluksen ohjain-toiminto-yhdistelmiin muodossa ohjain/toiminto. Lisäksi URL-parametreja voidaan tarvittaessa naamioida reititysmäärittelyjen kautta. Tätä hyödyntäessä paikanosoittajamääritteen on oltava samanniminen, kuin naamioitavan parametrin nimi ohjaimen toiminnossa. (Microsoft, 2016e.) Alla olevassa kuvassa (Kuva 3) esitellään paikanosoittajamääritteiden käyttöä.

```
public class RouteConfig
{
    1 reference | Korpimursu, 55 days ago | 2 changes
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
        routes.MapRoute(
            name: "AvaaLaskuSuoraan",
            url: "{controller}/{action}/{id}/{laskuid}",
            defaults: new { id = UrlParameter.Optional, laskuid = UrlParameter.Optional }
            );

        routes.MapRoute(
            name: "Actions",
            url: "{controller}/{action}/{id}",
            defaults: new { id = UrlParameter.Optional }
            );

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}",
            defaults: new { controller = "OmaHankintasopimus", action = "Index" }
        );
    }
}
```

Kuva 3. Työn web-sovelluksen RouteConfig.cs-tiedoston sisältö. AvaaLaskuSuoraan-reitissä määritellään kaksi parametria paikanosoittajamääritteillä. Default-niminen reitti on sovelluksen aloitussivu.

3.1.4 Suodatus

Ohjaimiin kohdistuviin pyyntöihin voidaan sitoa erilaisia toimintoja suodattimilla. Nämä toiminnot voidaan määrittää laukeamaan erilaisissa vaiheissa. Esimerkiksi kehittäjä voi halutessaan tutkia ja käsitellä pyyntöä ennen kuin ohjain käsittelee sen. (Microsoft, 2016f.)

ASP.NET MVC sisältää oletuksena suodattimen virheiden käsittelyä varten. Kyseisellä suodattimella ohjaimissa tapahtuvat virheet voidaan ohjata omalle virhesivulleen. Näin loppukäyttäjän ei virheen sattuessa tarvitse katsella perinteistä ASP.NET:in keltataustaista ja uhkaavaa virheilmoitusta, vaan kehittäjä voi määritellä sovellukselleen miellyttävämmän virhesivun. (Microsoft, 2016f.)

3.1.5 Niputus

Niputuksella voidaan luoda CSS- ja JavaScript-tiedostoista nimettyjä kokonaisuuksia. Web-sovelluksissa sekä tyyli- että skriptitiedostoja voi olla useita kymmeniä ja niillä voi olla riippuvuuksia toisiinsa. Niputtamalla tiedostot yhtenäisiksi kokonaisuuksiksi niiden hallinnasta tulee huomattavasti helpompaa. Niputusmäärittelyssä voidaan käyttää niin paikanosoittajamääritteitä kuin jokerimerkkejäkin (wildcard). Esimerkiksi {version}-paikanosoittajamäärite viittaa JavaScript-tiedoston versionumerointiin ja ”*”-merkillä voidaan viitata kansion kaikkiin CSS-tiedoistoihin ”*.css”-määritteellä. (Microsoft, 2016g.) Niputuksen käyttöä havainnollistetaan alla olevassa kuvassa (Kuva 4).

```
1 reference | Korpimursu, 94 days ago | 1 change
public class BundleConfig
{
    1 reference | Korpimursu, 94 days ago | 1 change
    public static void RegisterBundles(BundleCollection bundles)
    {
        bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
                    "~/Scripts/jquery-{version}.js"));

        bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
                    "~/Scripts/jquery.validate*"));
        // koodi katkaistu selvyyden vuoksi
```

Kuva 4. Työn web-sovelluksen BundeConfig.cs-tiedoston sisältö. Paikanosoittajamääritteellä viitataan jquery.js-skriptin mihin tahansa versioon ja jokerimerkillä puolestaan kaikkiin tiedostoihin, joiden nimet alkavat muodossa jquery.validate.

3.2 Suunnittelumallit

Ohjelmistotuotannossa suunnittelumalleilla tarkoitetaan luokkakokonaisuuksien jäsentämistä yhtenä suurempana kokonaisuutena. Käytännössä mallit perustuvat hyviksi havaittuihin ja useasti toistuviin ratkaisuihin ohjelmistosuunnittelussa. Kun näitä ratkaisumalleja on mallinnettu ja dokumentoitu yhtenäiseen muotoon, niistä on muodostunut yleisesti ymmärrettyjä ja tiedostettuja suunnittelutapoja. (Haikala & Märijärvi 2001, 345.)

Suunnittelumallit voidaan jakaa käyttötarkoituksensa ja näkyvyysalueensa mukaan. Käyttötarkoitukset jakautuvat kolmeen ryhmään: rakenteellisiin malleihin, käyttäytymismalleihin sekä luomismalleihin. Rakenteelliset mallit keskittyvät nimensä mukaisesti olio- tai luokkakokonaisuuksien rakenteisiin, käyttäytymismallit puolestaan olioiden ja luokkien väliseen kanssakäymiseen ja vastuunjakamiseen. Luomismalleissa olioiden luontiprosessi on tärkeimmässä roolissa. Näkyvyysalueet jakautuvat puolestaan siihen, koskeeko suunnittelumalli olioita vai luokkia. Oliomallit keskittyvät olioiden välisiin dynaamisiin suhteisiin, kun taas luokkamallit käsittelevät luokkien välisiä staattisia suhteita ja hierarkioita. Suurin osa suunnittelumalleista on näkyvyysalueeltaan oliomalleja. (Gamma, Helm, Johnson & Vissides 2009, 31-32.)

Harkinnanvarainen suunnittelumallien käyttö mahdollistaa hyväksi todettujen ja toimivien ratkaisumallien uudelleenhyödyntämistä ohjelmistojen suunnitteluvaiheessa. Ne helpottavat myös ohjelmistojen osien uudelleenkäyttöä ja uusien ominaisuuksien käyttöönottoa. Myös dokumentointi ja myöhempi ohjelmiston ylläpito helpottuvat huomattavasti. (Gamma ym. 2009, 21.)

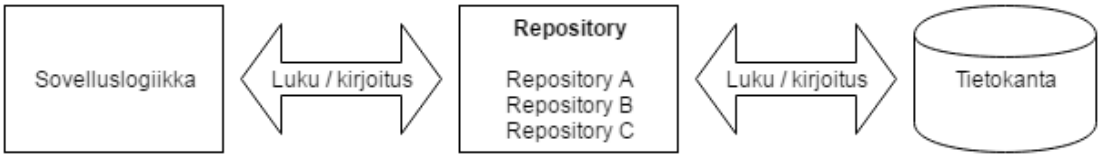
Suunnittelumallien käytössä esiintyy myös vaaroja. Suurin osa malleista moninkertaistaa sovellukseen sisältyvien luokkien ja rajapintojen määrän. Yksinkertaisista ja helposti hahmotettavista kokonaisuuksista saattaa näin tulla tarpeettoman monimutkaisia. Suunnittelumallien käyttö saattaa myös jäädä puolitiehen tai unohtua kokonaan kehitysvaiheen aikana, jos niiden käyttöä ei dokumentoida tarkkaan. (Koskimies & Mikkonen 2005, 117-118.)

Molemmat seuraavaksi läpikäytävistä suunnittelumalleista kuuluvat rakenteellisiin oliomalleihin.

3.2.1 Repository

Repository-suunnittelumallilla pyritään erottamaan tiedonsaantitaso varsinaisesta sovelluksesta, esimerkiksi tietokantatason erottamiseen sovellustasosta. Suunnittelumallissa tietokannasta lukeminen ja kirjoittaminen on luovutettu erityisille luokille, joissa määritellään, miten ja mitä tietoa kannasta haetaan sekä missä muodossa tieto luovutetaan eteenpäin. (Microsoft, 2016h.)

Suunnittelumallin käyttöä perustellaan sillä, että suoraan sovelluskoodin sekaan kirjoitetut tietokantakyselyt sisältävät usein toistoa, josta seuraa virhemahdollisuuksien kasvaminen. Myös kokonaisuuden hallitsemisesta ja ylläpidosta tulee vaikeampaa hajanaisuuden takia. Lisäksi vahva tyypitys saattaa kärsiä, jos ohjelmoija ei välitä missä muodossa ja miten haettu tieto käsitellään. (Microsoft, 2016h.) Alla olevalla kuvalla (Kuva 5) havainnollistetaan repository-suunnittelumallin periaatetta.

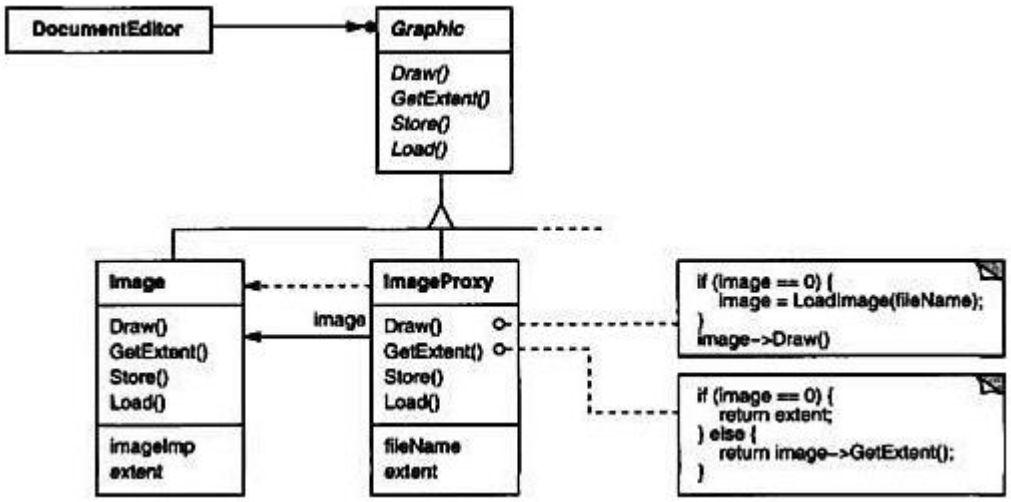


Kuva 5. Yksinkertaistettu kuvaus repository-suunnittelumallista.

3.2.2 Proxy

Proxy-suunnittelumallissa haluttuja olion ominaisuuksia määritellään käytettäväksi oliosta periytetyn proxy-toteutuksen kautta. Tämä mahdollistaa sen, että olioon ei tarvitse sen luontivaiheessa hakea kaikkea tietoa valmiiksi: tiedot haetaan vasta siinä vaiheessa, kun oliolta oikeasti pyydetään niitä. Proxy-luokassa määritellään, miten ja mistä pyydetty tieto haetaan ja mitä olion osia käytetään proxy-toteutuksen kautta. (Gamma ym. 2009, 223-224.)

Suunnittelumallin käyttöä perustellaan sen tehokkuudella. Olioiden luomisesta tulee huomattavasti kevyempää, kun kaikkea tietoa ei tarvitse hakea sen luontivaiheessa. Esimerkiksi oliolla voi olla useita ominaisuuksia, joiden paluuarvo perustuu tietokannasta tai muusta tietolähteestä saatuun dataan. Jos oliolle haetaan nämä tiedot aina kun se luodaan, niin resursseja kuluu turhaan todennäköisesti tarpeettomalle tiedolle. Yksi tapa korjata edellä mainittu ongelma proxy-toteutuksella on varustaa kaikki olion ulkopuolelle viittaavat ominaisuudet virtual-määritteellä ja proxy-luokassa yliajaa nämä ominaisuudet hakemaan tieto tietolähteestä. Tätä proxyn toteutustapaa kutsutaan virtual proxyksi. (Gamma ym. 2009, 224-226.) Seuraavassa kuvassa (Kuva 6) esitellään luokkakaavion avulla yhtä proxy-suunnittelumallin toteutustapaa.



Kuva 6. Luokkakaavio proxy-suunnittelumallin toteutuksesta. Kuva haetaan kovalevyltä vasta sitten, kun käyttäjä sitä pyytää. Proxy-luokka tietää fileName-kentän perusteella, mikä tiedosto levyltä kuuluu noutaa. (Gamma ym. 2009, 226.)

3.3 jQuery

JavaScript-koodin kirjoittaminen on hyvin usein samojen toimintojen ja ratkaisujen toistamista. Suurin osa koodista koostuu HTML-elementeille kohdistuvista toiminnoista, joissa esimerkiksi haetaan elementtien arvoja, muokataan niiden attribuutteja tai piilotetaan niitä käyttäjältä. jQuery-kirjasto pyrkii helpottamaan tätä rutiininomaista työtä intuitiivisemmalla syntaksilla ja valmiilla funktioilla. (McFarland 2008, 169.)

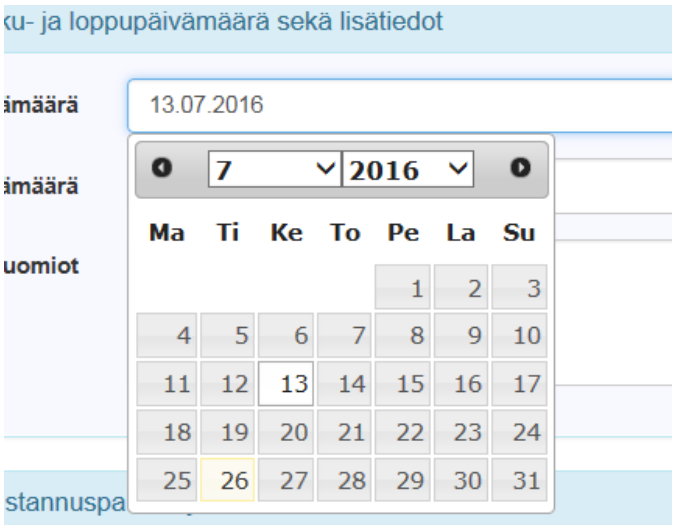
Yksi huomattavimmista parannuksista perinteiseen JavaScriptiin on valitsinsyntaksin (selector) käyttö. Valitsinsyntaksilla HTML-elementtiä voidaan hakea helposti joko itse elementtinä tai sen id:n, luokan tai attribuutin mukaan. Lisäksi sillä voidaan ketjuttaa useampia hakuehtoja yhteen hakuun. (McFarland 2008, 173-175.) Alla olevassa kuvassa (Kuva 7) esitellään valitsinsyntaksin käyttöesimerkkejä.

```
$('elementti'); //haetaan elementti
$('#id'); //haetaan id-arvon perusteella
$('.luokka'); //haetaan luokan perusteella
$('elementti[attribuutti="arvo"]'); //haetaan elementin attribuutin arvon perusteella
$('elementti#id'); //haetaan tietyn elementin id-arvon perusteella
$('select option:selected'); //haetaan select-elementin valittu option-elementti
```

Kuva 7. jQueryn valitsinsyntaksin käyttöesimerkkejä. Funktiosta palautuu jQuery-objekti, johon voidaan ketjuttaa edelleen jQuery-funktioita.

3.3.1 jQuery UI

jQuery UI (User Interface) on jQuery-kirjaston lisäosa, joka on erikoistunut käyttöliittymätoimintoihin. Se koostuu jQuery-kirjastoon tukeutuvista erilaisista käyttöliittymätoiminnoista, tehosteista ja teemoista. (jQuery UI, 2016.) Yksi käytetyimmistä jQuery UI:n tarjoamista komponenteista on interaktiivinen kalenteri, jota alla oleva kuva (Kuva 8) esittelee tarkemmin.



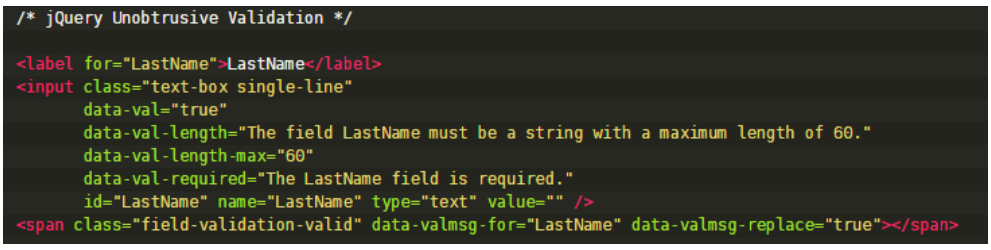
Kuva 8. jQuery UI:n kalenterikomponentti suomalaisilla kalenteriasetuksilla. (Nieminen 2015.)

3.3.2 jQuery Unobtrusive Ajax ja Validation

jQuery Unobtrusive Ajax on Ajaxin tehokkaampaan käyttämiseen tarkoitettu jQueryn lisäosa. Se tarjoaa HTML-elementtien käyttöön data-attribuutteja, joilla voidaan määrittää Ajaxin toimintoja ilman tapahtumasidontaa JavaScript-puolella. Data-attribuutit voidaan antaa HTML-elementeille joko käsin tai käyttämällä Razor-näkymämoottorin tarjoamia Ajax-apumetodeja. (Wilson, 2010a.) Seuraavassa kuvassa (Kuva 9) näytetään esimerkki Ajax.BeginForm-apumetodin käytöstä.

Kuva 9. Apumetodista muodostuva Ajax-form.

Unobtrusive Validation on selainpohjaiseen validointiin tarkoitettu jQueryn lisäosa. Sen avulla voidaan vähentää huomattavasti tarvittavien validointielementtien ja -attribuuttien määrää. Unobtrusive Validation toimii Unobtrusive Ajaxin tapaan data-attribuuteilla, joten elementteihin on mahdollista määrittää tarvittavat attribuutit myös käsin. HTML-apumetodeja käyttämällä mallista voidaan johtaa siihen liitettyjä validointioattribuutteja automaattisesti. (Wilson, 2010b.) Alla olevassa kuvassa (Kuva 10) näytetään esimerkkejä Unobtrusive Validationin attribuuttien käytöstä.



Kuva 10. Unobtrusive Ajaxin data-val –attribuuttien käyttö. (Wilson, 2010b.)

3.4 Ajax

Ajax (Asynchronous JavaScript and XML) on asynkronista web-sivujen päivittämistä varten luotu tekniikka. Siinä missä perinteisesti palvelin palauttaa kokonaisen HTML-sivun selaimen sitä pyytäessä, Ajaxilla voidaan hakea tarvittava tieto palvelimelta erikseen ja päivittää se osaksi aktiivista HTML-sivua. Tämä mahdollista huomattavasti käyttäjäystävällisemmän kokemuksen sekä vähentää palvelimelle kohdistuvaa kuormaa, koska palvelimelta ei tarvitse hakea kokonaisia sivuja. (Pars, Moroney & Grieb 2007, 7-8.)

jQuery sisältää useita funktioita Ajaxin helpompaa käyttöä varten. Näistä kaikkein muokattavimmissa oleva funktio on jQuery.ajax(), jonka avulla kehittäjä pystyy tarkemmin määrittelemään esimerkiksi Ajax-kutsun asetuksia. Tätä funktiota ei kuitenkaan tarvitse käyttää aina, vaan tarjolla on myös yksinkertaisempia, ylemmän tason funktioita. Esimerkiksi jQuery.getJSON() (hakee palvelimelta suoraan tietotyyppinä JSON) ja jQuery.load() (palauttaa haetun datan suoraan callback-funktiona). jQueryn Ajax-kutsuihin on myös helppo lisätä erilaisten tapahtumien hallintaa, esimerkiksi silloin kun tiedon hakeminen alkaa tai tieto on saatu kokonaisuudessaan haettua palvelimelta. (jQuery 2016.)

3.5 Modernizr

Web-sovelluksia käytetään selaimilla, joiden päivittäminen on täysin loppukäyttäjän vastuulla. Lisäksi jopa modernien selainten kesken on huomattaviakin eroja siinä, miten ne käsittelevät JavaScriptiä, CSS-määrityksiä ja HTML-koodia. Modernizr on JavaScript-pohjainen kokoelma erilaisia yhteensopivuustestejä, joilla kehittäjä voi testata sovelluksensa ominaisuuksien toimivuutta käyttäjän selaimessa. Tämä mahdollistaa kaikkien ominaisuuksien toimivuuden myös niihin kykenemättömissä selaimissa. (Modernizr, 2016.)

Ehkä yksi käytetyimpiä esimerkkejä on HTML5-standardin mukaisen date-attribuutilla varustetun input-elementin yhteensopivuus nykyaikaisissa selaimissa. Kysei-

sen elementin pitäisi tuottaa käyttäjälle kalenteri, jolla hän voi valita haluamansa päivämäärän. Elementti ei kuitenkaan toimi uusimmillakaan Internet Explorerin, Firefoxin, Safarin eikä Opera Mini:n selaimilla (Can I Use, 2016). Kehittäjä voi tarkastaa selaimen yhteensopivuuden esimerkiksi alla olevan kuvan osoittamalla tavalla (Kuva 11) ja toteuttaa ongelmallisille selaimille kalenterin jQuery UI:n avulla.

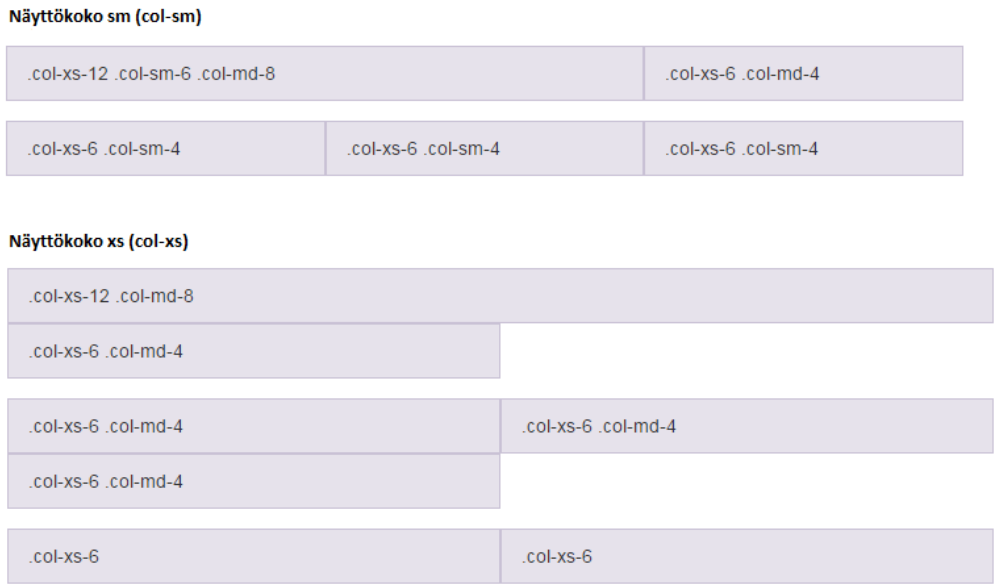
```
if (!Modernizr.inputtypes.date) {
    $('input[type="date"]').each(function () {
        var teksti = $(this).val();
        var osat = teksti.split('-');
        if (osat.length === 3) {
            $(this).val(osat[2] + '.' + osat[1] + '.' + osat[0]);
        }
    });
    $(":input[data-datepicker]").datepicker($.datepicker.regional['fi']);
} else {
    console.log('Selain tukee date-inputtia.')
}
```

Kuva 11. Esimerkki Modernizr-yhteensopivuustestin käytöstä. Testissä tarkastetaan, tukeeko selain date-attribuutilla varustettua input-elemettiä ja tehdään tarpeelliset toiminnot tuen puuttuessa.

3.6 Bootstrap

Bootstrap on suosituin ilmainen käyttöliittymäpään (front-end) kehys vuorovaikutteisten erikokoisille näytöille skaalautuvien web-sovellusten tekemiseen. Se muodostuu JavaScriptistä (jQuery), valmiista pienkuvakkeista (glyphicon) ja CSS-tyylisivujen kokonaisuudesta. Bootstrap vaatii jQueryn kirjaston toimiakseen. (Rascia 2015.)

Bootstrap käyttää elementtien asettelussa niin sanottua grid-järjestelmää, jolla sivun jokainen rivi jaetaan kahteentoista osaan. Elementin class-attribuuttiin annettavien col-alkuisten luokkien arvo kertoo grid-järjestelmälle, kuinka paljon tilaa ne kahdestatoista osasta vievät. Lisäksi col-luokille annetaan vielä näyttökoon mukainen määritys (xs, sm, md ja lg), jolloin elementti voi viedä eri määrän osia gridistä näyttökoon pienentyessä tai kadota kokonaan näkyvistä. Jos elementti ylittää gridin kahdentoista osan maksimiarvo, se sijoittuu seuraavalle riville. (Bootstrap, 2016a.) Seuraavan sivun kuvassa (Kuva 12) esitellään grid-järjestelmän toiminnallisuutta.



Kuva 12. Bootstrapin grid-järjestelmä näyttökoilla sm (small) ja xs (extra small). Samat elementit skaalautuvat eri tavalla näytön pienentyessä. (Bootstrap, 2016a.)

Grid-järjestelmän lisäksi Bootstrap tarjoaa kehittäjälle useita erilaisia tyylejä HTML-elementtien ulkoasun muokkaamiseen. Se myös sisältää muun muassa valmiin navigaatiovalikkomallin sekä jQueryn kautta periytyviä ominaisuuksia, esimerkiksi pakolliset ikkunat (modaalit) ja accordion-komponentit. (Rascia, 2015.)

4 OHJELMISTOSUUNNITTELU

4.1 Vaatimusmäärittely

Ohjelmistotuotannossa vaatimusmäärittelyä kuvaillaan useilla eri tavoilla. Käytännössä vaatimusmäärittely aloitetaan jo ennen varsinaisen projektin alkua määrittelemällä ensiksi projektin tarpeellisuus ja toteutuskelpoisuus. Alkumäärittelyn jälkeen projektille asetetaan tavoitteet ja vaatimukset sekä lopullinen ratkaisumalli. Tästä kokonaisprosessista syntyy toiminnallinen määrittely, josta käy ilmi sekä itse vaatimukset että ne täyttävän järjestelmän kuvaus. (Haikala & Märijärvi, 2001 66.) Huolellinen vaatimusmäärittely on edellytys onnistuneelle ohjelmistoprojektille. (Haikala & Mikkonen, 2011 61.)

Vaatimukset voidaan jakaa karkeasti kolmeen eri luokkaan: toiminnallisiin vaatimuksiin, ei-toiminnallisiin vaatimuksiin ja reunaehtoihin. Toiminnallinen vaatimus on järjestelmän toimintaan liittyvä vaatimus, esimerkiksi ”järjestelmä vahtii, etteivät hintaliitteen summat ylity”. Ei-toiminnallinen vaatimus ei liity järjestelmän varsinaiseen toimintaan, vaan esimerkiksi sen dokumentointiin tai yleiseen ulkoasuun. Reunaehdot ovat järjestelmän toteutuksen ehtoja. Ne ovat tyypillisesti teknisiä ja rajoittavat esimerkiksi järjestelmän toteutustekniikkaa. (Haikala & Mikkonen, 2011 61.)

Hyvän vaatimuksen ominaisuuksia ovat tarkkuus ja ymmärrettävyys, testattavuus sekä taaksepäin ja eteenpäin jäljitettävyys. Tarkkuus ja ymmärrettävyys takaavat sen, että vaatimuksen täyttymisen ehdot eivät ole epäselviä kummallekaan projektin osapuolelle. Testattavuudella tarkoitetaan sitä, että vaatimuksen täyttyminen pitää pystyä todentamaan. Vaatimuksen jäljitettävyydellä pyritään selvittämään sekä vaatimuksen alkuperä että sen lopullinen tekninen toteutus. (Haikala & Mikkonen, 2011 64.)

Yleisesti ottaen vaatimusmäärittelydokumenttien tulisi olla mahdollisimman tarkkoja ja selkeitä. Vaatimusmäärittelyn taso ja laatu riippuvat kuitenkin täysin projektin osapuolista ja itse kehitettävästä järjestelmästä. Jos asiakas käyttää organisaation ulkopuolista toimittajaa monimutkaiselle ja kalliille järjestelmälle, määrittelydokumenteista pyritään tekemään mahdollisimman tarkkoja ja kuvaavia. Organisaation sisäisen toimittajan tapauksessa dokumenteista voidaan tehdä tarvittaessa löyhempiä, jolloin mahdollisia epäkohtia voidaan helposti korjailla projektin edetessä. (Sommerville, 2004 136.)

Tarkkaan määritellyt ja lukitut vaatimukset ovat hankalia ketterän ohjelmistokehityksen kannalta, jonka mukaan vaatimukset elävät jatkuvasti. Projektin alussa laadittavien dokumenttien sanotaankin olevan vanhentuneita jo heti niiden valmistuessa. Esimerkiksi Extreme Programming (ketterä ohjelmistokehitystekniikka, joka keskittyy pariohjelmointiin) ehdottaa käytettäväksi vaatimuskortteja, joihin kerätään asiakkaan ehdottamia vaatimuksia projektin edetessä. Vaatimuskortit asetetaan yhdessä asiakkaan kanssa tärkeysjärjestykseen projektin seuraavassa iteraatiovaiheessa. (Sommerville, 2004 138.) Kunnollisen dokumentoinnin puute saattaa kuitenkin myös aiheuttaa sen,

että työlistaan tai vaatimuskortteihin kirjoitetut asiat unohtuvat tai kadottavat alkuperäisen merkityksensä. Kehittäjä ja asiakas saattavat muistaa vaatimuksen täysin eri tavalla. Huolellista vaatimusten dokumentointia ei siis kannata väheksyä ketterissäkään menetelmissä. (Haikala & Mikkonen, 2011 68.)

4.2 Tietokannan suunnittelu

Tietokannan suunnittelu mielletään usein vain tietokannan mallintamiseksi. Se koostuu kuitenkin myös tietokannan vaatimusmäärittelystä ja käytännön suunnittelusta. Vaikka loppukäyttäjä näkeekin sovelluksesta vain lopullisen käyttöliittymän, tietokannan huolellista suunnittelua ei kannata väheksyä. Kehittäjä joutuu maksamaan kovaa hintaa myöhemmin sovelluksen kehitysvaiheessa, jos taustalla toimiva tietokanta on huonosti suunniteltu ja kiireessä luotu. Näin tietokannan suunnittelussa säästetty aika maksetaan vielä moninkertaisesti sovelluksen kehitys- ja ylläpitovaiheessa. Perussääntönä voidaan pitää sitä, että mitä monimutkaisempi ja suurempi tietokanta on, sitä huolellisemmin se tulisi suunnitella. (Hovi, Huotari & Lahdenmäki, 2005 20.)

Hyvän tietokannan keskeisiä rakenteellisia ominaisuuksia ovat:

- Se sisältää kaikki kyselyissä tarvittavat tiedot.
- Tietokannan rakenteen pitäisi olla mahdollisimman yksinkertainen ja ilmaisuvoimainen.
- Tietokantaa pitää pystyä laajentamaan ja muuttamaan mahdollisimman joustavasti. Muutostilanteissa tulee välttää epämääräisten aputaulujen ja erikoisviritysten käyttöä.
- Sitä pitää pystyä soveltamaan erilaisissa ympäristöissä ilman suuria rakenteellisia muutoksia.
- Tiedon ja rakenteen pitää olla oikeellista ja ristiriidatonta. Tiedon toisteisuutta pitää välttää. Virhesyötteet eivät saa rikkoa tietokannan oikeellisuutta.
- Tietorakenteiden tulee olla selkeitä ja sarakkeet eivät saa olla riippuvaisia toisista sarakkeista.
- Nopeat vastausajat ja tehokkaat eräajot.

(Hovi, Huotari & Lahdenmäki, 2005 21-23.)

Itse suunnittelu voidaan jakaa kolmeen eri vaiheeseen: käsiteanalyysiin, tarveanalyysiin ja normalisointiin. Käsiteanalyysissä tietokantaa hahmotellaan loogisella tasolla. Hahmotelmasta luodaan käsitemalli, joka toimii toimittajan ja asiakkaan välisenä yhteisenä mallina. Käsitemalli ei siten ole vielä kovin tekninen, vaan se pyrkii kuvaamaan tietokannan rakennetta karkealla ja yleisellä tasolla. Tarveanalyysissä ensimmäisessä vaiheessa luotua käsitemallia tarkennetaan tietoteknisemmälle tasolle. (Hovi, Huotari & Lahdenmäki, 2005 24-33.)

Normalisoinnissa tietokannan tietorakenteita pyritään jalostamaan parempaan ja tehokkaampaan muotoon. Käytännössä tämä tarkoittaa rakenteen osalta sitä, että tietojen toistamista on minimoitu, niiden päivittäminen on tehokasta ja että tiedot tarvitsee päivittää vain yhteen paikkaan. Normalisoinnissa taulujen normalisaatiotason kuvaamiseen käytetään normaalimuoto-termiä (NF, Normal Form). Taulut eivät voi olla seuraavassa normaalimuodossa ennen kuin ne ovat täyttäneet myös edellisten normaalimuotojen ehdot. Tyypillisesti tietokantasuunnittelussa tähdätään vähintään kolmannen normaalimuodon normalisointiin asti. (Hovi, Huotari & Lahdenmäki, 2005 86.)

Jotta taulut olisivat ensimmäisessä normaalimuodossa (1NF), niiden sarakkeissa ei saa olla moniarvoista eikä toistuvaa tietoa. Sarakkeeseen ei siis saa tallentaa esimerkiksi pilkulla eroteltuja arvoja eikä rivillä saa olla omaa saraketta jokaiselle toistuvalle tiedolle. (Hovi, Huotari & Lahdenmäki, 2005, 87-88.) Tyypillisiä ensimmäisen normaalimuodon rikkomistapauksia esitellään alla (Kuva 13).

di	linjakoodit	koul
	155,860,888,266,191,192	5
	NULL	5
	191,192	5
	28	NUI

n	julkinen_1	julkinen_2	julkinen_3	julkinen_4	julkinen_5	julkinen_6
	1	1	1	1	1	1
	1	1	1	1	1	1
	0	0	0	0	0	0

Kuva 13. Esimerkki ensimmäisen normaalimuodon rikkomisesta. Ylin taulu rikkoo moniarvoisen tiedon sääntöä, alin taulu puolestaan toistuvan ryhmän sääntöä.

Toinen normaalimuoto (2NF) koskee vain niitä tauluja, joissa on moniosainen perusavain. Sen mukaan sarakkeiden tulee olla funktionaalisesti riippuvaisia koko perus-

avaimesta, eikä sen osasta tai osista. Funktionaalisella riippuvuudella tarkoitetaan yksittäisten tietojen välisiä riippuvaisuuksia. Sen kautta tarkastellaan myös kaikkia seuraavia normaalimuotoja. (Hovi, Huotari & Lahdenmäki, 2005 90-91.) Alla on esitelty toista normaalimuotoa selventävä esimerkki (Kuva 14).

Toimittajatun	Osanro	Pvm	Toim_nimi	Osanimi	Määrä
1202	707b	4.6.2003	Varaosa Oy	Tulppa 12	22
1202	708b	3.4.2003	Varaosa Oy	Tulppa 13	10
1275	707b	15.5.2003	A-osat Oy	Tulppa 12	5
...					

Kuva 14. Toimittajatun, Osanro ja Pvm ovat perusavaimena tässä esimerkkitaulussa. Toim_nimi on riippuvainen Toimittajatun-sarakkeesta, Osanimi puoletaan Osanro-sarakkeesta. Ainoastaan Määrä-sarake on riippuvainen koko perusavaimesta. Taulu ei siis ole toisessa normaalimuodossa. Tilanteen korjaaminen vaatii taulun rikkomista kolmeen tauluun: Toimittajatun/Toim_nimi-, Osanro/Osanimi- sekä Toimittajatun/Osanro/Pvm/Määrä-tauluihin. (Hovi, Huotari & Lahdenmäki, 2005 91-92.)

Kolmas normaalimuoto (3NF) saavutetaan silloin, kun taulun jokainen perusavaimeen sisältymätön sarake on funktionaalisesti riippuvainen vain ja ainoastaan perusavaimesta. Sarakkeella ei siis saa olla riippuvaisuuksia ei-perusavaimena toimivaan sarakkeeseen. Tyypillinen tätä normaalimuotoa rikkova esimerkki on sekä postinumero- että postitoimipaikkasarakkeen sisällyttäminen samaan osoitetietotauluun. Postitoimipaikka on tällöin funktionaalisesti riippuvainen postinumerosta, mikä ei ole perusavain. (Hovi, Huotari & Lahdenmäki 2005, 93-94.) Tämä esimerkki on kuitenkin siinä mielessä hyökkäävä, että esimerkiksi postitoimipaikan määrittämiseen käytetään yleisesti sekä postinumeroa että postitoimipaikan nimeä yhdessä.

Kolmatta normaalimuotoa laajentava ja sitä tiukempi normaalimuoto on Boyce-Codd normaalimuoto (BCNF). Sen mukaan taulun pitää olla kolmannessa normaalimuodossa ja kaikkien funktionaalisten määrääjien täytyy olla taulun avainehdokkaita. (Elmasri & Navathe, 2004 324-325.) BCNF-normaalimuoto laajentaa siis kolmatta normaalimuotoa niin, että kaikkien riippuvuuksien määrääjien täytyy olla taulun avainehdokkaita. Käytännössä suurin osa kolmannessa normaalimuodossa olevista tauluista on myös BCNF-normaalimuodossa.

Vaikka normalisoinnilla saavutetaankin useita etuja aina suorituskyvyn parantumisesta alkaen, joskus siitä on myös haittaakin. Normalisointi rikkoo tauluja useampaan osaan, jolloin kyselyissä täytyy tehdä enemmän taulujen välisiä liitoksia. Tämä saattaa joissain tapauksissa hidastaa merkittävästi kyselyaikaa, jolloin normalisoinnista on tietokannalle enemmän haittaa kuin hyötyä. Normalisoitujen rakenteiden palauttamista ei-normalisoituun muotoon suorituskyvyn parantamiseksi kutsutaan denormalisoinniksi. Tätä saatetaan hyödyntää kyselypainotteisissa tietokannoissa, esimerkiksi tietovarastoissa. (Huovi, Huotari & Lahdenmäki, 2005 95-96.)

4.3 Käyttöliittymän suunnittelu

Käyttöliittymän suunnittelun tärkeyttä vähätellään usein ohjelmistosuunnittelussa. Vaikka alan ammattilaisen pitäisi periaatteessa suunnitella käyttöliittymät, niin niiden suunnittelu jää yleensä ohjelmistosuunnittelijan tehtäväksi. Ohjelmistosuunnittelijalla on usein toki tarvittavat taidot käyttöliittymän toteuttamiseen, mutta heidän ratkaisunsa eivät välttämättä ole esteettisiä tai käyttäjäystävällisiä. Valmis ohjelma ei ikinä saavuta täyttä potentiaaliaan, jos sillä on kehno käyttöliittymä. (Sommerville, 2004 363.)

Käyttöliittymän suunnittelussa kannattaa kiinnittää huomiota seuraaviin asioihin:

- Käyttäjien aikaisemman kokemuksen ja osaamisen huomioiminen. Käyttöliittymässä käytettävien termien, ulkoasun ja toimintojen tulisi peilata käyttäjille jo ennestään tuttuja asioita. Suunnittelijan ei kuitenkaan pidä unohtaa uusien, mahdollisesti kokemattomien käyttäjien näkökulmaa.
- Käyttöliittymän on oltava yhdenmukainen. Käyttöliittymän logiikka ja ulkoasu ei saa vaihdella sovelluksen sisällä.
- Mahdollisimman selvät ja yllätyksettömät tapahtumat. Käyttäjä ei saisi koskaan yllättyä sovelluksen käyttäytymisestä.
- Virheistä ilmoittaminen ja niistä palautuminen. Virheilmoitusten tulee olla mahdollisimman selkokielisiä ja käyttäjälle hyödyllisiä, eikä käyttäjän syöttämä tieto saisi kadota virheiden tapahtuessa.

- Kriittisten toimintojen varmistaminen. Käyttäjältä tulisi aina varmistaa mahdollisesti kriittisten toimintojen suoritus, esimerkiksi tiedon poistaminen.

(Sommerville, 2004 364-365.)

Suunnitteluprosessi etenee tyypillisesti läheisessä yhteistyössä asiakkaan ja loppukäyttäjien kanssa. Prosessin ensimmäinen vaihe on käyttäjäanalyysi, jossa selvitetään muun muassa käyttäjien toimintatapoja, käyttöympäristöä ja muita heidän käyttämiään järjestelmiä. Käytännössä käyttäjäanalyysillä tähdätään käyttäjien aikaisemman kokemuksen ja osaamisen huomioimiseen. Toisessa vaiheessa kehittäjät luovat käyttöliittymäprototyyppejä käyttäjien testattavaksi. Kolmannessa ja viimeisessä vaiheessa käyttäjät antavat palautetta käyttöliittymäprototyypeistä, joiden perusteella kehittäjä muokkaa prototyypistä uuden version. Toista ja kolmatta vaihetta toistetaan, kunnes käyttöliittymä on hyväksyttävällä tasolla. (Sommerville, 2004 376-377.)

5 SOVELLUKSEN TOTEUTUS

5.1 Suunnittelu ja määrittely

Sovelluksen hankintasopimus- ja hintaliitepuolen määrittelyvaiheeseen käytettiin suurimmaksi osaksi aikaisemmin käytössä olleita lomakkeita. Näistä suurimmassa roolissa olivat hankintasopimuslomake (Liite 1) ja hintaliitelomake (Liite 2). Hintaliitelomaketta hyödynnettiin myös sovelluksen laskutuspuolen määrittelyssä.

Ensimmäinen vaatimusmäärittelydokumenttiin verrattavissa oleva dokumentti luotiin vasta hankintasopimuspuolen testiversion jälkeen, jolloin tietokantaa ja sovellusta oltiin suunniteltu jo suhteellisen pitkälle. Muut määrittelyt, esimerkiksi reunaehdot ja rajapintojen määrittely muihin järjestelmiin sovittiin pääsääntöisesti suullisesti. Vaikka dokumentoinnin ja tarkkojen vaatimuksien puutteen voisi tulkita ketteräksi tuotantomenetelmäksi (Haikala & Mikkonen, 2011 43-44), se ei ollut suunnitelmallista. Projekti olisi selkeästi kaivannut tarkempaa vaatimusmäärittelyä ja käyttäjäkeskeistä suunnittelua.

5.1.1 Toiminnalliset vaatimukset

Alkuperäisen vaatimusmäärittelydokumentin mukaan oppisopimusten hankintasopimuksien hallintasovellukseen kohdistuu seuraavia toiminnallisia vaatimuksia:

- Sopimukseen voi kuulua joko yksi tai useampi opiskelija.
- Jos sopimuksessa on useampi opiskelija, niin kaikilla on oltava sama tutkintotavoite ja tutkinnonosat. Hintaliitteiden lähipäivä- ja etätehtävämäärät sekä näistä riippumattomat muut kustannukset kertautuvat siis opiskelijamäärän mukaan.
- Hankintasopimukseen voi kohdistua joko välilaskuja tai loppulaskuja. Välilaskuja tehdään siinä tapauksessa, jos kaikki tavoitteita ei olla vielä saavutettu. Loppulasku on taas puolestaan hankintasopimuksen viimeinen lasku.
- Loppulasku voi olla myös vajaa, eli sen ei tarvitse täyttää hintaliitteen osoittaa hintaa. Tämä voi tapahtua esimerkiksi tapauksissa, joissa opiskelija keskeyttää opintonsa.

(Oppisopimuslaskutussuunnitelma, 2015.)

Vaatimusmäärittelydokumentin esittämät vaatimukset eivät suurimmalta osin peilaa enää sovelluksen nykytilaa. Vaatimukset muuttuivat ja kehittyivät nykyiseen muotoonsa varsinkin projektin keskivaiheilla. Myös uusia vaatimuksia syntyi jonkin verran, näistä mainittakoon hankintasopimuksien monivaiheinen tallentaminen sekä tulostusnäkymät ja prosessia tukevat sähköposti-ilmoitukset.

Merkittävämmät muutokset julkaistussa sovelluksessa koskivat useamman opiskelijan hankintasopimuksia ja siten koko laskutuspuolta. Jokaisella hankintasopimukseen liitetyllä opiskelijalla on nykyään henkilökohtainen hintaliite, mikä sallii erilaiset tutkinnonosakokonaisuudet tutkintotavoitteen pysyessä kuitenkin samana. Opiskelijat saattavat valita toisista hankintasopimuksen opiskelijoista poikkeavia valinnaisia tutkinnonosia, jolloin alkuperäisen suunnitelman mukainen hintaliitekokonaisuuden laskuttaminen ei olisi toiminut. Tämä mahdollistaa myös tarkemman tiedonkoonnin esimerkiksi tarkastuksia varten ja opiskelijakohtaisen etenemisen seuraamisen.

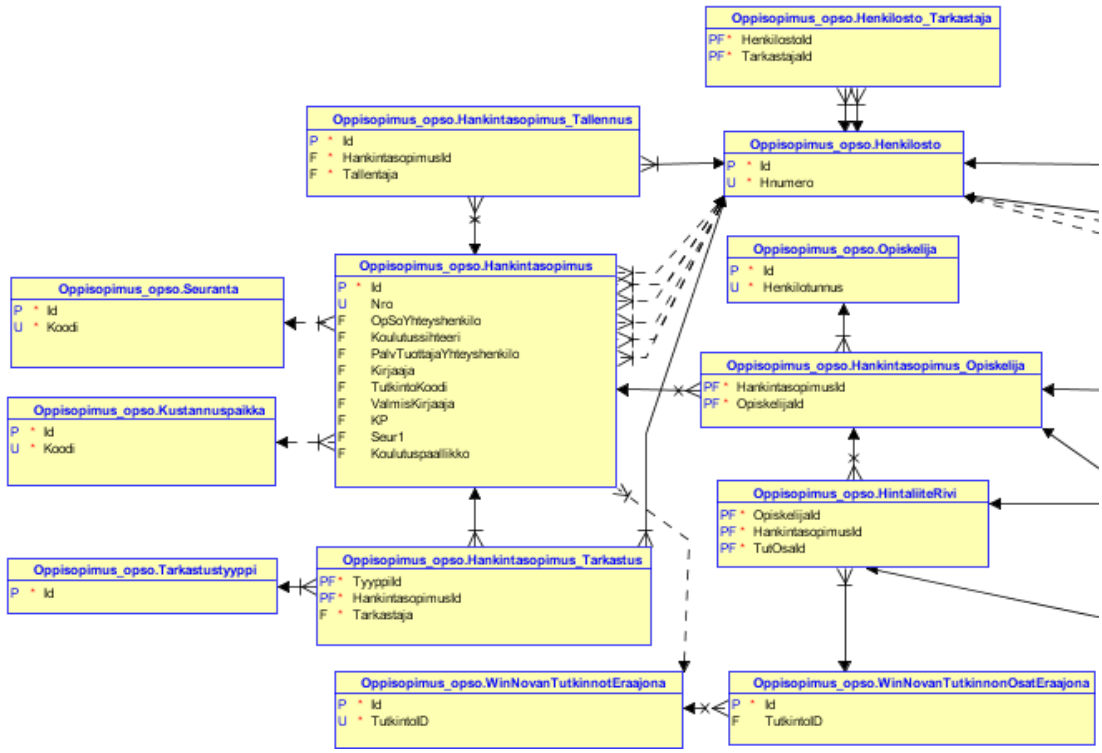
5.1.2 Reunaehdot

Reunaehdoista ei ole virallista dokumenttia, sillä ne sovittiin suullisesti projektin alkuvaiheessa. Sovelluksen tietokantana tuli toimia SQL Server 2008 ja sovelluslogiikka tuli toteuttaa joko Visual Basicillä tai C#:lla. Lisäksi hankintasopimussovelluksessa käytettävät tutkinnot ja niiden tutkinnonosien tuli olla sidottu WinNovassa käytettyyn tutkintorakennekäytäntöön, eli tuotteistamiseen. Tuotteistetut tutkinnot ja tutkinnonosat päivittyvät sovelluksen käyttöön WinNovan toisesta tietokannasta. Lisäksi toivottiin, että StudentaPlussassa olevat suoritusmerkinnät ilmenisivät jollakin tavalla myös laskutuksen puolella. Rajapinta tarkastettujen ja hyväksyttyjen laskujen siirtymiseen toteutettiin WinNovan puolesta. Myös yrityksen AD-tunnusten hyödyntäminen kirjautumisessa oli suotavaa. Lopullinen tekninen ja visuaalinen toteutustapa oli ohjelmistosuunnittelijan itse valittavissa.

5.2 Tietokantataso

Ensimmäinen versio tietokannasta suunniteltiin ja toteutettiin alkuperäisten vaatimusmäärittelyjen perusteella. Tietokanta koki voimakkaita muutoksia alkuperäisestä muodostaan, mikä tuki täysin erilaista suhtautumista opiskelijan, hankintasopimuksen ja laskujen välillä. Sitä yritettiin pitää toimintakykyisenä erikoisvirityksillä ja näkymäpohjaisilla ratkaisuilla. Asiakasyritys olisi kelpuuttanut myös aikaisemman, jonkinlaisessa välitilassa olleen viritelmän tietokannakseen. Tietokanta rakennettiin kuitenkin projektin loppupuolella lähes kokonaan uudelleen ohjelmistosuunnittelijan päätöksestä.

Tietokantamallin esittämisessä (Kuva 15 ja Kuva 16) on hyödynnetty ilmaista Oraclen SQL Developer Data Modeler-työkalua (Oracle, 2016). Työkalulla voidaan esimerkiksi ottaa yhteys jo olemassa olevaan tietokantaan, jonka jälkeen siitä voidaan mallintaa haluttuja näkymiä.



Kuva 15. Kuvankaappaus tietokannan hankintasopimuspuolesta. Tauluissa näkyvät vain perusavain-, avainehdokas- ja viiteavainsarakkeet. P tarkoittaa pääavainta, F viiteavainta ja U määriteltyä avainehdokasta.

Yrityksen henkilöstötietoja ylläpidetään Henkilosto-taulussa, mitä päivitetään päivittäin yrityksen tietovaraston kautta. Taulussa on myös Rooli-sarake, millä määritellään henkilön sovellusrooli. Tällä hetkellä vain yksi rooli per henkilö on ollut riittävä, mutta tilanteen muuttuessa tietokanta kaipaisi vielä omaa taulua henkilöstön roolitukselle. Lisäksi tärkeässä roolissa on Henkilosto_Tarkastaja-taulu, jonka avulla koulutuspäällikkökohtaisia näkymiä ja oikeuksia voidaan liittää toisille henkilöille.

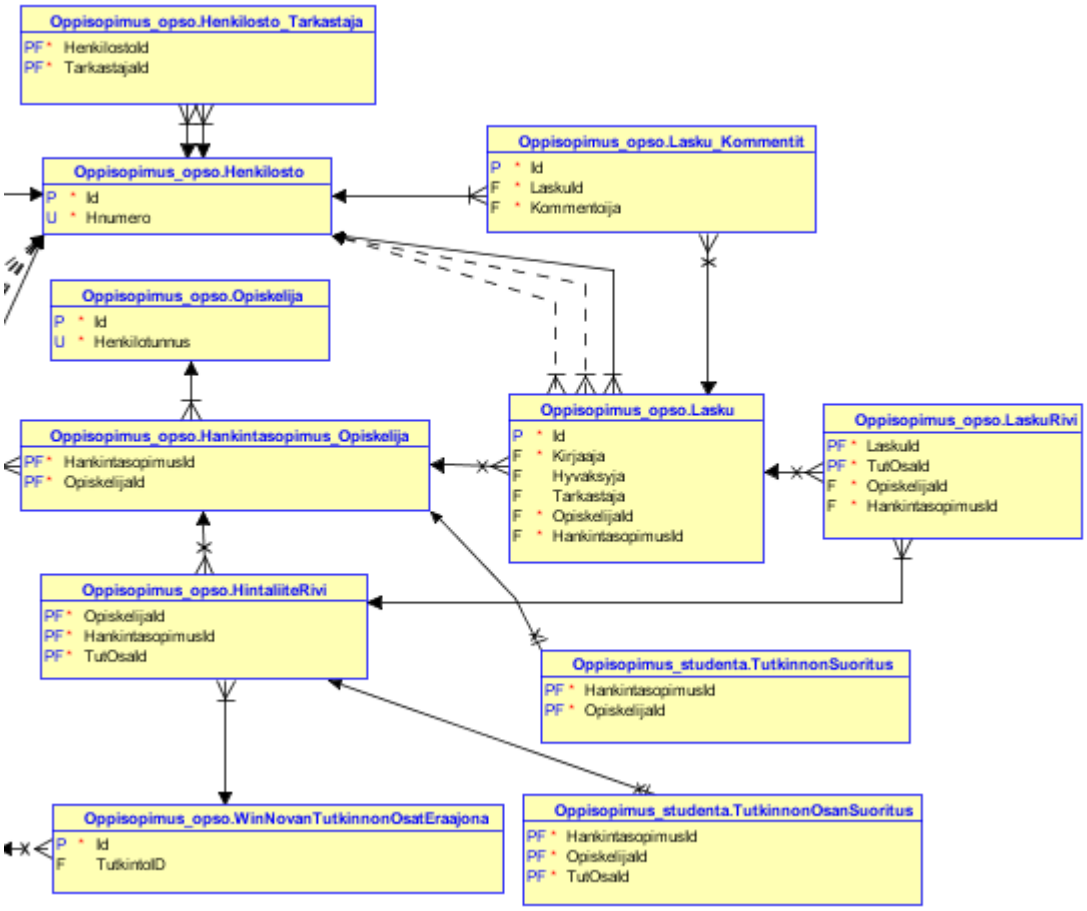
Sovelluksen tietokannan hankintasopimuspuoli on toteutukseltaan laskutuspuolta yksinkertaisempi. Hankintasopimuksen vaiheittainen tallentaminen on toteutettu LuontiKesken-sarakkeella, jonka tietotyyppinä on BIT (0-1 eli false-true). Niin kauan, kuin LuontiKesken-sarakkeen arvo on true, tietokanta antaa käyttäjän jättää vaadittavat sarakkeet myös ilman arvoa. Tämä on mahdollistettu eheyssäännöllä (Hovi, Huotari & Lahdenmäki, 2005, 113). Sen avulla varmistetaan, etteivät vaadittavat sarakkeet ole NULL, jos LuontiKesken-sarakkeen arvo on false. Toteutuksen huonona puolena voidaan pitää sitä, että NULL-sarakkeita on useita.

Hankintasopimuksen tallennusten kirjaamiseen on käytetty Hankintasopimus_Tallennus-taulua. Tauluun tallentuu aina hankintasopimuksen tallennuksen yhteydessä tallennusajankohta ja tallentaja. Lisäksi siihen on liitetty laukaisin (trigger), mikä laukeaa aina tauluun kirjoittamisen jälkeen. Yksinkertaistettuna laukaisimet ovat SQL-skriptiä, mikä on määritelty ajettavaksi tietyn SQL-komennon kohdistuessa tauluun (Hovi, Huotari & Lahdenmäki, 2005, 14). Tätä on hyödynnetty sähköpostien lähettämisessä hankintasopimukseen liitetyille henkilöille. Tallennusten yhteydessä hankintasopimukseen liitetty koulutuspäällikkö ja itse tallentaja saavat viestin hankintasopimuksen luomisesta ja linkin kyseiseen hankintasopimukseen.

Hintaliitteen tarkastamiseen, sekä mahdollisesti myös muihin tuleviin tarkastuksiin, käytetään Hankintasopimus_Tarkastus-taulua. Koulutusasiantuntijoiden suorittama hintaliitteen tarkastaminen ei kuitenkaan ole vielä pakollinen hankintasopimuksen luonnin kannalta.

Hankintasopimuksen ja opiskelijan yhteys on määritelty Hankintasopimus_Opiskelija-taulussa. Valmiissa hankintasopimuksessa jokaisella hankintasopimuksen ja opiskelijan yhdistelmällä on hintaliiterivejä HintaliiteRivi-taulussa. HintaliiteRivi-taulun perusavain on kolmiosainen: OpiskelijaId ja HankintasopimusId viittaavat yhdessä Hankintasopimus_Opiskelija-tauluun, kun taas TutOsaId viittaa WinNovanTutkinnonOsatEraajona-tauluun. Jälkimmäisen taulun nimi saattaa olla harhaanjohtava. Eräajoksi saatetaan mieltää päivitystekniikka, jossa koko taulu tyhjennetään ja täytetään uudelleen. Tutkinnonosia päivitetään kuitenkin huolellisesti proseduurilla, jotta tietokannan eheys olisi taattu.

Hintaliite- ja laskurivien hintasummia ei ole tallennettu tietokantaan. Summat lasketaan sovelluksen proxy-luokissa tai tietokannan omissa näkymissä laskusiirtoja varten. Näin vältetään toisista sarakkeista riippuvaiset sarakkeet ja tiedon eheyden vaarantaminen.



Kuva 16. Kuvankaappaus tietokannan laskutuspuolesta.

Hankintasopimuksen laskut Lasku-taulussa kohdistuvat hankintasopimuksen ja opiskelijan yhdistelmiin. Taulu pitää yllä laskujen tilamuutoksia ja kirjaamistietoja. Jokaisella laskulla on myös laskurivejä LaskuRivi-taulussa, jotka sisältävät tutkinnonosakohtaisesti lähipäivä- ja etätehtävämäärät. Laskuriveissä on myös viittaus opiskelijan hintaliiteriviin, jolla varmistetaan hintaliiterivin olemassaolo tietyllä opiskelijalla, hankintasopimuksella ja tutkinnon osalla. Tämä liitos liittyy myös tutkinnonosakohtaisiin hintoihin: laskurivit eivät sisällä tutkinnonosakohtaisia hintoja, vaan ne noudetaan laskun hintaliiterivistä. Lasku_Kommentit-taulu sisältää puolestaan laskulle annetut käyttäjien kommentit. Laskun hylkäämisen yhteydessä nämä kommentit liitetään lähetettävään sähköpostiin.

Laskujen tilamuutoksien yhteyteen on liitetty sähköpostilaukaisin. Laukaisin lähettää sähköpostia sidoshenkilöille aina laskun tilan muuttuessa. Luotu lasku on alkuvaiheessa tarkastamaton ja hyväksymätön. Tarkastusvaiheessa lasku voidaan joko hyväk-

syä tai hylätä. Hyväksymisen yhteydessä sekä laskun että hankintasopimuksen kirjaajat saavat viestin. Jos tarkastaja hylkää laskun, niin vain laskun kirjaaja saa tästä ilmoituksen. Tarkastusvaiheen jälkeen lasku etenee hyväksymisvaiheeseen. Hyväksymisvaiheessa tapahtuvasta hylkäämisestä tai hyväksymisestä ilmoitetaan vain laskun tarkastajalle.

Laskutuspuoli sisältää myös eheyttä valvovia laukaisimia. Yksinkertaisimmasta päästä on loppulaskua valvova laukaisin, mikä varmistaa sen, että hankintasopimuksen ja opiskelijan yhdistelmään ei kohdistu enää loppulaskun jälkeen enempää laskuja. Tärkein ja monimutkaisin laukaisin on laskurivien luontiin ja päivittämiseen liitetty laukaisin, jonka katkelmaa esitellään alla (Kuva 17). Sillä varmistetaan se, että laskujen hinnat ja määrät eivät koskaan ylitä hintaliitteen asettamia arvoja. Laukaisin peruuttaa laskurivin luonnin, jos se rikkoo tietokannan eheyttä.

```
INSERT INTO @LaskuRivi (TutOsaId, HankintasopimusId, OpiskelijaId, LahipaivienMaara, EtatehtavienMaara,
                        ValmistavaKoulutusYhteensa, TutkintoTilaisuudenHinta)
(
    SELECT T.TutOsaId, T.HankintasopimusId, T.OpiskelijaId, T.LahipaivienMaara, T.EtatehtavienMaara,
           T.ValmistavaKoulutusYhteensa, T.TutkintoTilaisuudenHinta
    FROM
    (
        SELECT A.TutOsaId,
               A.HankintasopimusId,
               A.OpiskelijaId,
               B.LahipaivienMaara - SUM(D.LahipaivienMaara) AS LahipaivienMaara,
               B.EtatehtavienMaara - SUM(D.EtatehtavienMaara) AS EtatehtavienMaara,
               B.ValmistavaKoulutusYhteensa - SUM(D.ValmistavaKoulutusYhteensa) AS ValmistavaKoulutusYhteensa,
               B.TutkintoTilaisuudenHinta - SUM(D.TutkintoTilaisuudenHinta) AS TutkintoTilaisuudenHinta
        FROM INSERTED A JOIN Oppisopimus.opso.LaskuRivi D ON A.TutOsaId = D.TutOsaId
                                                         AND A.HankintasopimusId = D.HankintasopimusId
                                                         AND A.OpiskelijaId = D.OpiskelijaId
                        JOIN Oppisopimus.opso.Lasku C ON D.LaskuId = C.Id
                        JOIN Oppisopimus.opso.HintaliiteRivi B ON A.HankintasopimusId = B.HankintasopimusId
                                                              AND A.OpiskelijaId = B.OpiskelijaId
                                                              AND A.TutOsaId = B.TutOsaId

        WHERE C.Hyvaksytty IS NULL OR C.Hyvaksytty = 1
        GROUP BY A.TutOsaId, A.HankintasopimusId, A.OpiskelijaId, B.LahipaivienMaara, B.EtatehtavienMaara,
                 B.ValmistavaKoulutusYhteensa, B.TutkintoTilaisuudenHinta
    ) AS T
    WHERE T.LahipaivienMaara < 0
          OR T.EtatehtavienMaara < 0
          OR T.ValmistavaKoulutusYhteensa < 0
          OR T.TutkintoTilaisuudenHinta < 0
)
```

Kuva 17. Kuvankaappaus laskurivien summia vahtivasta laukaisimesta. Laukaisin huomioi myös aikaisempien odottavien tai hyväksyttyjen laskujen summat. Toteutuksessa kuvan kysely kirjoitetaan muuttujatauluun: jos rivejä on yksi tai enemmän, niin hintaliitteen summat ylittyvät, laskurivin luonti peruutetaan ja käyttäjälle ilmoitetaan virheen aiheuttaneet tutkinnonosat.

StudentaPlussan rajapinta sijaitsee myös käytännössä laskutuspuolen tauluissa. Opiskelijalla ei periaatteessa voi olla tutkinnonosien suorituksia ennen, kuin hankintasopimus on siirtynyt laskutusvaiheeseen. Rajapinnan taulut ovat omassa studenta-schemassaan ja niiden tietoja päivitetään StudentaPlussan raportointitietokannan kautta. Raportointitietokanta on Solenovon oma tietovarastotyyppinen tietokanta, joka ajetaan päivittäin WinNovan omalle tietokantapalvelimelle.

5.3 Tiedonsaantitaso

5.3.1 Luokat

Kaikki sovelluksen luokat ovat pääsääntöisesti tietokannan tauluista johdettuja. Näihin luokkiin on myös sisällytetty validointilogiikkaa DataAnnotations-nimiavaruuden kautta tarjottavilla attribuuteilla. Attribuuttien avulla määriteltyjä sääntöjä noudatetaan myöhemmin automaattisesti ASP.NET MVC:n sovellus- ja käyttöliittymäpuolella. Lisäksi luokkiin voidaan määrittää ominaisuuksien näkymänimiä (DisplayName), eli millä nimellä ominaisuus esitetään käyttöliittymässä. (Microsoft, 2016i.) Alla olevassa kuvassa esitellään Opiskelija-luokassa olevien DataAnnotations-attribuuttien käyttöä (Kuva 18).

```
50 references | Korpimursu, 128 days ago | 2 changes
public class Opiskelija
{
    [Required(ErrorMessage="Anna opiskelijan etunimet")]
    25 references | Korpimursu, 133 days ago | 1 change
    public string Etunimet { get; set; }
    9 references | Korpimursu, 133 days ago | 1 change
    public int? Henkilonro { get; set; }
    [DisplayName("Henkilötunnus")]
    [Required(ErrorMessage="Anna opiskelijan henkilötunnus")]
    [RegularExpression(@"^\d{6}[+-A]\d{3}[0-9A-Y]$", ErrorMessage = "Anna henkilötunnus oikeassa muodossa")]
    19 references | Korpimursu, 133 days ago | 1 change
    public string Henkilotunnus { get; set; }
```

Kuva 18. Esimerkki validointi- ja näkyvyystekstiattribuuttien käytöstä. ErrorMessage-ominaisuuden avulla voidaan määrittää näytettävä virheteksti. RegularExpression-attribuutilla pystytään määrittelemään monimutkaisempia regular expressioniin tukeutuvia validointilauseita.

Luokat sisältävät myös virtual-ominaisuuksia proxy-suunnittelumallin toteutusta varten. Ominaisuudet ovat täysin sidoksissa tietokannan taulujen välisiin suhteisiin. Esimerkiksi hankintasopimuksella on Kustannuspaikka-tyyppinen ominaisuus (KP-sarakkeen avulla haettava Kustannuspaikka-taulun olio) sekä List<Lasku>-tyyppinen ominaisuus (Id-sarakkeen avulla haetaan kaikki Lasku-taulun oliot). Näiden ominaisuuksien tuottamiseksi tarvitaan repository-olioita, jotka sisällytetään proxy-luokkiin.

Tietokannasta johdettujen luokkien lisäksi sovelluksen luokkakirjasto sisältää myös niin sanottuja apuluokkia (Kuva 19). Niiden ainoana tarkoituksena on laajentaa jo olemassa olevia luokkia käyttöliittymätasoa varten. Luokkia käytetään esimerkiksi opiskelijan ja hintaliiterivien yhdistämisessä hankintasopimuksen luonnin yhteydessä. Arkkitehtuurillisesti apuluokat voisivat sijaita myös MVC-projektin mallikokoelmassa, koska niiden ainoa tarkoitus on helpottaa näkymien ja ohjaimien välistä kommunikaatiota.

```
7 references | Korpimursu, 133 days ago | 1 change
public class HOpiskelija: Opiskelija
{
    11 references | Korpimursu, 133 days ago | 1 change
    public List<HintaliiteRivi> HRivi { get; set; }
    6 references | Korpimursu, 133 days ago | 1 change
    public List<WinNovanTutkinnonOsatEraajona> TutOsat { get; set; }
}
```

Kuva 19. Yksinkertainen HOpiskelija-apuluokka, joka periytyy Opiskelija-luokasta. Apuluokkaa käytetään hankintasopimuksen luontivaiheessa. HRivi sisältää kyseisen opiskelijan hintaliiterivit, kun taas TutOsat sisältää kaikki opiskelijalle valittavissa olevat tutkinnonosat.

5.3.2 Repository- ja Proxy-luokat

Jokaiselle tietokannan taulusta johdetulle luokalle on olemassa oma repository- ja proxy-luokkansa. Repositoryn hakumetodit palauttavat palvelevan luokkansa mukaisia olioita, jotka on luotu proxy-luokkansa kautta (Kuva 20). Kaikki repository-luokat periytyvät abstraktista DataAccess-luokasta, jonka ConnectionString-ominaisuuden kautta proxy-luokkiin upotettaville repository-luokille voidaan antaa helposti sama yhteysmerkkijono.

```
private HintaliiteRivi TeeRivistaHintaliiteRivi(IDataReader reader)
{
    HintaliiteRiviProxy paluu = new HintaliiteRiviProxy(
        int.Parse(reader["HankintasopimusId"].ToString()),
        int.Parse(reader["OpiskelijaId"].ToString()),
        int.Parse(reader["TutOsaId"].ToString()),
        reader["EtatehtavanHinta"] is DBNull ? (decimal?)null : decimal.Pars
        reader["EtatehtavienMaara"] is DBNull ? (decimal?)null : decimal.Par
        reader["Huom"] is DBNull ? null: reader["Huom"].ToString(),
        reader["LahipaivanHinta"] is DBNull ? (decimal?)null : decimal.Parse
        reader["LahipaivienMaara"] is DBNull ? (decimal?)null : decimal.Pars
        reader["TutkintoTilaisuudenHinta"] is DBNull ? (decimal?)null : deci
        reader["ValmistavaKoulutusYhteensa"] is DBNull ? (decimal?)null : de
        );
    paluu.WinNovanTutkinnonOsatEraajonaRepository = new WinNovanTutkinnonOsa
    paluu.OpiskelijaRepository = new OpiskelijaRepository(ConnectionString);
    paluu.HankintasopimusRepository = new HankintasopimusRepository(Connecti
    paluu.LaskuRiviRepository = new LaskuRiviRepository(ConnectionString);
    paluu.TutkinnonOsanSuoritusRepository = new TutkinnonOsanSuoritusReposit
    return paluu;
}
```

Kuva 20. HintaliiteRiviRepository-luokassa oleva metodi, mikä olioistaa tietokannasta haettavat HintaliiteRivi-taulun rivit. Proxy-luokka kapsuloidaan isäluokkansa sisälle. Lopputuloksena palautuvan HintaliiteRivi-olion virtual-ominaisuudet palvelevat proxy-luokan määrittelemää toteutusta.

Repository-ominaisuuksien lisäksi proxy-luokat sisältävät yksityisinä kenttinä kaikki isäluokkansa vaatimat yliajettavat ominaisuudet sekä oman boolean-kentän jokaiselle näistä. Proxy tallentaa haetut tiedot kenttiin myöhempiä kyselyjä varten ja boolean-kentällä tarkistetaan, onko tieto jo haettu. Proxy-luokan ei ole kuitenkaan pakko laajentaa vain olioita palauttavia ominaisuuksia, vaan niihin voidaan myös sisällyttää esimerkiksi muiden olioiden ominaisuuksien perusteella laskettavia arvoja, kuten alla olevassa kuvassa havainnollistetaan (Kuva 21).

```
public override decimal? LahipaivienMaaraJaljella
{
    get
    {
        if (_lahipaivienmaarajaljellafetched)
        {
            return _lahipaivienmaarajaljella;
        }
        else
        {
            _lahipaivienmaarajaljella = HintaliiteRivi.LahipaivienMaara - Hintaliit
            _lahipaivienmaarajaljellafetched = true;
            return _lahipaivienmaarajaljella;
        }
    }
```

```
public override HintaliiteRivi HintaliiteRivi
{
    get
    {
        if (_hintaliiterivifetched)
        {
            return _hintaliiterivi;
        }
        else
        {
            _hintaliiterivi = HintaliiteRiviRepository.ReadLaskuRivi(base.Hankintas
            _hintaliiterivifetched = true;
            return _hintaliiterivi;
        }
```

Kuva 21. Kuvankaappaus LaskuRiviProxy-luokasta. Yliajettava ominaisuus laskee kyseisen laskurivin maksamatta olevat lähipäivien määrät HintaliiteRivi-ominaisuuden avulla. Jokaiseen laskuriviin liittyy aina yksi hintaliiterivi HankintasopimusId-, OpiskelijaId- ja TutOsaId-sarakkeiden kautta.

Jokainen repository-luokan kantaan yhteyden ottava metodi sisältää try-catch-finally -lohkon, SqlConnection- ja SqlCommand-olion (kantayhteys ja SQL-komennon suorittaja) sekä ajettavan SQL-komentotekstin. SqlCommand-olioon tallennetaan myös mahdolliset parametrit SqlParameter-olioina Parameters-ominaisuuteen. Tämä on tärkeää mahdollisten SQL-injektioiden torjumiseksi: parametreja ei saa missään nimessä katenoida suoraan SQL-komentotekstiin.

Tietokannan tietoja muokkaavat metodit sisältävät SqlTransaction-olion, jolla kantaan tehtävä toiminto voidaan virheen sattuessa peruuttaa (rollback) tai hyväksyä (commit). Näin tauluihin ei kirjoiteta virheellistä tai vajavaista tietoa virheen sattuessa. Esimerkiksi uutta hankintasopimusta tallennettaessa tietoa tallennetaan useampaan tauluun. Jos tietokantaan kirjoittaminen keskeytyy esimerkiksi virheeseen, niin kaikki tehdyt

muutokset on pystyttävä peruuttamaan. Alla olevassa kuvassa (Kuva 22) esitellään sovelluksen yhtä repository-metodia, jossa toteutetaan edellä mainittuja toimintaperiaatteita.

```
public int LisaaUusiLaskuOsineen(Lasku lasku, List<LaskuRivi> laskuosat)
{
    int laskuid = -1;
    string sql = "";
    SqlConnection con = null;
    SqlTransaction tran = null;
    SqlCommand cmd = null;
    try
    {
        con = new SqlConnection(ConnectionString);
        con.Open();
        tran = con.BeginTransaction(IsolationLevel.ReadCommitted);

        //Ensiksi lisätään uusi lasku
        sql = "INSERT INTO Oppisopimus.opso.Lasku (LuontiPvm, Kirjaaja, Komme
        cmd = new SqlCommand(sql, con, tran);
        cmd.Parameters.Add(new SqlParameter("@LuontiPvm", lasku.LuontiPvm.Has
        cmd.Parameters.Add(new SqlParameter("@Kirjaaja", lasku.Kirjaaja));
```

Kuva 22. Osa LaskuRepositoryn uuden laskun riveineen luovasta metodista. Jos sekä lasku että sen rivit lisätään onnistuneesti kantaan, kutsutaan SqlTransaction-olion Commit()-metodia. Lisäyksen epäonnistuessa siirrytään catch-lohkoon, jossa kutsutaankin samaisen olion Rollback()-metodia. Lopputuloksesta huolimatta finally-lohkossa tietokantayhteys suljetaan kutsumalla SqlConnection-olion metodia Close().

5.4 Käyttäjähallinta

5.4.1 AD-integraatio

Sovellus käyttää kirjautumiseen Windows-autentikaatiota. Näin käyttäjä voi kirjautua yrityksen omilla AD-tunnuksillaan sovellukseen. Lisäksi suurin osa selaimista osaa huomioida tällä hetkellä Windowsiin kirjautuneen käyttäjän ja kirjautua sovellukseen ilman erillistä kirjautumisikkunaa. Toinen toteutustapa olisi Forms-autentikaatio, jossa tietokantaan sisällytetään käyttäjätunnukset ja salasanat ja sovellukseen määritellään annettavien tunnuksien validointi. Sovelluksessa ei kuitenkaan ole pakko olla autentikaatiota. Esimerkiksi rajapintasovelluksissa vain tietyt ohjaimen toiminnot voivat vaatia jonkinasteista tunnistautumista.

Henkilosto-taulussa oleva Hnumero-sarake vastaa jokaisen käyttäjän AD-tunnuksessa ominaisuutta employeeID. Hnumero-sarake on kuusinumeroinen luku, mikä on jokaiselle henkilökunnan jäsenellä yksilöllinen. Sarakkeen avulla voidaan hakea sovelluksen tietokannasta kirjautuneen käyttäjän tiedot esimerkiksi lokitusta ja roolitusta varten (Kuva 23).

```
public class AdData
{
    private string _yhteys = "[redacted]";
    16 references | Korpimursu, 131 days ago | 1 change
    public string HaeHenkilonumero(string strtunnus)
    {
        return GetUserInfo(strtunnus, "employeeID");
    }
    1 reference | Korpimursu, 131 days ago | 1 change
    public string GetUserInfo(string inSAM, string inType)
    {
        try
        {
            string sPath = _yhteys;
            string SamAccount = inSAM.Substring(inSAM.IndexOf("\\") + 1);
            DirectoryEntry myDirectory = new DirectoryEntry(sPath, "[redacted]", "[redacted]");
            DirectorySearcher mySearcher = null;
```

Kuva 23. AdData-luokka, jolla haetaan Active Directorystä kirjautuneen käyttäjän käyttäjätunnuksen perusteella hänen employeeID-kenttänsä arvo. Saadulla arvolla saadaan Henkilosto-taulusta vastaava rivi.

5.4.2 Roolitus

Sovelluksessa on seuraavia rooleja:

- Admin, jolla on kaikki oikeudet. Ohjelmistosuunnittelija on sovelluksen kehitys- ja käyttöönottovaiheessa tässä roolissa.
- Hyväksyjä, joka voi lisätä valmiita tai keskeneräisiä hankintasopimuksia tietokantaan ja hyväksyä tarkastettuja laskuja. Lisäksi heillä on täydet käyttöoikeudet ylläpitopuolelle. Tällä hetkellä tässä roolissa on kolme oppisopimuskeskuksen koulutuspalvelusihteeriä, jotka ovat hallinneet myös vanhassa prosessissa hankintasopimuksia ja niiden laskuja.
- Ostaja, joka tarkastaa hintaliitteen hinnat. Voi tarvittaessa tallentaa keskeneräisiä hankintasopimuksia, mutta ei omaa tehtäviä laskutuksen puolella. Kaikki koulutusasiantuntijat ovat tässä roolissa.

- Tarkastaja, joka voi tallentaa hankintasopimuksia keskeneräisenä sekä tarkastaa luotuja laskuja. Tämän roolin omaavat henkilöt ovat pääsääntöisesti koulutuspäälliköitä, mutta myös heidän tehtäviään sovelluksessa suorittavat henkilöt laitetaan tähän rooliin.
- Laskuttaja, joka voi luoda laskuja. Ei omaa mitään tehtäviä hankintasopimuksen luontivaiheessa. Toimii sovelluksen vakioroolina, johon kaikki edellä mainittuihin rooleihin kuulumattomat henkilöt kuuluvat.

Kirjautuneen käyttäjän ja roolin yhdistäminen suoritetaan Global.asax-tiedoston Application_AcquireRequestState-metodissa (Kuva 24). Suoritusjärjestyksessä tapahtuma sijoittuu käyttäjän pyynnön autorisaation ja palvelimen vastauksen väliin (Microsoft, 2016j). Roolit luetaan joko tallennetusta Session-muuttujasta tai tarvittaessa tietokannasta. Session-muuttujat tallentuvat palvelimelle avain-arvo -pareina. Käyttäjän selain tallentaa SessionId-cookien, jolla palvelimelle tallennetut Session-muuttujat yhdistetään oikeaan käyttäjään. (Microsoft, 2016k.) Windows-autentikaatiota käytettäessä itse määritellyt roolit katoavat käyttäjältä aina kutsujen välissä. Tämän torjumiseksi ne pitää asettaa käyttäjälle aina kutsujen välissä uudelleen. Session-muuttujia hyödyntämällä kevennetään huomattavasti turhien Active Directory- ja tietokantahakujen määrää.

```
protected void Application_AcquireRequestState(object sender, EventArgs args)
{
    if (HttpContext.Current != null && HttpContext.Current.Session != null)
    {
        if (HttpContext.Current.Session["Roolit"] != null)
        {
            GenericPrincipal principal = new GenericPrincipal(HttpContext.Current.User.Identity, (string[])Session["Roolit"]);
            Thread.CurrentPrincipal = HttpContext.Current.User = principal;
        }
        else
        {
            AdData Ad = new AdData();
            HenkilostoRepository rep = new HenkilostoRepository(ConfigurationManager.ConnectionStrings["Oppisopimus"].ConnectionString);
            Henkilosto h = rep.ReadHenkilonro(int.Parse(Ad.HaeHenkilonumero(HttpContext.Current.User.Identity.Name)));
            string[] roles = new string[1] { h.Rooli };
            HttpContext.Current.Session.Add("Roolit", roles);
            HttpContext.Current.Session.Add("HenkiloId", h.Id);

            GenericPrincipal principal = new GenericPrincipal(HttpContext.Current.User.Identity, roles);
            Thread.CurrentPrincipal = HttpContext.Current.User = principal;
            Ad = null;
        }
    }
}
```

Kuva 24. Kirjautuneen käyttäjän ja hänen roolinsa yhdistäminen toisiinsa. GenericPrincipal-olio ottaa konstruktorissaan vastaan IIdentity-rajapinnan toteuttavan olion eli kirjautuneen käyttäjän. Toisena parametrina konstruktori vastaanottaa string-jonon haluttuja rooleja eli tietokannasta haetun Henkilosto-olion Rooli-ominaisuuden arvon.

Ohjaimien toimintojen rajoittaminen vain tietyille käyttäjille sekä näkymien ulkoasun yksilöiminen käyttäjäkohtaiseksi tapahtuu sovelluksessa täysin roolien avulla. Ohjaimissa toimintojen rajoittaminen toteutetaan Authorize-attribuutin avulla. Sen lisäksi, että attribuutti edellyttää kirjautunutta käyttäjää, sen Roles-ominaisuuteen voidaan määritellä pilkulla erottelemalla kaikki sallittavat roolit. Attribuutin voi laittaa joko yksittäisen toiminnon yhteyteen tai ohjainluokan alkuun, jolloin kaikki ohjaimen toiminnot perivät määritellyn rajoituksen. (Microsoft, 2016l.) Toimintojen ja näkymien hallinnointia roolien kautta havainnollistetaan alla (Kuva 25).

```
// POST: UusiHankintasopimus/Jatka
[HttpPost]
[Authorize(Roles="Admin,Hyvaksyja")]
0 references | Korpimursu, 9 days ago | 4 changes
public ActionResult Jatka(UusiHankintasopimusViewModel model)...
```

```
<div class="form-horizontal">
    @if (Model.NaytaHankintasopimukset)
    {
        <div class="form-group">
            <h3>Hankintasopimukset</h3>
        </div>
        <div class="form-group" id="id_hankkarit">
            @if (User.IsInRole("Hyvaksyja") || User.IsInRole("Admin"))
            {
                <div class="col-md-6">...</div>
                <div class="col-md-6">...</div>
            }
            @if (User.IsInRole("Ostaja"))
            {
                <div class="col-md-6">
                    <div class="panel panel-default">
```

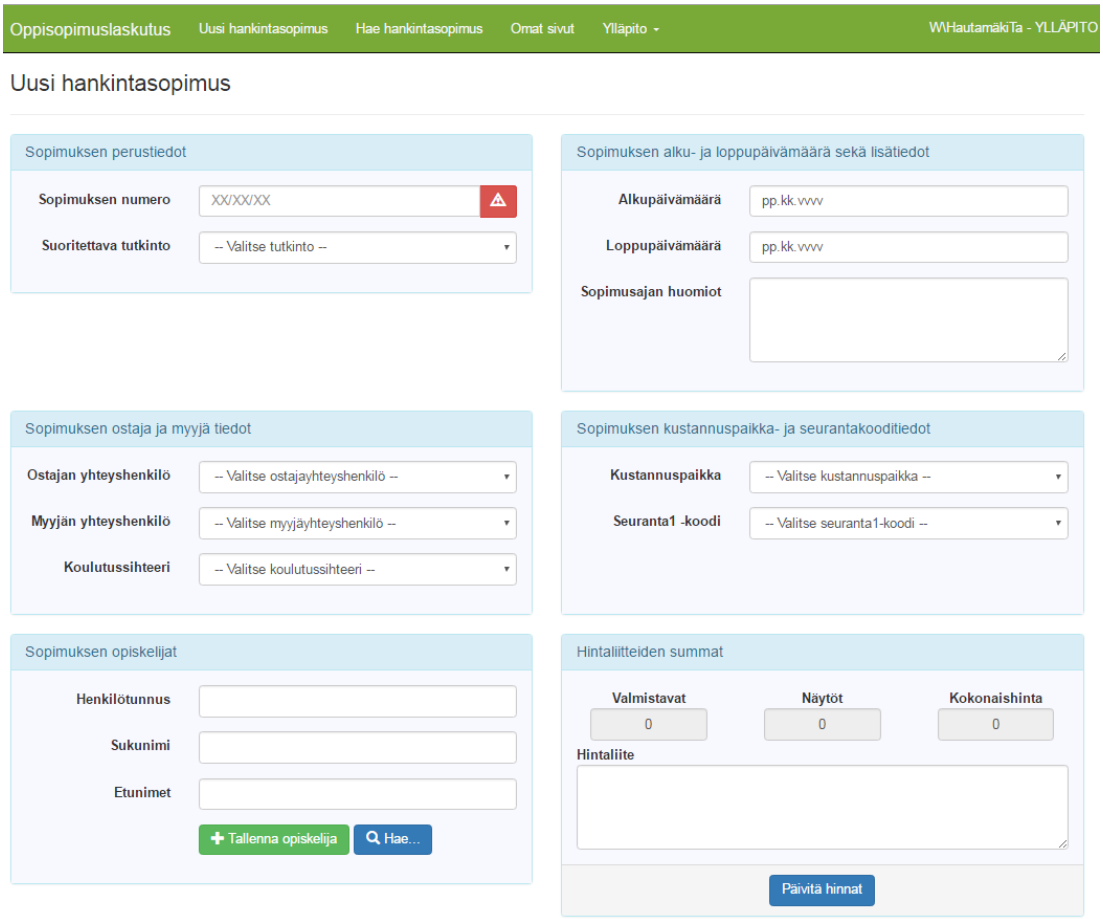
Kuva 25. Authorize-attribuutin käyttöä hankintasopimuksen lopullisesti lisäävässä toiminnossa. Näkymissä käytetään puolestaan IsInRole-metodia esimerkiksi erilaisten ulkoasujen toteuttamiseen tai painikkeiden piilottamiseen.

5.5 Käyttöliittymä

5.5.1 Uusi hankintasopimus

Uuden hankintasopimuksien tallentamisen ja luomisen käyttöliittymä on ensisilmäykseltä varsin perinteinen web-lomake (Kuva 26). Suurin osa lomakkeen kentistä on joko

teksti- tai päivämääräkenttiä tai alasvetovalikoita. Sivun toteutuksessa oli kuitenkin haasteita erityisesti opiskelijoiden ja heidän hintaliitteidensä osalta.



Kuva 26. Uuden hankintasopimuksen luomisnäkymä. Lomake on jaettu kuuteen paneeliin, jotka siirtyvät näyttökoon pienentyessä allekkain.

”Sopimuksen numero”-kentän oikealla puolella oleva merkki kertoo käyttäjälle, onko hankintasopimuksen numero jo käytössä. Kenttään on sidottu keyup-tapahtuma, joka tarkistaa aina ohjaimen kautta numeron yksilöllisyyden ja sijoittaa tuloksen piilotettuun input-elementtiin. Hankintasopimuksen numero-kenttään on tehty oma validaatiometodi, joka tarkastelee edellä mainitun piilotetun kentän arvoa määrittääkseen numeron oikeellisuuden. Punainen merkki tarkoittaa, että numero on joko käytössä tai se puuttuu. Vihreä merkintä puolestaan kertoo käyttäjälle, että numero on vielä käyttämätön. Hankintasopimuksen numeron yksilöllisyyttä vahditaan myös tietokannan puolella, koska se on määritelty taulun avainehdokkaaksi. Se on myös ainoa pakollinen kenttä hankintasopimusta tallennettaessa.

”Hintaliitteiden summat”-paneeli kertoo käyttäjälleen hankintasopimuksen hintaliitteiden hintasummat valmistava- ja näyttöhintoina sekä näiden kahden kokonaissummana. Hinnat lasketaan summaamalla hintaliitteiden valmistavat hinnat (määrät kertaa hinnat) sekä näyttöhinnat (tutkintotilaisuuden hinnat). Tallennettua hankintasopimusta avattaessa nämä summat lasketaan välittömästi käyttäjän nähtäväksi, mikä helpottaa osaltaan hintaliitteen tarkastamista.

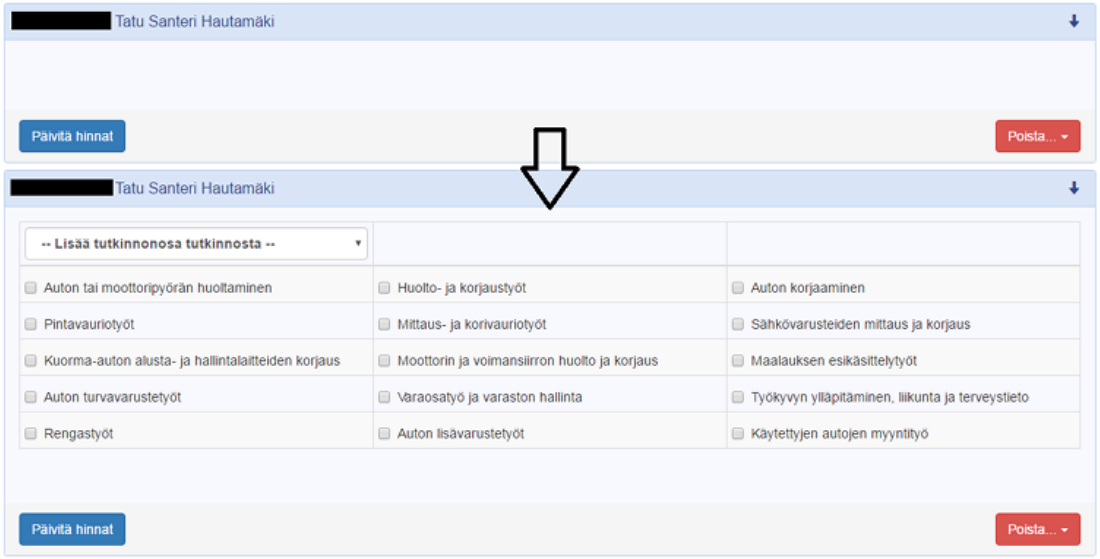
”Sopimuksen opiskelijat”-paneeli on käytännössä oma lomakkeensa. Se ei kuitenkaan lähetä missään vaiheessa tietoa palvelimelle, vaan tallennetut opiskelijat lisätään sivun rakenteeseen. Kentät sisältävät itsessään validointia, mikä toteutetaan vaihtamalla Unobtrusive Ajaxin data-val -attribuutin arvoksi true aina, kun ”Tallenna opiskelija”-painiketta on painettu. Tämän jälkeen jokaiselle elementille kutsutaan erikseen val()- ja valid()-metodeita, joista ensimmäinen ajaa kaikki kentälle määritellyt validoinnit ja jälkimmäinen palauttaa validointien tuloksen. ”Hae…”-painike puolestaan hakee 25 ensimmäistä kenttien arvoihin sopivaa henkilöä WinNovan tietovarastosta ja näyttää heidät käyttäjälle modaali-ikkunassa (Bootstrap, 2016b). Kenttiä voi siis käyttää myös hakuehtoina. Myös avautuvasta ikkunasta voi lisätä opiskelijoita hankintasopimukseen (Kuva 27). Kummassakin tapauksessa tarkastetaan, ettei opiskelija ole jo liitettynä hankintasopimuksessa.

Kuva 27. Avautuva modaali-ikkuna haettaessa sukunimellä ”Hautamäki”. Vihreää painiketta painamalla henkilö lisätään hankintasopimukseen.

Opiskelijan lisäämisen jälkeen hänelle luodaan oma paneeli sivun alalaitaan, joka sisältää opiskelijan tiedot piilotettuina input-kenttinä. Muuten paneelia käytetään pelkkänä pohjana itse hintaliitteelle. Se voidaan laajentaa tai pienentää takaisin pelkälle

otsikkotasolle sen otsikkopalkkia painamalla. Tämä on toteutettu Bootstrapin data-toggle ja data-target -attribuuteilla (Bootstrap, 2016b).

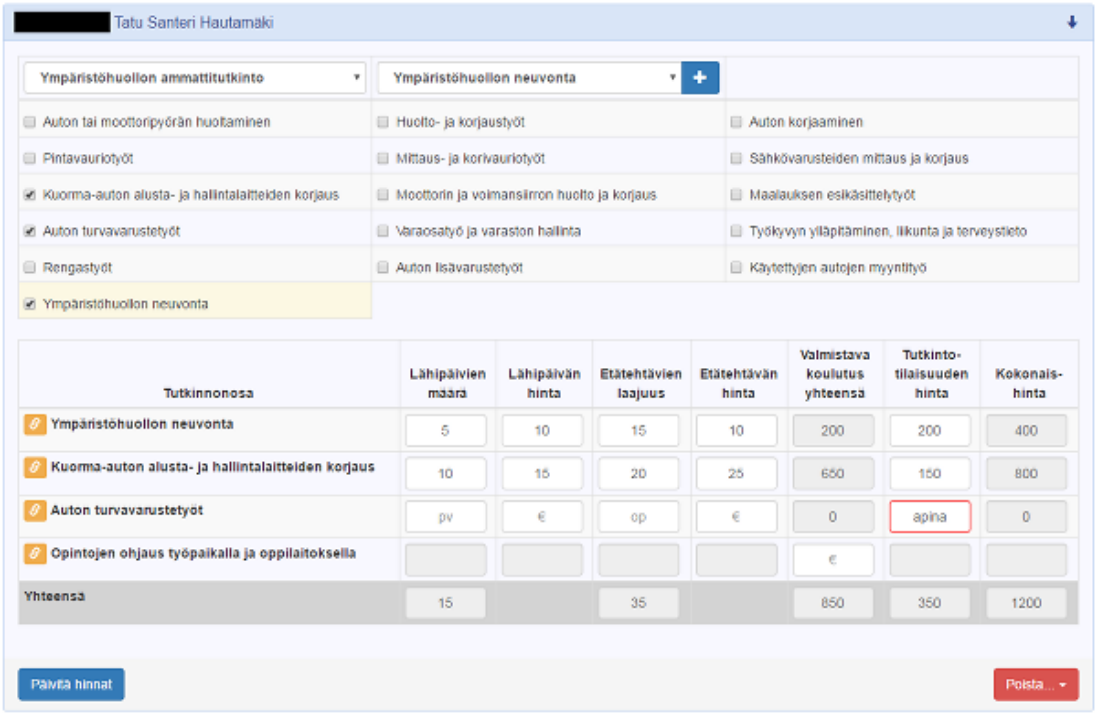
Jos hankintasopimukselle on valittu tutkinto lomakkeen alkupuolelta, niin lisättävälle opiskelijan hintaliitteelle haetaan kyseisen tutkinnon kaikki tutkinnonosat ohjaimelta JSON-muodossa. Jos tutkinto valitaan vasta jälkikäteen, niin kaikkien jo liitettyjen opiskelijoiden hintaliitteiden tutkinnonosavalikoima päivitetään vastaamaan valittua tutkintoa. Alla oleva kuva (Kuva 28) havainnollistaa tarkemmin tutkinnonosavalikoiman muodostumista hintaliitteelle.



Kuva 28. Liitetyn opiskelijan hintaliitteen tutkinnonosavalikoima. Ylempi kuva kuvaa tilannetta, jossa tutkintoa ei olla vielä valittu. Alemmassa kuvassa taas tutkinto on valittuna, jolloin hintaliitteen tutkinnonosavalikoima täytetään. Valikoimaan on mahdollista myös lisätä tutkinnon ulkopuolisia tutkinnonosia ”Lisää tutkinnonosa tutkinnosta”-alasvetovalikon avulla.

Paneeli sisältää myös ”Päivitä hinnat”- ja ”Poista…”-painikkeet. Edellä mainittu laskee hintaliitteen hinnat, kun taas jälkimmäinen on tyylitelty alasvetovalikko, josta voi valita poistetaanko opiskelija ja hintaliite vai vain hintaliitteen rivit. Molemmissa tapauksissa poistotapahtuma varmistetaan käyttäjältä modaali-ikkunalla.

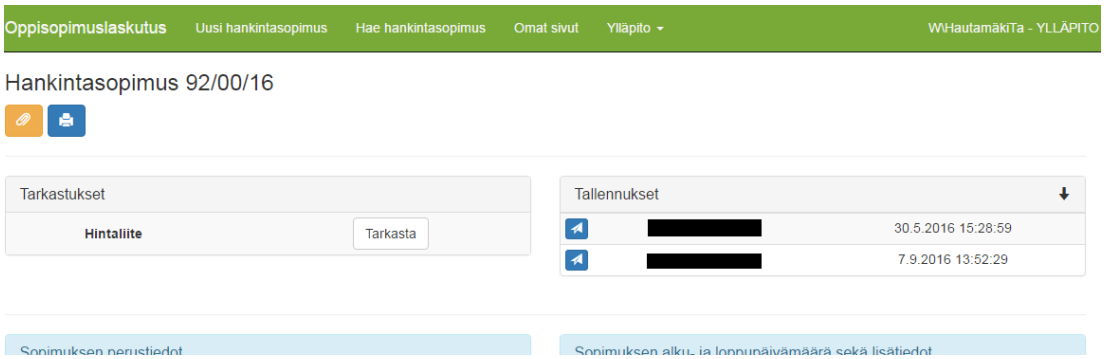
Hintaliitteelle lisätään rivejä valitsemalla suoritettavat tutkinnonosat valikoimasta. Kaikille valikoimataulun sisältämille input-elementeille on määritelty onchange-tapahtuma, joka lisää valitun tutkinnonosan hintaliiteriveihin. Ensimmäisenä lisättävä tutkinnonosa tuo myös mukanaan ”Opiskelujen ohjaus työpaikalla ja oppilaitoksella”-rivin, johon voi asettaa vain valmistavan koulutuksen hinnan kokonaissummana. Jokainen luotu hintaliiterivi sisältää validoinnin, jolla varmistetaan, että syötetyt arvot ovat numeerisia. Rivin tutkinnonosan nimen edessä on myös keltainen painike. Painiketta painamalla kyseinen tutkinnonosa voidaan kopioida hintoineen kaikille muillekin hintaliitteille, jolloin lisättävän rivin tutkinnonosa myös tarvittaessa kopioidaan hintaliitteen valikoimaan. Tällaiset massaominaisuudet ovat miellyttäviä käyttäjälle erityisesti tilanteissa, joissa hankintasopimuksessa on useita opiskelijoita samoilla hintaliiterakenteilla. Edellä mainittuja ominaisuuksia esitellään alla olevassa kuvassa (Kuva 29).



Kuva 29. Opiskelijan hintaliite, johon on valittu tutkinnonosia. Tutkinnonosavalikoimaan on myös lisätty tutkinnon ulkopuolinen tutkinnonosa, mikä näkyy valikoimassa keltaisella taustalla. Hintaliite laskee hinnat ”Päivitä hinnat”-painiketta painamalla. Virheelliset kentät merkitään punaisella reunuksella.

Hankintasopimus voidaan tallentaa joko keskeneräisenä tai valmiina. Kummallekin toiminnolle on oma painikkeensa, joiden näkyvyyttä rajoitetaan roolien avulla. Keskeneräisenä tallennettaessa suurin osa validoinneista otetaan pois käytöstä. Tämä on toteutettu vaihtamalla data-val -attribuutin arvoksi false kaikissa lomakkeen kentissä, pois lukien hankintasopimuksen numero- ja hintaliiterivien kenttiä. Jos ja kun hankintasopimus lisätään lopullisesti järjestelmään, niin painiketta painaessa avautuu vielä yksi modaali-ikkuna. Ikkunassa kirjataan hankintasopimuksen siirtopäivämäärä Sentraali-järjestelmään, joka tehdään toistaiseksi vielä manuaalisesti. Modaali-ikkuna sisältää myös kaikista validointivirheistä yhteenvedon, sillä käyttäjä ei välttämättä näe virheilmoituksia ikkunan ollessa lomakkeen tiellä. Vaikka kenttä voisi sijaitakin lomakkeella, se ei varsinaisesti ole hankintasopimukseen liittyvä asia. Lisäksi ikkuna toimii muistutuksena käyttäjälle siitä, että hankintasopimuksen on hyvä olla jo lisättynä Sentraaliin.

Keskeneräinen hankintasopimus aukeaa uuden hankintasopimuksen luontilomakkeelle. Lomakkeen toiminnallisuus ei muutu, mutta sen alkuun ilmestyy kaksi lisäpainiketta ja -paneelia (Kuva 30). Painikkeet ilmestyvät sivun vasempaan ylälaitaan ja niiden avulla voidaan liittää liitetiedostoja hankintasopimukselle tai saada tulostukseen paremmin soveltuva näkymä uuteen välilehteen. Liitteet tallentuvat palvelimelle kansiorakenteeseen, jossa kansio nimetään hankintasopimuksen id-sarakkeen mukaan. Vasemmanpuoleinen paneeli on hintaliitteen tarkastamista varten ja se on käytettävissä vain Ostaja- tai Hyväksyjä-roolin omaavilla henkilöillä. Hylätty päätös näytetään käyttäjälle punaisella rivillä, hyväksytty vihreällä ja tarkastamaton harmaalla. Oikeanpuoleinen paneeli sisältää puolestaan kaikki hankintasopimusta tallentaneet henkilöt aikaleimoineen. Jokaisen henkilön nimen edessä on painike, jota painamalla hänelle voidaan lähettää sähköpostia.

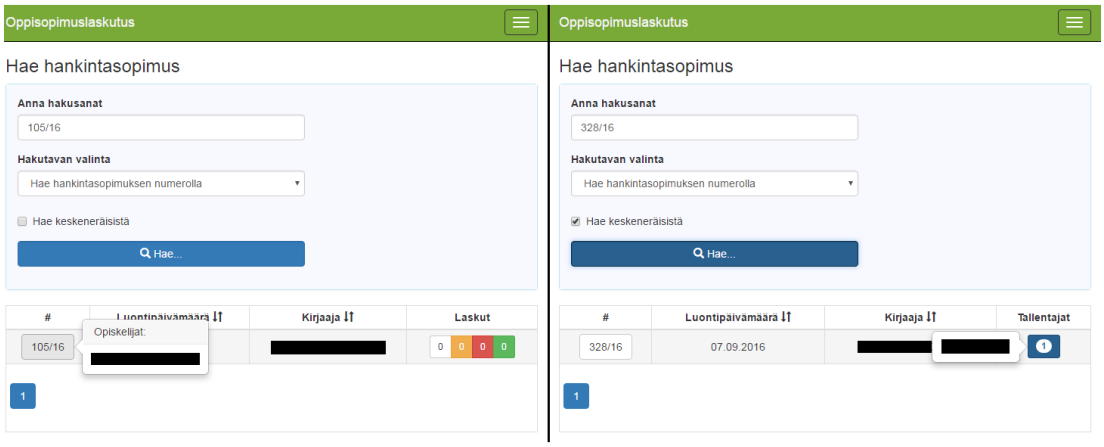


Kuva 30. Tallennettu ja keskeneräinen hankintasopimus. Keltainen painike avaa modaali-ikkunan liitteiden lisäämistä varten, kun taas vieressä oleva sininen painike avaa hankintasopimuksen ja sen hintaliitteet uuteen välilehteen tulostusystävällisempään näkymään. Tarkastukset-paneeli toimii myös modaali-ikkunaperiaatteella: ”Tarkasta”-painikkeesta aukeaa erillinen ikkuna, johon päätös kirjataan kommentteineen.

5.5.2 Hae hankintasopimus

”Hae hankintasopimus”-välilehden näkymä on todella pelkistetty hakusivu (Kuva 31). Käytännössä hankintasopimuksia voidaan hakea muutamalla hakusanaehdolla ja määrittelemällä, halutaanko hakea keskeneräisistä hankintasopimuksista vai jo laskutuksessa olevista. Hakusanaehdoissa on muun muassa numerolla, kirjaajan nimellä ja opiskelijan nimellä hakeminen.

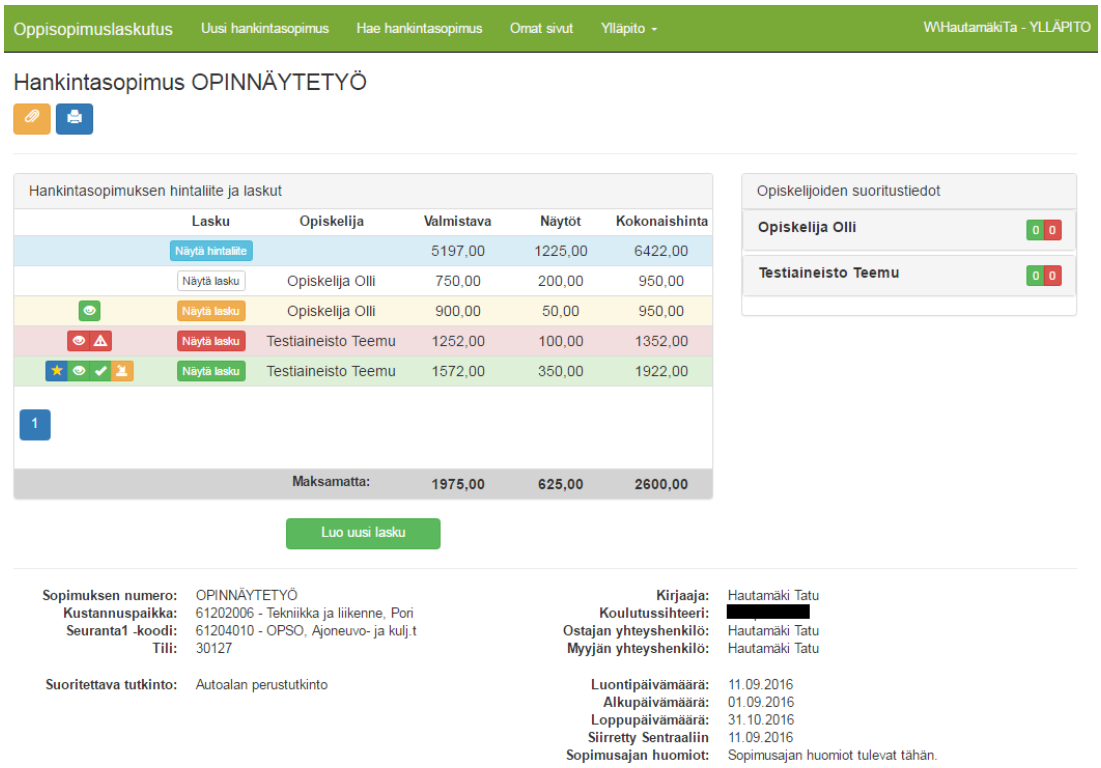
Hakutulos näytetään sivutettuna tauluna, jossa jokaisella sivulla näytetään enimmillään 20 hankintasopimusta. Keskeneräisissä hankintasopimuksissa näytetään tallentajat ja heidän lukumääränsä, kun taas laskutuksessa olevista hankintasopimuksista näytetään eri tilassa olevien laskujen lukumäärä. Molemmissa tapauksissa hankintasopimuksen numero-painikkeesta siirrytään haluttuun sopimukseen. Lisäksi kyseiseen painikkeeseen on lisätty Bootstrapin data-toggle -attribuuteilla popover-ikkuna (Bootstrap, 2016c), joka näyttää sopimukseen liitetyt opiskelijat osoittimen ollessa painikkeen päällä. Popoveria käytetään myös keskeneräisten hankintasopimuksien tallentajien nimien esittämiseen.



Kuva 31. Kaksi erilaista hakutulosta rinnakkain, joissa on haettu hankintasopimuksen numerolla yksi hankintasopimus. Laskutuksessa olevasta hankintasopimuksesta laskujen lukumäärä esitetään eritellysti niiden vaiheiden mukaan, joissa värittömät ovat tarkastamattomia, oranssit tarkastettuja, punaiset hylättyjä ja vihreät hyväksyttyjä laskuja. Lisäksi kuvissa näkyy popover-elementit sekä opiskelijoille että tarkastajille.

5.5.3 Laskutus

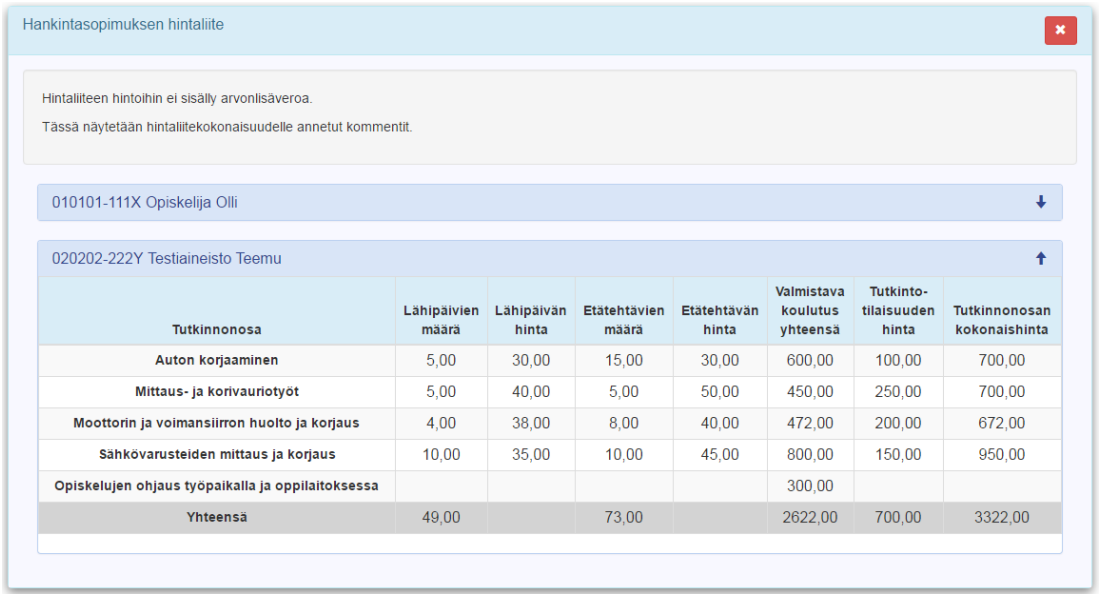
Laskutusnäkymässä on pyritty tuomaan kaikki laskutusta koskevat tiedot mahdollisimman selkeästi esille (Kuva 32). Hintaliite, luodut laskut, maksamatta olevat määrät ja opiskelijoiden suoritustiedot ovat näkyvimmässä roolissa, kun taas hankintasopimuksen lomaketiedot on siirretty aivan sivun alalaitaan. Laskut on esitetty taulussa, jossa hintaliite on aina ylimpänä. Itse laskut on sivutettu samalla tavalla, kuin hakunäkymässä, näyttäen enimmillään kahdeksan laskua per sivu. Opiskelijoiden suoritustiedot näytetään paneelikokoelmana, jossa jokaisen opiskelijan suoritukset on asetettu oman laajennettavan paneelinsa sisälle. Opiskelijan nimen vieressä vihreällä taustalla oleva luku ilmoittaa hyväksyttyjen tutkinnonosasuoritusten määrän, kun taas punaisella taustalla oleva luku kertoo hylättyjen suoritusten määrän.



Kuva 32. Käyttäjälle avautuva näkymä hankintasopimuksen laskutuspuolesta. Hankintasopimuksesta voidaan avata tulostettava versio ja sen liitteitä voidaan hallinnoida vielä laskutusnäkymässäkin. Hylättyjä laskuja ei lasketa mukaan ”Maksamatta”-rivin määriin.

Näkymässä on käytännössä kolme tärkeää toiminnallista osaa: hintaliitteen tarkasteleminen, laskujen hallinnointi ja uusien laskujen luominen. Näille jokaiselle on oma paneelinsa, johon tarvittava tieto haetaan Ajax-kutsulla. Kutsun valmistuttua sivussa oleviin piilotettuihin paneeleihin luodaan osittaisten näkymien avulla sisältö, jonka jälkeen paneeli näytetään käyttäjälle. Paneelit ja niiden siirtymätehosteet on puolestaan tyylitelty ikkunoiksi, jotka asettuvat sivulle allekkain. Käyttäjällä voi siis olla tarvittaessa avoinna saman aikaisesti sekä hankintasopimuksen hintaliite, yksi lasku sekä uuden laskun luontilomake. Käyttäjä voi sulkea paneelin joko paneelin omasta sulkemisnapista tai aktivoimalla uudelleen paneelin avanneen toiminnon.

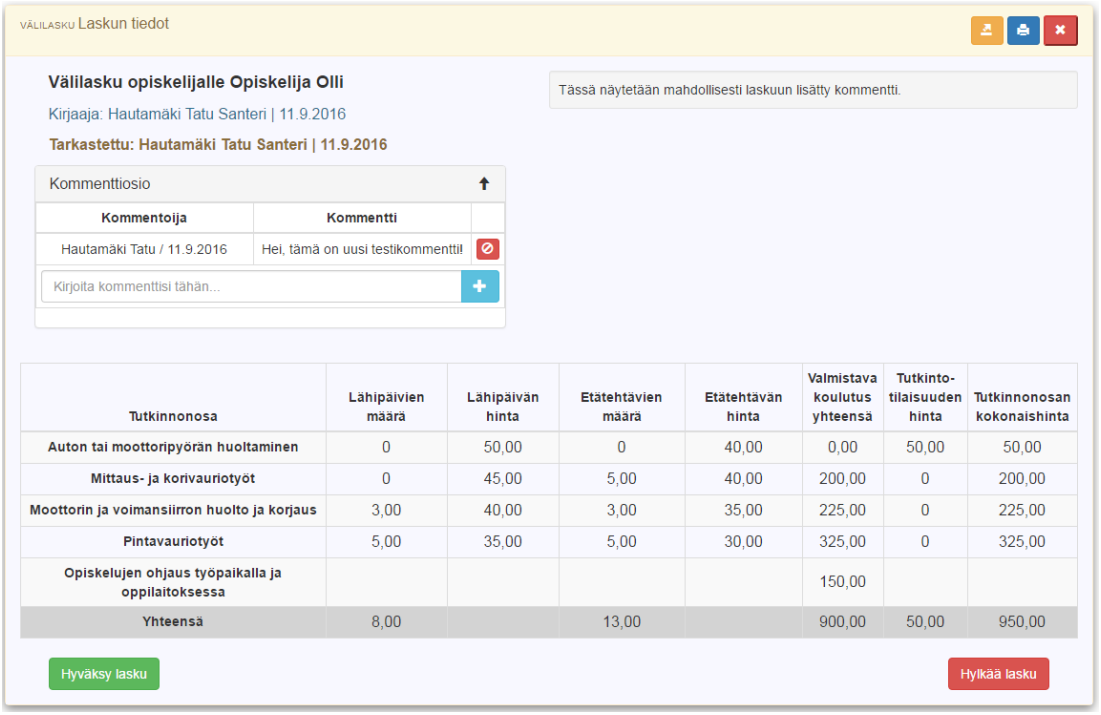
”Näytä hintaliite”-painikkeesta käyttäjälle näytetään hankintasopimuksen hintaliite (Kuva 33). Paneelissa näytetään hintaliitteelle mahdollisesti annettu kommentti ja jokaisen opiskelijan hintaliite omana laajennettavana paneelinaan. Ulkoasu ja toiminnallisuus muistuttaa siis todella paljon uuden hankintasopimuksen luonnin yhteydessä olevia opiskelijakohtaisia hintaliitteitä.



Tutkinnonosa	Lähipäivien määrä	Lähipäivän hinta	Etätehtävien määrä	Etätehtävän hinta	Valmistava koulutus yhteensä	Tutkinto-tilaisuuden hinta	Tutkinnonosan kokonaishinta
Auton korjaaminen	5,00	30,00	15,00	30,00	600,00	100,00	700,00
Mittaus- ja korivauriotyöt	5,00	40,00	5,00	50,00	450,00	250,00	700,00
Moottorin ja voimansiirron huolto ja korjaus	4,00	38,00	8,00	40,00	472,00	200,00	672,00
Sähkövarusteiden mittaus ja korjaus	10,00	35,00	10,00	45,00	800,00	150,00	950,00
Opiskelujen ohjaus työpaikalla ja oppilaitoksessa					300,00		
Yhteensä	49,00		73,00		2622,00	700,00	3322,00

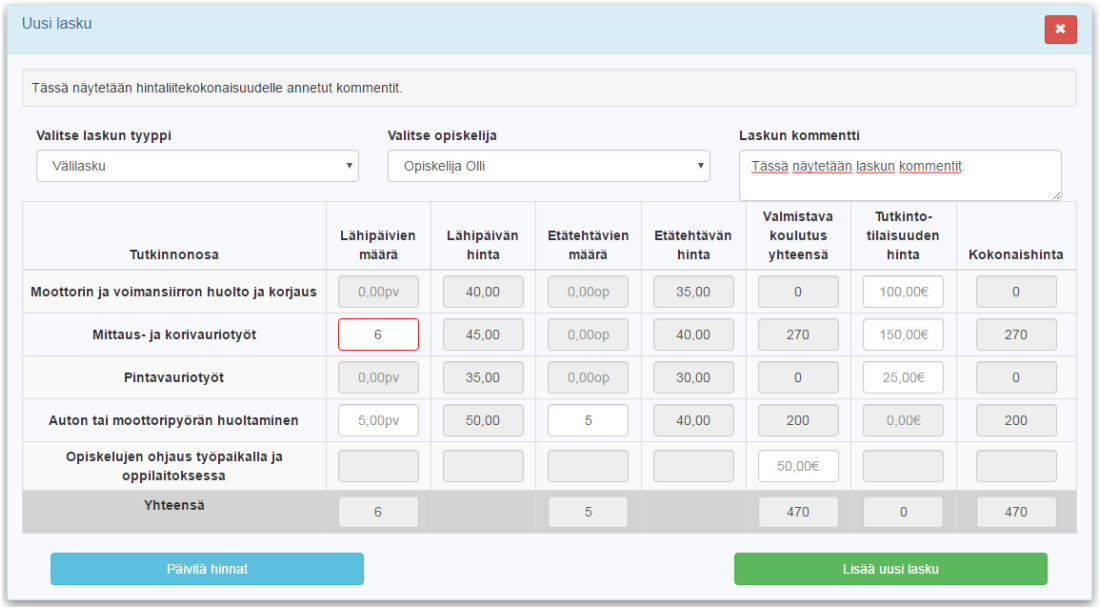
Kuva 33. Laskutusnäkymässä näytettävä hintaliite. Paneeli voidaan sulkea joko painamalla punaista nappia paneelin oikeassa ylälaidassa tai painamalla uudelleen ”Näytä hintaliite”-painiketta, jossa lukee nyt hintaliitteen ollessa auki ”Piilota hintaliite”.

Lasku avataan painamalla laskun kohdalla ”Näytä lasku”-painiketta. Jokainen laskurivi ja laskupaneeli on värikoodattu samalla tavalla, kuin hakusivun erittelyssäkin. Lisäksi jokaisen rivin alussa näytetään Bootstrapin glyphiconeja hyödyntäen lisätietoja laskun tilasta. Tähdellä tarkoitetaan loppulaskua, silmällä tarkastusta, hyväksytty-ruksilla hyväksyntää, punaisella vaaramerkillä hylkäystä ja keltaisella nuolella sitä, että lasku on siirretty SopPro-järjestelmään. Jokaisesta edellä mainitusta tapahtumasta saa lisätietoa siirtämällä osoittimen kuvakkeen päälle, jolloin ilmestyvästä popover-ikkunasta voi tarkistaa esimerkiksi tapahtuman ajankohdan ja tekijän. Laskupaneelin oikeassa ylälaidassa on punaisen sulkemispainikkeen lisäksi laskun tulostusnäkymäpainike sekä SopPro-siirtopainike, josta aukeavan modaali-ikkunan kautta lasku voidaan merkata siirretyksi SopPro-järjestelmään (Kuva 34).



Kuva 34. Tarkastettu lasku kommentteineen. Vain kommentin lisännyt käyttäjä voi poistaa oman kommenttinsa. Käyttäjä voi myös tarvittaessa poistaa oman hylätyn laskunsa.

Uusi lasku luodaan painamalla ”Luo uusi lasku”-painiketta. Ilmestyvään paneeliin (Kuva 35) valitaan laskun tyyppi, joka on joko välilasku tai loppulasku sekä laskutettava opiskelija. Opiskelijan valinnan jälkeen palvelimelta haetaan hänen laskutettavat tutkinnonosansa, joissa näytetään placeholder-attribuutissa jäljellä olevat määrät. Jos nämä määrät ylitetään, niin käyttäjälle ilmoitetaan välittömästi validointivirheestä punaisella reunuksella input-elementin ympärillä ja estetään laskun luonti. Jos käyttäjä kiertää tämän selainvalidoinnin tai toinen käyttäjä tekee samanaikaisesti laskun, niin validointivastuu siirtyy kokonaan laskun lisäävän ohjaimen toimintoon ja tietokannan laukaisimelle.

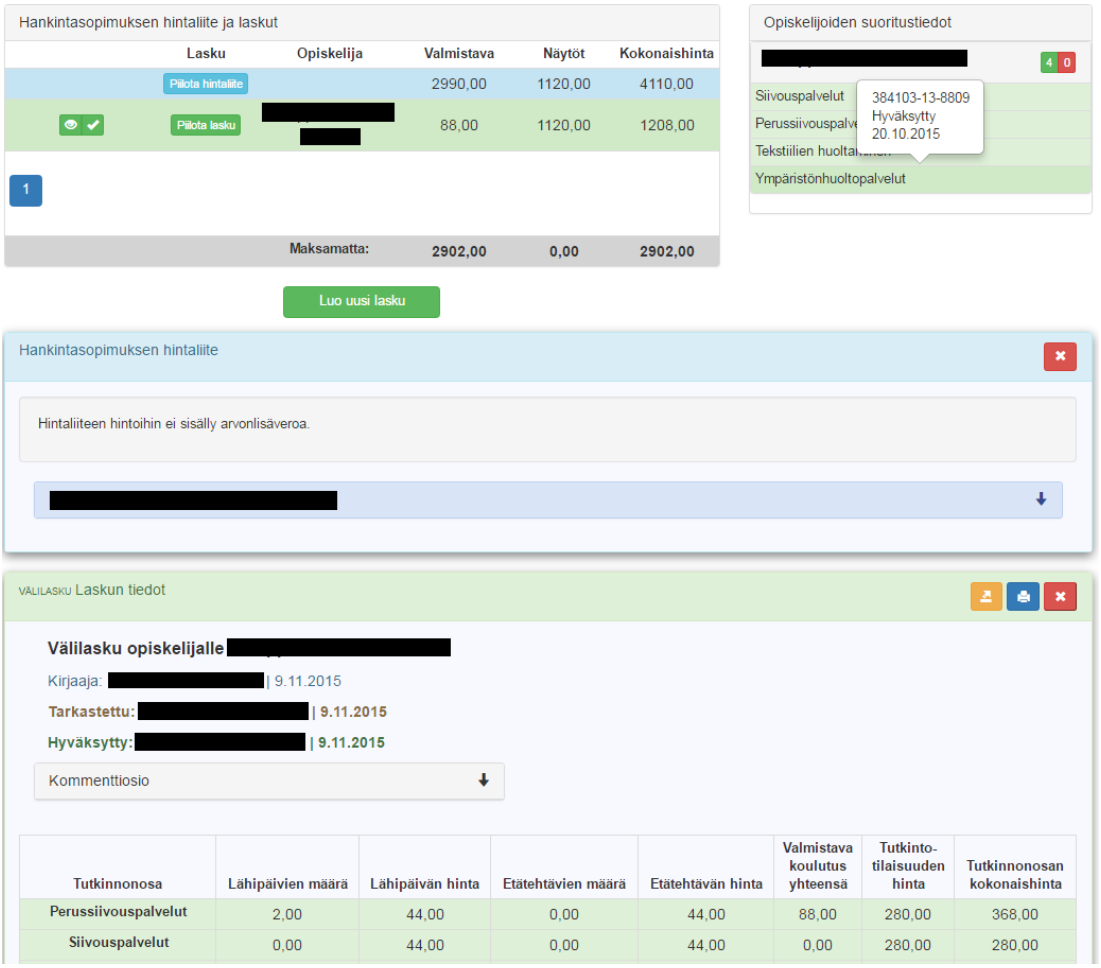


Kuva 35. Kuvankaappaus uuden laskun lisäämisestä. Laskupohjassa näytetään hintaliitteen kommentti. ”Lähipäivän hinta”- ja ”Etätehtävien määrä”-kenttien arvot haetaan hintaliitteeltä. Kenttiä myös poistetaan käytöstä, jos kaikki saatavilla olevat määrät ja hinnat on jo laskutettu.

Laskutuksessa on myös kokeiltu massaominaisuuksia, jossa opiskelija-alasvetovalikosta voi valita myös ”Tee lasku kaikille opiskelijoille”-vaihtoehdon. Tällöin laskuriveihin haetaan kaikille opiskelijoille käyvät määrät ja hinnat huomioiden myös opiskelijakohtaisesti eriävät tutkinnonosat. Käytännössä riveihin haetaan siis yhteisistä tutkinnonosista jäljellä olevat minimimäärät ja jokaisen opiskelijan henkilökohtaiset tutkinnonosat. Tämä massaominaisuus on kuitenkin vielä testaamatta käyttäjillä, eli siihen tulee todennäköisesti muutoksia.

Jokaisessa laskutuksen toiminnallisuudessa on myös yhteys opiskelijoiden tutkinnonosasuorituksiin. Jos opiskelija on suorittanut hyväksytysti tutkinnonosan, niin hänellä näytetään niin hintaliite- kuin laskuriveissäkin kyseinen tutkinnonosarivi vihreällä taustalla. Hylätyn suorituksen tapauksessa rivien taustaväri on punainen.

Seuraavan sivun kuvassa käyttöliittymää esitellään tilanteessa, jossa opiskelijan suoritustiedot, hintaliite ja yksi lasku on avattu samanaikaisesti näkyville (Kuva 36).



Kuva 36. Laskunäkymä, jossa opiskelijan suoritustiedot, hintaliite ja yksi lasku on avattu näkyville. Laskun rivit ovat vihreät, koska kyseinen opiskelija on suorittanut kaikki tutkinnonosat hyväksytysti.

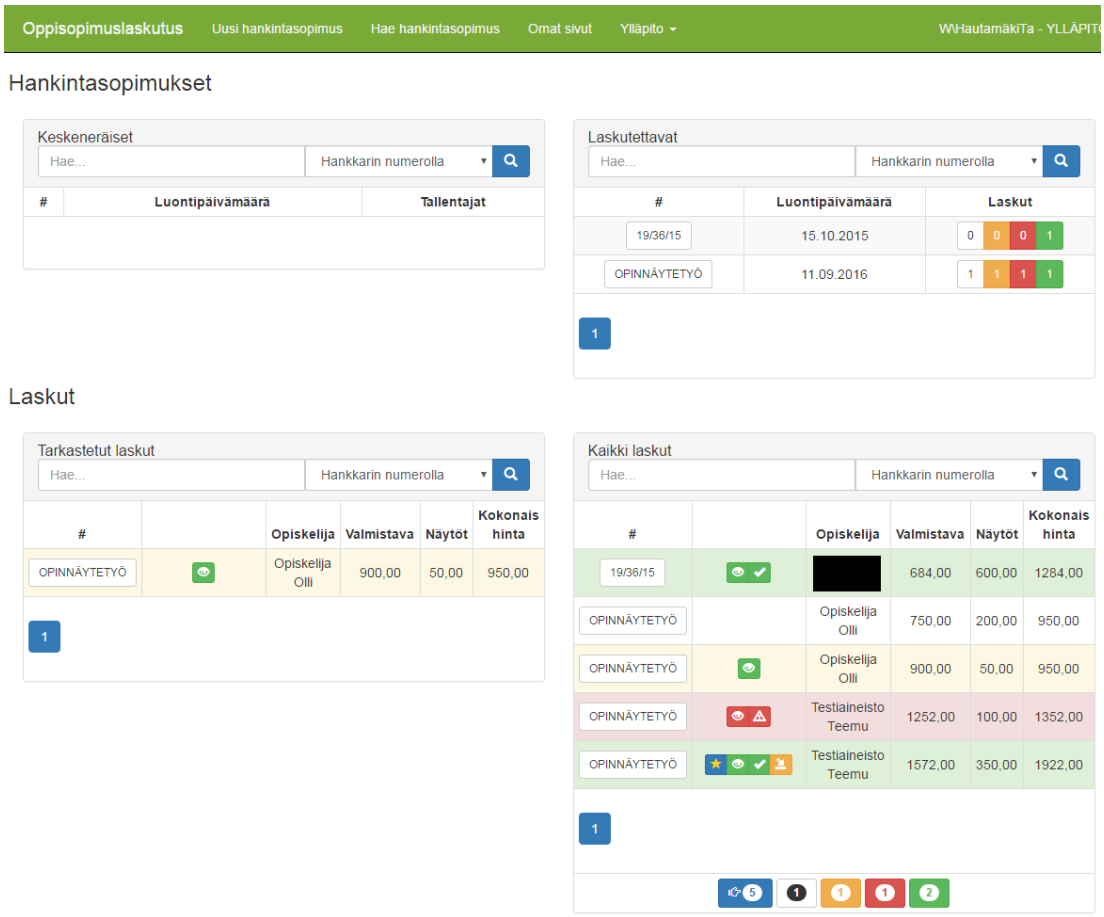
5.5.4 Omat sivut

Sovelluksen ”Omat sivut”-välilehti toimii myös sovelluksen aloitussivuna (Kuva 37). Se luotiin tukemaan prosessia. Käyttäjälle ei ole mieluista aina lähteä hakusivun kautta etsimään hankintasopimuksia tai tallentamaan niiden URL-osoitteita kirjanmerkkeihinsä. Käyttäjän ei tulisi myöskään tarvita ulkoisia muistiinpanoja tai sähköpostin selailua selvittääkseen, mikä lasku hänen pitikään tarkastaa tai mikä hintaliite täyttää. Näkymässä pyritään näyttämään siis käyttäjää kiinnostavia asioita hänen roolinsa perusteella. Jokaisessa näkymän taulussa on oma sisäänrakennettu hakutoiminto, jonka avulla hankintasopimuksia tai laskuja voidaan hakea hankintasopimuksen numeron tai

opiskelijoiden nimien perusteella. Lisäksi osassa laskuja sisältävistä tauluista laskuja voi eritellä niiden tilan mukaan.

Hyväksyjä- ja Tarkastaja-roolit näkevät yhteensä neljä taulua, joista kaksi on varattu hankintasopimuksille ja kaksi laskuille. Muut roolit näkevät vain kaksi taulua. Käyttäjät näkevät roolinsa perusteella seuraavat asiat:

- Hyväksyjä-roolissa olevat käyttäjät näkevät omat keskeneräiset ja laskutuksessa olevat hankintasopimuksensa. Laskuista he näkevät kaikkien hankintasopimuksiensa laskujen lisäksi kaikki tarkastetut laskut, jotka kaipaavat heidän hyväksymistoimenpidettään.
- Tarkastaja-roolissa olevat käyttäjät näkevät kaikki keskeneräiset ja laskutuksessa olevat hankintasopimukset, joissa he ovat valittuna myyjän yhteyshenkilöksi. Lisäksi he näkevät kaikki edellä mainittujen hankintasopimuksien laskut ja vielä eriteltynä ne laskut, jotka odottavat heidän tarkastustaan.
- Ostaja-roolin omaavat käyttäjät näkevät kaikki keskeneräiset hankintasopimukset, joissa he ovat merkittynä ostajan yhteyshenkilöksi ja joiden hintaliitteissä on hintatietoja. Näiden hankintasopimuksien hintaliitteet odottavat heidän tarkastustaan. Lisäksi he näkevät kaikki keskeneräiset hankintasopimukset, joihin he ovat jo suorittaneet tarkastuksen.
- Laskuttaja-roolissa olevat käyttäjän näkevät kaikki heidän tallentamansa hankintasopimukset ja laskut.



Kuva 37. Hyväksyjä-roolin omaavalle käyttäjälle avautuva näkymä. Jokainen taulu on sivutettu näyttäen enimmillään kahdeksan riviä per sivu. Taulujen ulkoasu ja logiikka noudattavat sovelluksen yleistä teemaa. Hankintasopimusrivit näyttävät samalta, kuin hakusivulla. Laskut puolestaan muistuttavat laskutuspuolen laskulistauksen ulkoasua. Jos laskun hankintasopimukseen siirrytään laskurivin hankintasopimusnumerosta, niin käyttäjälle avataan välittömästi laskutukseen siirryttäessä myös valittu lasku.

5.5.5 Ylläpidon toiminnot

Ylläpidolle on tällä hetkellä saatavilla kaksi toimintoa: hankintasopimuksien poistaminen ja muokkaaminen. Hankintasopimuksia voidaan poistaa, jos niiden luonti on kesken tai jos niissä ei ole vielä yhtäkään laskua. Poistamista kuitenkin käytetään vain todella harvoin, yleensä vain niissä tapauksissa, joissa opiskelijan oppisopimussopimus raukeaa jo ennen aikojaan. Hankintasopimuksien muokkaaminen puolestaan liittyy pikemminkin opiskelijoiden hintaliitteiden tutkinnonosavalintoihin. Opiskelija voi vaihtaa valinnaista tutkinnonosaansa vielä hankintasopimuksen lopullisen lisäämisen

jälkeenkin. Tällaisissa tapauksissa ylläpito voi vaihtaa opiskelijan hintaliitteen tutkinnonosaa, jos siihen ei kohdistu aktiivisia laskuja. Muussa tapauksessa muutokset on tehtävä tietokantatasolla. Tutkinnonosan vaihtaminen sen laskuttamisen jälkeen on kuitenkin käsitteellisesti väärin, koska vaihdon jälkeen myös edelliset laskut kohdistuvatkin uuteen tutkinnonosaan vanhan sijasta.

6 YHTEENVETO

Aluksi yksinkertaiseksi suunnitellusta sovelluksesta paisui lopulta tuhansia rivejä koodia sisältävä valtava kokonaisuus. Myös kehityspotentiaalia on, sillä esimerkiksi SopPro:n kanssa on suunnitteilla rajapinnat, joiden avulla hyväksyttyjä laskuja saisi kirjattua automaattisesti heidän järjestelmäänsä. Suurimmat haasteet projektissa olivat koodimassojen järkevä hallinta ja heikosta suunnittelu- ja määrittelyvaiheesta johtuneet korjaustyöt. Huolellisemmin tehty suunnittelu ja määrittely sekä käyttäjälähtöinen suunnittelu olisivat aikaistaneet sovelluksen valmistumista usealla kuukaudella. Kärjistettynä voidaan sanoa, että jokainen tunti joka säästettiin suunnitteluvaiheessa maksettiin myöhemmin moninkertaisesti takaisin korjausten yhteydessä. Kunnollisen vaatimusmäärittelyn ja suunnittelun tärkeyttä ei siis koskaan pidä väheksyä.

Projekti ja siitä syntynyt sovellus olivat oman ammatillisen osaamiseni kehittymisen kannalta todella tärkeitä. Suurin osa projektissa käytetyistä tekniikoista ja käsitteistä olivat minulle tuttuja jo opinnoistani, mutta niiden konkreettinen hyödyntäminen työelämässä oli todella opettavainen kokemus. Suunnittelen ja teen asioita nykyään johdonmukaisesti kokonaisuutta ja tulevaisuutta silmällä pitäen, enkä pyri vain äkkiä saamaan jotakin tulosta aikaan epämääräisellä näpertelyllä. Otan myös nykyisissä projekteissani aktiivisesti käyttäjiä mukaan suunnittelu- ja kehitysvaiheeseen. Tulokset puhuvat puolestaan ja projektit valmistuvat huomattavasti nopeammin eikä niihin tarvitse tehdä suuria korjauksia enää käyttöönoton jälkeen. Suunnittelen myös tietokantoja huomattavasti perusteellisemmin ja huolellisemmin, kuin harjoittelujaksoni alussa. Huomaan nykyään jo suunnitteluvaiheessa, jos tietokannassa on jokin käsitteellisesti väärin tai jos jokin sen rakenteessa aiheuttaa ongelmia myöhemmin sovelluksessa tai tietokannan ylläpidossa.

Sovelluksen hankintasopimuspuoli on tällä hetkellä päivittäisessä käytössä asiakasyrityksessä. Laskutuksessa on kuitenkin ongelmia sen käyttöönoton kanssa, sillä henkilöstöä ei missään vaiheessa koulutettu kunnolla sovelluksen käyttöön. Laskutuksessa on huomattavasti enemmän toimijoita kuin hankintasopimusten hallinnassa, minkä vuoksi ei voida olettaa tiedon välittyvän eteenpäin laskuttajille ilman virallisia koulutuksia ja ohjeistusta. Riskinä on se, että yrityksessä palataan takaisin paperisiin laskutuskortteihin välinpitämättömyyden tai tietämättömyyden takia, jolloin koko prosessi jää selvittämättömään välitilaan ja sovellus unohdetaan laskutuspuolen käyttämättömyyden takia. Tätä torjumaan järjestetyt laskuttajille tähdätyt koulutukset kuitenkin toivottavasti korjaavat asian ja pitävät sovelluksen aktiivisessa käytössä.

LÄHTEET

Oppisopimuskeskus 2015. WinNova. Pori. Haastattelu 21.8.2015. Haastattelijana Tatu Hautamäki. Muistiinpanot haastattelijan hallussa.

WinNova 2012. Oppisopimuskoulutuksen laskutus. WinNovan IMS-järjestelmä. Viitattu 10.7.2016.

Microsoft, 2016a. ASP.NET Overview. Viitattu 11.7.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/4w3ex9c2.aspx

Microsoft, 2016b. ASP.NET MVC Overview. Viitattu 15.7.2016. Saatavissa: http://www.asp.net/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

Microsoft, 2016c. Walkthrough: Creating a Web Site using Razor Syntax in Visual Studio. Viitattu 15.7.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/gg606533(v=vs.100).aspx

Koskimies, K. & Mikkonen, T. 2005. Ohjelmistoarkkitehtuuri. Jyväskylä: Talentum.

Codeproject, 2016. Introduction to ASP.NET MVC. Viitattu 15.7.2016. Saatavissa: http://www.codeproject.com/Articles/188226/Introduction-to-ASP-NET-MVC

Microsoft, 2016d. Global.asax File. Viitattu 17.7.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/1xaas8a2(v=vs.71).aspx

Microsoft, 2016e. ASP.NET Routing. Viitattu 17.7.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/cc668201.aspx

Microsoft, 2016f. Filtering in ASP.NET MVC. Viitattu 20.7.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/gg416513(VS.98).aspx

Microsoft, 2016g. Bundling and Minification. Viitattu 20.7.2016. Saatavissa: http://www.asp.net/mvc/overview/performance/bundling-and-minification

Haikala, I. & Märijärvi J. 2001. Ohjelmistotuotanto. 7. p. Vantaa: Talentum Media Oy.

Gamma, E., Helm, R., Johnson, R. & Vissides, J. 2009. Design Patterns: Elements of Reusable Object-Oriented Software. 37. p. Addison-Wesley.

Microsoft, 2016h. The Repository Pattern. Viitattu 24.7.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/ff649690.aspx

McFarland, D. 2008. JavaScript: The Missing Manual. O'Reilly Media, Inc.

jQuery UI, 2016. About jQuery UI. Viitattu 25.7.2016. Saatavissa: http://jqueryui.com/about/

Nieminen, H. 2015. Palvelinohjelmointi, osa II.

Wilson, Brad. ’Unobtrusive Ajax in ASP.NET MVC 3’. Brad Wilson. 6.10.2010a. Viitattu 25.7.2016. http://bradwilson.typepad.com/blog/2010/10/mvc3-unobtrusive-ajax.html

Wilson, Brad. ’Unobtrusive Client Validation in ASP.NET MVC 3’. Brad Wilson. 6.10.2010b. Viitattu 25.7.2016. http://bradwilson.typepad.com/blog/2010/10/mvc3-unobtrusive-validation.html

Pars, R., Moroney, L. & Grieb, J. 2007. Foundations of ASP.NET AJAX. Apress.

jQuery 2016. jQuery.ajax(). Viitattu 27.7.2016. Saatavissa: http://api.jquery.com/jquery.ajax/

Modernizr, 2016. What is Modernizr? Viitattu 27.7.2016. Saatavissa: https://modernizr.com/docs/#what-is-modernizr

Can I Use, 2016. Can I use input date? Viitattu 27.7.2016. Saatavissa: http://caniuse.com/#search=input%20date

Rascia, T. ’What is Bootstrap and How Do I Use It?’. Tania Rascia. 9.11.2015. Viitattu 28.7.2016. https://www.taniarascia.com/what-is-bootstrap-and-how-do-i-use-it/

Bootstrap, 2016a. Grid system. Viitattu 28.7.2016. Saatavissa: http://getbootstrap.com/css/#grid

Haikala, I. & Mikkonen T. 2011. Ohjelmistotuotannon käytönnöt. 12. p. Helsinki: Talentum Media Oy.

Sommerville, I. 2004. Software Engineering. 7. p. USA: Pearson Education Limited.

Hovi, A., Huotari, J. & Lahdenmäki, T. 2005. Tietokantojen suunnittelu & indeksointi. 1. p. Jyväskylä: Docendo Finland Oy.

Elmasri, R. & Shamkant, B. 2004. Fundamentals of Database Systems, Fourth Edition. 4. p. USA: Pearson Education, Inc.

Oracle, 2016. SQL Developer Data Modeler. Viitattu 30.8.2016. Saatavissa: http://www.oracle.com/technetwork/developer-tools/datamodeler/overview/index.html

Microsoft, 2016i. Using Data Annotations for Model Validation. Viitattu 1.9.2016. Saatavissa: http://www.asp.net/mvc/overview/older-versions/mvc-music-store/mvc-music-store-part-6

Microsoft, 2016j. ASP.NET Application Life Cycle Overview for IIS 7.0. Viitattu 4.9.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/bb470252.aspx

Microsoft, 2016k. ASP.NET Session State Overview. Viitattu 4.9.2016. Saatavissa: https://msdn.microsoft.com/en-us/library/ms178581(v=vs.110).aspx

Microsoft, 2016l. AuthorizeAttribute Class. Viitattu 4.9.2016. Saatavissa: https://msdn.microsoft.com/fi-fi/library/system.web.mvc.authorizeattribute(v=vs.118).aspx

Bootstrap, 2016b. JavaScript. Viitattu 12.9.2016. Saatavissa: http://getbootstrap.com/javascript

LIITE 1

HANKINTASOPIMUSLOMAKE

OPPISOPIMUSKOULUTUKSEN HANKINTASOPIMUS Nro

Palautettava Satakunnan oppisopimuskeskukseen . . mennessä.

Sopijaosapuolet LASKUTUS-OSOITE Yhteyshenkilö Puhelin Sähköposti Lisätietoja	KOULUTUKSEN JÄRJESTÄJÄ / OSTAJA: **Länsirannikon Koulutus Oy WinNova** **y-tunnus 2245018-4** **Satakunnan oppisopimuskeskus** **PL 17, 28101 PORI** **044** **@winnova.fi** SUU	PALVELUNTUOTTAJA / MYYJÄ:
Opiskelijatiedot	Nimi: Osoite:	Opiskelijaluettelo liitteenä ☐
Työnantajatiedot	Nimi: Osoite: Puhelin:	Yhteyshlö: Sähköposti: Vastuullinen kouluttaja:
Oppisopimusaika		
Järjestettävä koulutus	☐ Koko tutkintoon valmistava koulutus ja näyttötutkinto ☐ Tutkinnon osaan valmistava koulutus ja näyttötutkinnon osa ☐ Lisäkoulutus, joka ei johda tutkintoon	
Hankinnan hinta	**Hankinnan kokonaishinta <u>henkilökohtaistamisasiakirjan/-asiakirjojen</u> mukaan:** Valmistavan koulutuksen hinta _____€ Näyttötutkinnon suoritushinta _____€ **Kaikki kustannukset yhteensä** _____€ ☐ *Hinnat sisältävät alv:a %:a.* **Hinnan muodostuminen esitettävä tarkemmin liitteessä 2 (jokaisesta opiskelijasta laadittava oma).** Opiskelijan ja/tai työnantajan erityistarpeista johtuen, voidaan harkinnan perusteella erillisellä sopimuksella sopia tätä sopimusta täydentävästä lisähankinnasta.	
Lisätietoja	***Palveluntuottajan tulee ottaa yhteyttä työpaikkakouluttajaan ja opiskelijaan kahden (2) viikon kuluessa saatuaan tiedon oppisopimuksesta.** ***Satakunnan oppisopimuskeskuksen ohjeistuksen mukaan tehty henkilökohtaistamisasiakirja on toimitettava Satakunnan oppisopimuskeskukseen kolmen (3) kuukauden kuluessa tämän hankintasopimuksen toimittamisesta palveluntuottajalle. Laatiminen sisältyy valmistavan koulutuksen hintaan. Mikäli oppisopimus puretaan kolmen (3) kuukauden kuluessa oppisopimuksen alkamispäivästä, ohjeistuksen mukaan tehdystä henkilökohtaistamisasiakirjasta maksetaan erikseen 350 euroa.** ***Hankintasopimusta EI HYVÄKSYTÄ ILMAN HENKILÖKOHTAISTAMISASIAKIRJAA.**	

LIITE 2

HINTALIITELOMAKE

	Palvelun tuottaja täyttää							
Satakunnan oppisopimuskeskus								
Tutkinnon suorittajan nimi								
Työnantajan nimi								
Tutkinnon nimi								
Tutkinnon osat	Lähipäivien määrä	Lähipäivän hinta	Etätehtävien laajuus	Etätehtävistä aiheutuvat kustannukset	**Valmistava koulutus yhteensä**	Tutkinto-tilaisuuden hinta	**Tutkinnon osan *koko-naishinta***	**Mahdollista täsmennettävää**
Koulutuksen määrät / hinta yhteensä								
Havaitut lisäohjaustarpeet	- selvitys							
	- tuntihinta (ei vaikuta koulutuksen kokonaishintaan)							
Henkilökohtaistaminen	- palveluntuottaja laatii Satakunnan oppisopimuskeskuksen ohjeiden mukaisesti							
	- mikäli oppisopimus puretaan kolmen (3) kuukauden kuluessa oppisopimuksen alkamispäivästä, Satakunnan oppisopimuskeskuksen ohjeistuksen mukaan tehdystä henkilökohtaistamisasiakirjasta maksetaan erikseen 175 euroa (osatutkinto) tai 350 euroa (kokotutkinto).							
Valmistava koulutus sisältää								
	– lähi- ja etäopiskelua (etäopiskelua 10 - 20 % lähiopetuksen määrästä)							
	– työssäoppimisen tukemista ja ohjausta							
	– tietopuolisen oppimisen ohjausta ja arviointia							