HAAGA-HELIA
ammattikorkeakoulu

Opinnäytetöiden seurantajärjestelmän (ONT) toteutus Microsoft .NET -tekniikalla

Marko Ruotsalainen

Raportti
Tietojenkäsittelyn koulutusohjelma
2011

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

3.10.2011

Tietojenkäsittelyn koulutusohjelma

Tekijä tai tekijät Marko Ruotsalainen	**Ryhmä tai aloitusvuosi** 2008
Opinnäytetyön nimi Opinnäytetöiden seurantajärjestelmän (ONT) toteutus Microsoft .NET -tekniikalla	**Sivu- ja liitesivumäärä** 49 + 6
Ohjaaja tai ohjaajat Juhani Välimäki	

Tämä raportti kertoo mikä ONT-seurantajärjestelmä on ja miksi sitä tarvitaan HAAGA-HELIAn ammattikorkeakoulussa uutena tietojärjestelmänä. Raportissa selitetään myös järjestelmän toimintaa käyttäjän kannalta. ONT-seurantajärjestelmällä opiskelijat, ohjaajat ja koordinaattorit voivat helposti ja järjestelmällisesti seurata ja hallita koko opinnäytetyöprosessia sekä viestiä keskenään sen välityksellä.

Raportissa kerrotaan mitä tekniikkaa on käytetty ONT-seurantajärjestelmää kehitettäessä sekä selitetään miksi on päädytty käyttämään ”Microsoft ASP.NET Web Forms”-kehitysmallia web-sovelluksen toteuttamiseen ja miksi SQL Server 2005 on valittu sovelluksen taustalla olevaksi tietokannanhallintajärjestelmäksi.

Raportti esittää eri ratkaisuja, joita on käytetty ONT-seurantajärjestelmän toiminnallisuuksien ja tietorakenteiden saavuttamiseksi. Siinä kerrotaan mm. millaista arkkitehtuuria on käytetty suunnittelussa, miten sovelluksen käyttöliittymän, sovelluslogiikan, tietokantayhteyden ja muiden teknisten ominaisuuksien toteutukset on tehty, jotta ne palvelisivat asiakkaan (HAAGA-HELIA ammattikorkeakoulun) vaatimia määrityksiä.

Lopuksi pohditaan projektin onnistumista ja tuloksia sekä esitetään ehdotuksia ONT–seurantajärjestelmän jatkokehittämisestä. Tämän lisäksi pohditaan myös projektin merkitystä työn tekijän ammatilliselle kasvulle.

Asiasanat
ASP.NET, Web Forms, SQL Server, Olio-ohjelmointi, Tietojärjestelmä, web-sovellus

HAAGA-HELIA
University of Applied Sciences

Abstract

Date of presentation

Degree programme

Author or authors Marko Ruotsalainen	**Group or year of entry** 2008
The title of thesis Implementation of Thesis management system with Microsoft .NET technology	**Number of pages and appendices** 49 + 6
Supervisor or supervisors Juhani Välimäki	

This report introduces ONT Thesis tracking system and why it is needed for HAAGA-HELIA University of Applied Sciences as a new information system. The report also explains the functionality of the system from user's point of view. With the help of ONT- Thesis tracking system students, instructors and coordinators can easily and systematically monitor and manage the entire thesis creation process, and communicate with each other through it.

The report describes the technology that has been used in developing ONT Thesis tracking system and explains how we ended up using the Microsoft ASP.NET Web Forms model for developing a web application and why the SQL Server 2005 has been selected as the DBMS.

The report introduces various solutions that have been used to implement the functionalities and data structures to achieve certain requirements. It explains what kind of architecture has been used in the design graphical user interface, application logic, database connection functions and other technical characteristics that have been made in order to provide solutions for the customer (HAAGA-HELIA) requirements.

Finally, the report analyzes the success of the project and the results and sets out proposals for further development of ONT Thesis tracking system. In addition, report explains how the project has affected on author's professional growth.

Key words
ASP.NET, Web Forms, SQL Server, OOP, Information system, web application

Sisällys

Termit ja käsitteet

ASP.NET Web Forms

Microsoftin julkaisema web-sovellustekniikka, jonka ansiosta on mahdollistaa suorituskykyisen, turvallisen ja skaalautuvan sovelluksen kehittämisen web-ympäristöön.

ASP.NET MVC

ASP.NET tekniikkaan perustuva ohjelmistokehys, joka noudattaa MVC (Model-View-Controller) arkkitehtuuria

Bugi

Ohjelmavirhe

Debugata, debuggaus

Etsiä virheitä sovelluksesta sovelluskehittimen avulla

Dialogi (tässä asiayhteydessä)

Valintaikkuna

DLL (Dynamic Link Library)

Microsoftin arkkitehtuurin mukainen binääritiedosto, joka yleensä on jaettu mm. eri sovellusten kesken sovelluskoodin uudelleenkäyttöä ajatellen.

Iteroida, iterointi

Algoritmin tai prosessin tai sen osan toistuva suoritus

Julkinen ominaisuus (public property)

Olio-ohjelmointikielissä käytetty termi, joka tarkoittaa olion sisäistä tilaa käsittelevää olion jäsentä

Klassinen ASP

ASP.NET:ä edeltävä tekniikka, jossa käännettyjen binääritiedostojen sijaan käytetään komentotulkkia, joka tulkitsee sovelluskoodin ja ajaa sen web-palvelimen sisällä. Klassinen ASP ei myöskään noudata OOP-mallia.

Klusteri, klusterikone

Tarkoittaa tietokonetta, joka ohjaa tiedonkäsittelykuorman toisiin tietokoneisiin, jotka ovat siihen yhteydessä

Konstruktori

Olio-ohjelmoinnisissa käytetty metodi, joka suorittaa olion alustustoimet

Kollaatio

Tietokannan merkkijonojen järjestelysääntö, jota voidaan soveltaa eli kielissä. Mm. Ä-kirjain järjestetään eri paikkaan suomenkielisessä kollaatiossa kuin englanninkielisessä.

Kontrolli, palvelinkontrolli (tässä asiayhteydessä)

Sovelluskehityskomponenttia, joka pitää sisällään oman tiedon käsittelyn ja jota voidaan käyttää ASP.NET-sivuissa. Voi tarkoittaa ”Web User Control”-tyyppistä oliota tai muuta sovelluskomponenttia, joka voidaan sijoittaa web-sovelluksen sivulle.

Kulttuuri (tässä asiayhteydessä)

Tieto, joka sisältää tietyn kirjoitetun kielen (esim. suomi ja englanti) ja mm. määritelmät kielen päivämäärien ja numeroiden muotoilusta

Liiketoimintaobjekti

Olio, jonka tehtävänä on käsitellä sovelluksen tieto sen toimintalogiikan mukaisesti

Migraatio (tässä asiayhteydessä)

Datan siirtäminen tietokanta-arkkitehtuurista toiseen

OOP, Olio-ohjelmointi

Olioajatteluun perustuva ohjelmointitekniikka

Parsia, parsiminen (tässä asiayhteydessä)

Tiettyjen tietojen arvojen poiminen toisesta tietokokonaisuudesta

Public accessor

Web-kontrolleissa käytetty julkisesti näkyvä ominaisuus, johon web-sivu voi viitata. Public Accessor:n arvon muuttaminen muuttaa web-kontrollin sisäistä tilaa.

Renderöidä

Piirtää näytölle tai näkymälle kontrolleja

Skripti

Ohjelmakoodi joka suoritetaan sellaisenaan ilman erityistä tapahtumaketjua

Skaalautuminen, skaalautuvuus

Tarkoitetaan ohjelmistokehityksessä ohjelmiston tai järjestelmän mahdollisuutta laajentua rakenteeltaan, toiminnallisuuksiltaan ja ominaisuuksiltaan

Spaghettikoodi

Sovelluskoodi, joka sisältää sekaisin palvelimella ajettuja käskyjä ja HTML–merkkikieltä

Tieto-olio

Olio-ohjelmoinnissa tarkoittaa oliota, joka sisältää tietyn tietokokonaisuuden (mm. tietokanta taulun rivin) yksittäisen ilmentymän

Transaktioloki

Tiedosto, johon tallennetaan tietokannan transaktiotiedot ja vähitellen tietokannanhallintajärjestelmä siirtää tiedot oikeaan tietokantaan

Validointi, validaattori

Validoinnilla tarkoitetaan käyttäjän syötteiden tarkistamista sovelluksen käsittelysääntöjen mukaisesti. Validaattori on web-komponentti, jonka tehtävänä on tehdä tarvittavat tarkistukset käyttäjän syötteille.

Web-kontrolli

Erillinen sovelluskehittäjän määrittelemä kokonaisuus, joka sijoitetaan dynaamiselle web-sivulle, kuten muutkin ASP.NET:n web-komponentit

1 Johdanto

Valitsin opinnäytetyöksi hankkeen, jota oli jo aiottu toteuttaa, mutta siitä ei vielä opinnäytetyön aloitukseen mennessä ollut tehty yhtään kelpaavaa tai toimivaa toteutusta. Kyseessä on HAAGA-HELIA ammattikorkeakoulun opinnäytetöiden ohjausjärjestelmä eli lyhyesti ONT–seurantajärjestelmä (myöhemmin vain ONT). ONTia oli jo alettu miettimään vuoden 2009 syksyllä, jolloin ryhmä HAAGA-HELIAn tietojenkäsittelyn koulutusohjelman opiskelijoita laati tietojärjestelmälle määrittelydokumentaation. Kyseinen ryhmä ei ehtinyt aloittamaan varsinaista tietojärjestelmän suunnittelu- tai kehitystyötä. Vaikka määrittelyissä viitattiinkin suunniteltuun tekniikan käyttöön, yritysmaailmassa ei ole tapana päättää toiminnallisessa määrittelyssä varsinaista tekniikkaa, jolla järjestelmä toteutetaan. Määrittelijähän monesti ei ole edes tekniseltä alalta, joten hänen kannaltaan ei ole juurikaan merkitystä, millä tekniikalla järjestelmä toteutetaan, kunhan se palvelee määriteltyjä toiminnallisuuksia. Juuri siitä syystä itselleni jäi lopullinen tekniikan valinta järjestelmän toteuttamiselle.

Koska osallistuin HAAGA-HELIAn opintojen mukaisiin Microsoft.NET kehityskursseihin ja itsekin olen ollut jo 4 vuotta .NET kehittäjä, pitäydyin kuitenkin Microsoft.NET-tekniikassa kuten aiemmin mainittu määrittelijäryhmä. ONTista tehtiin siis ”ASP.NET Web Forms Application” –tyyppinen sovellus, jonka tietokantana käytetään SQL Server 2005:ä. Käytin määrittelyjä ja opinnäytetyön koordinaattorin palautteita pohjana järjestelemän suunnitteluun. Sovellusta kehittäessä pyrin mahdollisimman hyvään koodin luettavuuteen, tinkimättä kuitenkaan sovelluksen laadusta. Huolenani oli myös sovelluksen skaalautuvuus, suorituskyky sekä turvallisuus.

Skaalautuvuudessa on tärkeää, että järjestelmän lähdekoodi on hyvin jäsennettyä ja selkeät kokonaisuudet erottuvat toisistaan. Pohjautuen perinteiseen käyttöliittymä-liiketoiminta-data arkkitehtuuriin sovelsin hieman samantyyppistä arkkitehtuuria, jossa erona liiketoimintakerrokseen on sidottu rinnakkainen tietokerros, jota kutsuin myös nimellä tieto-oliokerros. Data ja tieto-oliokerros poikkeavat siinä, että datakerros toimii puhtaasti web-sovelluksen ja tietokannan välisenä sovittajana ja tieto-oliokerros sisältää web-sovelluksessa käsiteltävän datan. Tästä tulee tarkemmin luvussa Työn tulokset.

Suorituskyvyn kannalta pyrin välttämään mm. liian pitkän tietokantayhteyden ylläpitämiseen. Olen jopa työelämässä nähnyt karmivia toteutuksia web–sovelluksissa, joissa pahimmillaan tietokantayhteyttä pidetään yllä varaten 7 tärkeää tietokannan taulua iteroiden web-sovelluksessa pitkään haettua dataa. Sellainen jos mikä on todellinen suorituskyvyn pullonkaula. Muutoin ottaen huomioon samanaikaisten järjestelmän käyttäjien määrän – joka on melko vähäinen – pyrin pitämään istunnon tietoja web-palvelimen muistissa. Tämä on myös turvallisuuden kannalta tärkeää.

Turvallisuutta ajatellen, panostin tähän osaan tavallista enemmän. Muun muassa arkaluontoisen datan kryptaus, injektioiden torjuminen, tunnistautumisen ja virheiden järjestelmällinen hallinta ovat peruspilareita turvalliseen web–sovellukseen. Myös näitä ratkaisuja olen hyödyntänyt kehittäessäni ONTia.

Olen pyrkinyt saamaan järjestelmän ja siinä keskeisessä roolissa olevan web-sovelluksen sellaiseen tilaan, että sitä on mahdollista jatkokehittää helposti ja että järjestelmän käyttöönotto HAAGA-HELIAn tietoteknisessä ympäristössä olisi mahdollisimman jouhevaa. Tätä varten olen laatinut järjestelmäkuvauksen HAAGA-HELIAn IT-palveluille, joka on tämän opinnäytetyön salainen liite. Tämän opinnäytetyön rajaus olikin kehittää järjestelmä niin pitkälle, että työn tilaaja, eli HAAGA-HELIA ammattikorkeakoulu olisi valmis sen järjestelmätestaukseen, pilotointiin ja käyttöönottoon. Opinnäytetyöstä oli rajattu myös ”Winha Wille” –rajapinnan toteuttaminen.

Tämän projektin hallinnasta ja projektin etenemisestä on enemmän liitteessä 6 (Loppuraportti). Tämän projektin tekemiseen on käytetty HAAGA-HELIAn dokumentointimenetelmiä, käytäntöjä ja ohjeiden soveltamista. Järjestelmän kehityksessä on käytetty huolellista olio-ohjelmointitekniikan soveltamista suunnittelussa, ohjelmoinnissa sekä tuotteen laadunvarmistuksessa.

Tätä opinnäytetyötä luettaessa tulee ottaa huomioon, että näyttökuvat ja muut tiedot järjestelmän toteutuksesta viittaavat ratkaisuihin, jotka on tehty ennen opinnäytetyön

luovutusta syksyllä eli ennen lokakuuta 2011. Mikäli järjestelmään on tehty muutoksia tämän jälkeen, kaikki esitetyt ratkaisut eivät ole välttämättä ajantasaisia.

2 Tausta työlle

2.1 ONT-seurantajärjestelmän tarve

Kuten johdannossa tuli esille, ONT syntyi tarpeesta, jossa haluttiin hallita järjestyksellisesti opinnäytetöiden seurantaa ja helpottaa mm. ohjaajien työtä, jotta sähköpostiliikenteestä ei tulisi täysin hallitsematonta. Varmasti jo viiden opinnäytetyön seuraaminen samanaikaisesti voi aiheuttaa harmaita hiuksia, jos asiat eivät ole järjestyksessä. Vaikka itse ohjaaja ei olisikaan maailman järjestyksellisin ihminen, ONT pitäisi asioita järjestyksessä ja tekisi joitakin asioita automaattisesti, jolla muuten ohjaaja tuhlaisi arvokasta aikaansa ja täten hän pystyisi keskittymään varsinaiseen työhönsä sen sijaan, että hänen työaikansa menisi suureen määrään turhaa ”käsi- ja paperityötä”. Koululla on siis ollut tarve kehittää tällainen järjestelmä, joka ohjaisi ja helpottaisi kaikkien opinnäytetyön valmistusprosessiin osallistuvien henkilöiden työtehtäviä.

Järjestelmässä on monia ominaisuuksia ja palveluja, jotka ovat pääpiirteittäin seuraavat: Järjestelmän peruspalveluja ovat:

- kirjautuminen
- kielen vaihtaminen kesken asioinnin
- tallennettujen tiedostojen ja aihe-ehdotusten kommentointi
- automaattinen sähköpostikuittaus opinnäytetyöprosessin ei vaiheissa

Opiskelijalle tarjotut palvelut ovat:

- aihe-ehdotuksen tallentaminen ja lähettäminen
- tiedostojen tallentaminen, muokkaus ja poisto
- kokousten tallentaminen, muokkaus ja poisto ja niihin liittyvien tiedostojen tallennus
- opiskelijan edeltävyyksien automaattinen tarkistaminen

Ohjaajalle tarjotut palvelut ovat:

- yksittäisen opinnäytetyöprosessin hallinta
- helppo kokousten järjestyksellinen seuranta
- uusien tiedostojen näyttäminen heti kirjautumisen jälkeen
- järjestelmään tallennettujen tiedostojen lataus

Koordinaattorille tarjotut palvelut ovat:

- yksittäisen opinnäytetyöprosessin hallinta
- järjestelmään tallennettujen tiedostojen lataus
- järjestelmäilmoitusten luominen ja julkaiseminen
- koordinaattorin oman edeltävyysmatriisin luominen ja hallinta
- omien koulutusohjelmien ohjaajien hallinta järjestelmässä
- opiskelijoiden edeltävyyksien vahvistaminen tehdyiksi ilman Winha Willeä
- ohjaajan valitseminen opinnäytetyöhön ilman ohjaajan kuittausta

Koordinaattorille ONT tuo selkeitä hyötyjä. Koordinaattorikin voi seurata yksittäisiä oman koulutusohjelmien opinnäytetöitä. Tämän lisäksi hän ottaa vastaan opiskelijoiden aihe-ehdotukset ja on osa aihe-ehdotusten hyväksymistyövuota. Aihe-ehdotusten tilat muuttuvat koordinaattorin ja ohjaajien valintojen perusteella. Kaikissa näissä työvaiheissa koordinaattorin tai ohjaajan ei tarvitse lähettää yhtäkään sähköpostia (lisää aihe-ehdotuksen tiloista ks. Liite 4. Tilakaavio). Järjestelmä hoitaa automaattisten viestien lähetyksen.

Yhdellä koulutusohjelmalla on normaalisti vain yksi koordinaattori. Siksi koordinaattori tarvitsee oman edeltävyystarkistusmatriisin, jonka avulla hän voi hallinnoida edeltävyyksiä ja edeltävyyksien toteutuksia eri opintojaksojen perusteella. Sama edeltävyys voi olla eri koulutusohjelmissa eri opintojaksolla, mutta täyttävät saman edellytyksen. Koordinaattori voi ONTin avulla hallinnoida tätä matriisia. Matriisia käytetään siinä vaiheessa kun opinnäytetyötä aloittavan opiskelijan edeltävyyksiä tarkistetaan. Nämä edeltävyydet tarkistetaan ennen kuin hän aloittaa aihe-ehdotuksen kirjoittamisen. Edellä on esimerkki keinotekoisesta (eli ei mitenkään sitoudu oikeisiin

HAAGA-HELIAn toteutuksiin, edeltävyyksiin tai opintojaksoihin) edeltävyysmatriisista, jota koordinaattori tarvitsee edeltävyyksien hallinnointiin.

Koulutusohjelma Pakolliset opinnot	Tietojenkäsittelyjen koulutusohjelma ilta (ennen vuotta 2009 aloittaneet)	Tietojenkäsittelyjen koulutusohjelma päivä	Tietojenkäsittelyjen koulutusohjelma päivä (ennen vuotta 2009 aloittaneet)
Tietotekniikan perusteet	ICT1001	ICT2001	ICT3001
Ohjelmoinnin perusteet	ICT1002	ICT2002	ICT3002
Tietokannan rakenteet	ICT1003	ICT2003	ICT3003
Tietoverkot	ICT1004	ICT2004	ICT3004
IT- juridiikka	ICT1005	ICT2005	ICT3005
Yritystieto	ICT1006	ICT2006	ICT3006
Ohjelmistoarkkitehtuurit	ICT1007	ICT2007	ICT3007

Kuva 1. Koordinaattorin edeltävyysmatriisi

Koordinaattorilla on myös tarve hallinnoida oman koulutusyksikkönsä ohjaajia. Hän voi valita järjestelmän avulla ketkä suorittavat opinnäytetöiden ohjaustyötä. Tämän lisäksi hän voi luoda järjestelmään ilmoituksia, jotka näkyvät opiskelijoille heti sovelluksen etusivulla, kun he ovat kirjautuneet sisään.

Opiskelijalla toisaalta on tarve enimmäkseen syöttää tietoa järjestelemään. Hän lähettää arvioitavat työt, aihe-ehdotukset ja tulevat kokoukset järjestelmään. Vaikka opiskelijalla on toki samat tiedostot myös mm. omalla työkoneellaan, hän voi kätevästi lähettää järjestelmään ainoastaan sellaiset työt, jotka sillä hetkellä ovat mm. arvioitavina tai kommentoivina. Kommentointiominaisuus onkin olennainen osa järjestelmää. Näin palaute voidaan kohdistaa juuri tiettyyn opinnäytetyön osaan ja se pysyy muistissa eikä kenenkään tarvitse käydä kahlaamassa omia sähköpostiviestejään läpi, jotta löytäisi tietyn tiedon/palautteen.

2.2 Valittu sovellustekniikka - ASP.NET Web Forms

Järjestelmää varten tarvittiin sellainen tekniikka, joka mahdollistaisi järjestelmän skaalautumisen sekä asiakasohjelmariippumattomuuden. ”ASP.NET Web Forms” -tyyppinen sovellus täyttäisi tämän tarpeen. Mieleenkään ei tullut käyttää vanhaa ASP-tekniikkaa, jossa käytetään käännettyjen binääritiedostojen sijaan tulkkia, joka on muutenkin hidas kun useampi säie ajaa samaa sovelluskoodia. ASP.NET Web

Form:ssa on se etu, että lähdekoodit käännetään binääritiedostoiksi, ne ajetaan ensimmäisen kerran, jonka jälkeen niiden toiminnallisuudet ovat välimuistissa ja suorituskyky on huomattavasti parempi. Myös turvallisuuden kannalta ASP.NET Web Forms on vaihtoehtona hyvä. Web Forms myös sopii tämäntyyppiselle ja -kokoiselle sovellukselle. Siinä ei tarvita hienoa grafiikkaa tai vuorenvarmaa sovellusta, jossa on varajärjestelmä tai klusterikone käyttökatkojen varalta. Itse olen käyttänyt ASP NET:ä jo vuosien ajan ja olen jo melkoisen harjaantunut sillä kehittäessäni web-sovelluksia.

ASP.NET Web Forms on osa ASP.NET Web Application Framework:a. Se on ASP MVC mallia vanhempi kehitysmalli, jota on käytetty jo ensimmäisistä .NET Framework julkaisuista lähtien. Web Forms:ssa sivut ovat ikään kuin web-lomakkeita, joiden kautta käyttäjät pystyvät syöttämään ja vastaanottamaan dataa. ”Web Forms” – malli lähentelee Windows-lomakekehitysmallia, mutta toimii siis web-maailmassa. Sivut ovat osittain HTML:ää ja osittain sovelluskoodia, jota palvelin ohjaa.
Nämä sivut ovat käännettyä ja käyttäjän pyytäessä sivua vastaus suoritetaan palvelimella ja luodaan HTML:ää, joka palautetaan että käyttäjän selaimeen.

Visual Studion avulla on luodaan ASP.NET Web sivuja (lomakkeita). Kehitystyötä suoritetaan hieman samaan tapaan kuin Windows-ohjelmoinnissa, jossa komponentteja voi sijoittaa lomakkeille ”Drag-and-Drop”-tyylillä tai suoraan merkkikieleen. Komponenttien ominaisuuksia, metodeja ja tapahtumia voi helposti käsitellä ja sen avulla voi määrittää sovelluslogiikkaa ja mm. graafista ulkoasua. Ohjelmointikielet, joita normaalisti käytetään, ovat C# ja Visual Studio.NET

ASP.NET Web Forms tarjoaa mm.:

- HTML-merkkikielen ja sovelluskoodin erottelun
- laajat ja rikkaat web-komponentit web-lomakkeiden luomiseen jopa vaativia tehtäviä varten
- ”Databind”-ominaisuudet ja erilaiset työkalut niiden tekemiseen
- Ajaxin käyttö ilman aiempaa Javascript osaamista

(Microsoft 2011)

ASP.NET helpottaa huomattavasti itse sovelluksen kehitystä. Tarkoituksena tässä sovelluksen kehityksessä olikin, että se olisi helposti muokattavissa ja siksi pyrin pitämään sovelluksen eri kerrokset erillään ja mahdollisimman riippumattomina toisistaan. Ei ole enää vanhaa spaghettikoodia, jossa palvelimella ajettu koodi oli sekaisin puhtaan HTML merkkikielen seassa. Näin koodin luettavuus ei kärsi ja on helpompi erottaa käyttöliittymän näyttäminen sen ohjaamisesta, kuten ”ASP.NET”-MVC mallissa on pyritty tekemään. Vanhempaa ASP-tekniikkaa vielä käytetään useiden yritysten verkkosovellusratkaisuissa, vaikka näitä on pyritty uudistamaan uudemmalla ASP.NET-tekniikalla. Ongelmana usein on myös, että ne, jotka ovat tottuneet kehittämään vanhalla ASP:lla, eivät välttämättä tiedä paljoakaan olio-ohjelmoinnista ja näin heidän tekemät uudet ratkaisut eivät käytä uutta tekniikkaa täysin hyväkseen. ASP.NET antaa hyvän lähestymistavan niille kehittäjille, jotka ovat tottuneet tekemään tapahtumakeskeistä ohjelmakoodia, kuten Visual Basic kehittäjät. Vaikka sovellus onkin koodattu C# kielellä, sama tapahtumakeskeisyys löytyy myös ”Web Forms”-tekniikasta. Siitähän idea onkin lähtenyt, että weblomakkeet olisivat mahdollisimman samanlaisia Windows-lomakeohjelmoinnin kanssa.

Mitkä ovat siis ASP.NET-tekniikan edut verraten klassiseen ASP-tekniikkaan?
ASP.NET Web Forms lähestymistavassa lähdetään siitä, että mahdollisimman vähän kirjoitetaan varsinaista HTML merkkikieltä tai JavaScript–komentosarjoja. Tämä mahdollistaa hyvän selain riippumattomuuden. Joskus laitteet, jotka lukevat WML merkkikieltä, voivat käyttää ASP.NET:llä tehtyjä sovelluksia kun haetut resurssit kirjoitetaan eri merkkikielellä kun web-palvelin palauttaa vastauksen. Sivuille asetetaan mahdollisimman paljon palvelinkontrolleja ja komponentteja, jolloin IIS:n ja ASP.NET Framework:n tehtävänä on luoda tarvittava merkkikieliteksti sekä JavaScript komentosarjat. Kuten aiemmin mainittiin, klassinen ASP käyttää tulkkia sen sijaan, että koodi olisi käännettyä ja täten ASP.NET on huomattavasti suorituskykyisempi. Visual Studion kannalta ASP.NET-verkkosovellusta voidaan debugata kun taas klassisessa ASP:ssa tämä ei ole mahdollista. Huomattavaa on, että klassisen ASP:n sijaan ASP.NET-verkkosovellusta voidaan ajaa riippumatta IIS-palvelun asetuksista. Lisäksi ASP.NET sovellusta voidaan ajaa myös web-palvelimella, joka ei ole riippuvainen Microsoft-teknologiasta. Sitä voidaan ajaa esimerkiksi myös ”Apache”-web-

palvelimella. Nämä ja monet muut ominaisuudet tekevät ASP.NET-verkkosovelluksista ylivertaisia klassisiin ASP-verkkosovelluksiin nähden.
(Bean Software)

Huomattavaa ASP.NET kehityksessä on myös eräs asia, joka monelta kehittäjältä jää huomioimatta. ASP.NET sivuilla on tietty elinkaari ja eri elinkaarien osassa tulisi tehdä eri asioita kuin toisissa. Esimerkiksi *OnInit*, *OnLoad* ja *OnPreRender* ovat eri sivun elinkaaren tapahtumia ja niissä ei tulisi suorittaa mitä tahansa koodia. Esimerkiksi olen nähnyt ASP.NET toteutuksia, joissa lähes kaikki sivun toiminnallisuus on sullottu ”Page_Load” -tapahtumaan. Tämä varmaan juontaa juurensa ajalta, kun klassisen ASP:n kehittäjät eivät tunteneet uudempaa ASP.NET-tekniikkaa kovin hyvin ja päättivät tehdä koko vanhan toiminnallisuuden yhden tapahtuman sisään. Tämä haittaa sovelluksen hallittavuutta, koodin luettavuutta ja virhemarginaali kasvaa. Eri elinkaaren tapahtumilla on omat rajoituksensa ja ominaisuutensa. Esimerkiksi *OnInit* on elinkaaren ensimmäisiä tapahtumia, jolloin ASP.NET ei voi esimerkiksi vielä tehdä päätelmiä käyttäjän palvelimelle lähetetyistä syötteistä. ”OnInit”-tapahtumassa voidaan luoda dynaamisesti oliota ENNEN kuin sivu renderöidään asiakkaan näytölle. *OnPreRender* sopii hyvin datan sitomiseen näytettävälle sivulle, koska kaikki mahdollinen data on jo saatu tässä myöhäisessä elinkaaren vaiheessa kun alkaa olla aika piirtää näytölle dataa. (Microsoft 2011)

Java-tekniikkaa en harkinnut käyttää projektissa, koska en tunne Javaa kovin hyvin ja muutenkin Java-sovellusten ylläpito on hinnoiteltu kalliiksi yritysmaailmassa. Mikäli tätä sovellusta aiotaan jatkokehittää tulevaisuudessa, tämä seikka on syytä ottaa huomioon.

Loppujen lopuksi ASP.NET oli hyvä vaihtoehto sovelluksen toteutukselle, vaikkei se olisi ollut ainoa. Ottaen huomioon asiakkaan tarpeen sekä kykyni ja taitoni, päädyin tähän vaihtoehtoon.

2.3 Valittu tietokantaratkaisu - SQL Server 2005 ja ADO.NET

Monesti Oracle–kehittäjät ovat pitäneet SQL Serveriä vähemmän luotettavana ja vakaana tietokantahallintajärjestelmänä Oracleen verrattuna. Vaikka kyseinen lausunto olisikin totta, tähän kyseeseen tarpeeseen tietokantapalvelimena palvelisi hyvin SQL Server. SQL Serverissä on toimiva transaktiohallinta, se kestää miljoonien tietueiden käsittelyn ja on muutenkin vakaa Microsoft ympäristössä, joten asiakkaan tarpeeseen se on sopiva. Kaupallisessa käytössä SQL Server on kallis, joten vaihtoehtoja tietenkin SQL Serveriin olisi ollut. Järjestelmää ei ole mitenkään sidottu SQL Server 2005:een, vaikka aiemmat tämän projektin toteutusyritykset olivatkin sellaisia. Tietokantasovittajakerros voidaan tehdä täysin uudelleen uusilla sovittajilla. On myös olemassa paljon erilaisia työkaluja joiden avulla voidaan tehdä migraatio SQL Serverin ja muiden tietokantahallintajärjestelmien kanssa. SQL Server sopii hyvin monenlaisiin web-sovelluksiin ja se onkin lähes aina käytössä erilaisissa ASP.NET web-sovelluksissa.

Päällimmäinen syy SQL Serverin käyttämiseen oli lähinnä sen ydintuntemus omasta kokemuksestani sekä sen hallintatyökalun SQL Server Management Studion käytön osaaminen. SQL Server tarjoaa Microsoft-maailmassa joihinkin muihin tietokantahallintajärjestelmiin verrattuna jotain enemmän. Tiedon hakuun ja käsittelyyn sopii mainiosti ”Stored Procedure” -komentosarjat, jotka integroituva Microsoft ADO.NET-tekniikan kanssa ja jotka toimivat hyvin SQL injektioita vastaan, sillä ne ottavat mm. merkkijonot sellaisinaan vastaan eikä mitään heittomerkkitemppuilua voida tehdä, sillä SQL Server ei rakenna ”Stored Procedure” -komentosarjojen parametreista kokonaista SQL lausetta, vaan parametrit käsitellään erillään SQL–komennoista. Tämän lisäksi ”Stored Procedure” -komentosarjat käännetään ja ne pysyvät tietokantapalvelimen muistissa käännöksen jälkeen maksimaalisen suorituskyvyn saamiseksi. Tietokantaoperaatiot suoritetaan yhtenäisesti, vakaasti ja turvallisesti. (Pinal Dave 2007)

Tietenkin ”Stored Procedure” -komentosarjat tulee olla järkevästi rakennettu, jottei niistä tulisi suorituskyvyn pullonkauloja. Mm. turhat alikyselyt hidastavat SQL Serverin kyselyn tekemistä huomattavasti. Lisäksi komentosarjoissa ei tulisi tehdä päätelmiä sovelluksen tiedonkäsittelystä, koska lähes kaikki päättelylogiikkaa tehdään sovelluksen

liiketoimintakerroksessa. Siksi olen jättänyt ne SQL–komennoista pois ja muutenkin olen pyrkinyt tekemään tietokantakomennoista mahdollisimman yksinkertaisia ajatellen myös luettavuutta.

Tietokantayhteys ASP.NET-maailmaan löytyy samasta tekniikan altaasta, eli Microsoft .NET-tekniikasta. Microsoft.NET:ssä on laaja ja tehokas tuki SQL Serverin käyttöön ADO.NET Framework:ssa. Pidän loogisena käyttää saman toimittajan tekniikkaa niiden yhteensopivuuden vuoksi. ADO.NET takaa yhtenäisen pääsyn tietolähteisiin, kuten SQL Serveriin ja XML:ään, ja ”OLE DB”- ja ODBC-tietolähteisiin.

ADO.NET erottaa tiedon haun tietojen käsittelystä erillisillä komponenteilla joita voidaan käyttää erikseen tai yhdessä. (vrt. SqlCommand ja SqlDataReader), ADO.NET käsittää yhteyden tietokantaan, komentojen suorittamisen ja datan haun ja käsittelyn. Hakutulokset joko käsitellään suoraan tai data sijoitetaan DataSet-objektiin, josta data on voitu hakea useista lähteistä, kuten XML:stä. ADO.NET luokat löytyvät System.Data.dll binääritiedostosta ja ovat integroitu XML:n tiedonkäsittelyluokkiin, jotka löytyvät ”System.Xml.dll”-binääritiedostosta. ADO.NET sisältää myös ominaisuuksia, joissa tarvitaan toimintoja natiiviin ”Component Object Model” – komponenttimallin (COM) käsittelyyn, vaikka tätä ei suositella. On parempi käyttää ADO.NET:n omia komponenttimalleja. (Microsoft)

3 Työn tulokset

3.1 Järjestelmän arkkitehtuuri

Kuten jo johdannossa mainitsin, ONT on kehitetty käyttämällä ”Microsoft.NET Framework 3.5” –alustaa. Työvälineenä käytin web-sovelluksen kehityksessä Visual Studio 2008:a ja tietokannan kehityksessä SQL Server Management Studio:a. Web–sovellus on tehty ”ASP.NET Web Forms Application” –tyyppiseksi sovellukseksi. Näin taustalla tapahtuva tiedonkäsittely tehdään binääritiedostoissa, mikä on turvallisempaa ja suorituskyky on parempaa, kun sovelluksessa on valmiiksi käännetyt binääritiedostot eli DLL:t.

Koska web-sovelluksessa olisi siis arkkitehtuurin mukaisesti ainakin 4 kerrosta: käyttöliittymä-, liiketoiminta-, tieto-olio- ja datakerros, loin kullekin niistä oman Visual Studio projektin ja käytin kunkin niissä ”*<yritys>.<sovellus>.<kerros>*” -mukaista nimiavaruuden nimeämiskäytäntöä. Tietenkin pohjana on ”Visual Studion Solution” -projektitiedosto, jonka sisään voi lisätä useita projekteja. Siksi neljä projektia nimesin seuraavasti:

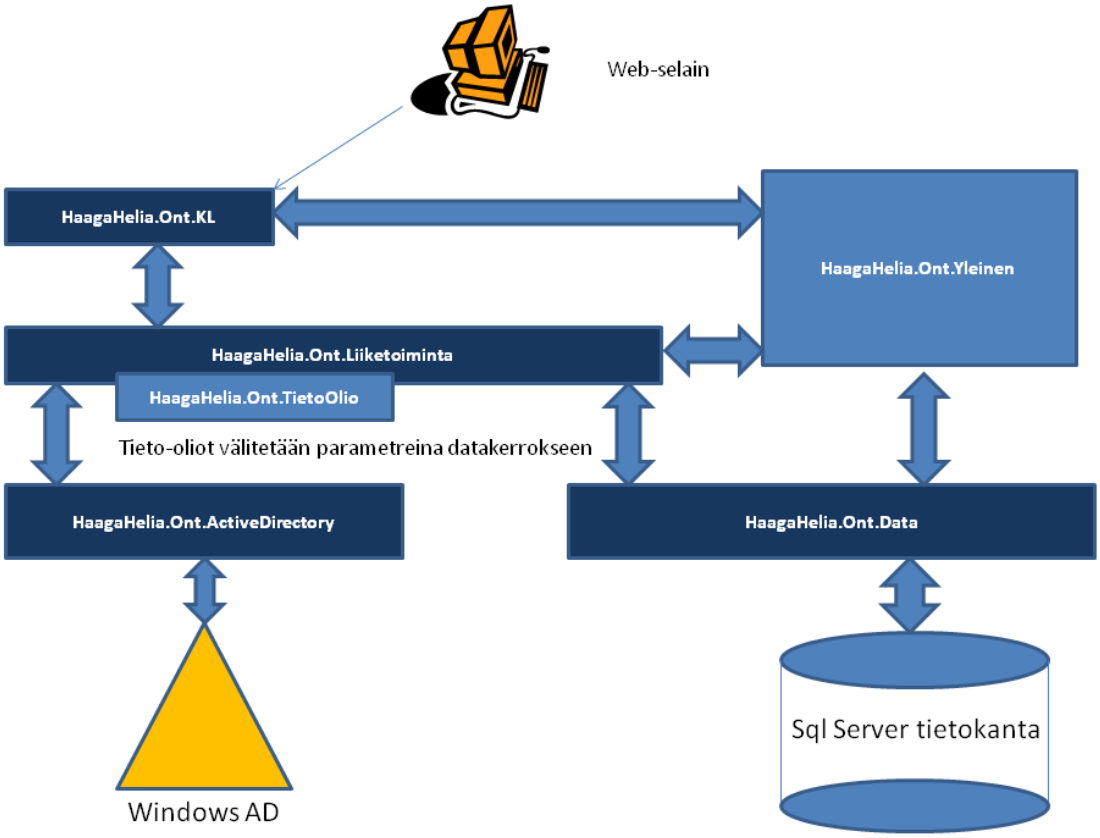
- HaagaHelia.Ont.KL (käyttöliittymä, ASP.NET Web Application -projekti)
- HaagaHelia.Ont.Liiketoiminta (liiketoiminta, Class Library)
- HaagaHelia.Ont.TietoOlio (tieto-oliokerros, Class Library)
- HaagaHelia.Ont.Data (data, Class Library)

Tämän lisäksi pidin tarpeellisena pitää Active Directory yhteyttä erillään muista kerroksista, joten loin sitä varten projektin HaagaHelia.Ont.ActiveDirectory. Tämä sisältää ainoastaan yhden luokan, *AdAvustaja*. Tämän lisäksi tarvitsin vielä kaksi projektia: yhden ”Class Library” -projektin ja yhden ”Windows Application” -projektin. HaagaHelia.Ont.Yhteinen on projekti, jota käyttävät kaikki muut kerrokset eli sillä ei ole viiteriippuvuutta mihinkään toiseen projektiin. Näin vältetään esimerkiksi ”Circular reference” –tyyppinen ongelma, eli koodi ei käänny, koska DLL:t tarvitsevat kaikki toisiaan ja näin syntyy riippuvuussilmukka. Tämä projekti sisältää kaikki

apuluokat, jotka auttavat mm. web-sovelluksen konfigurointitiedoston lukemisessa, lokalisoinnissa ja lokiin kirjoittamisessa.

Pieni Windows sovellus (HaagaHelia.Ont.KryptausApuohjelma) auttaa kryptaamaan tekstiä ja purkamaan kryptausta salausavaimen mukaan.

Visual Studion Solution sisältämät projektit, arkkitehtuurin kerrokset ja niiden väliset yhteydet voidaan kuvata seuraavalla kaaviolla.



Kuva 2. Kaavio Visual Studio Solution -projektien välisistä suhteista

Tietokanta on luotu SQL Server 2005 –palvelimelle ja siinä on eri taulujen väliset viite-eheystarkistukset ja primääriavaimet kohdallaan. Tietokannan yhteen tauluun tallennetaan myös binääridataa, jonka kokoa rajoitetaan web-sovelluksesta käsin. Kaikki tietokantaan tehdyt kyselyt ja komennot tehdään SQL Serverin ”Stored Procedure” –komentosarjojen kautta. Web–sovelluksesta kutsutaan ADO.NET-rajapinnan kautta SQL Serverin ”Stored Procedure” -komentosarjoja parametreineen ja ADO.NET palauttaa (mikäli kyseessä on kysely) haetun datan takaisin web-sovellukseen, jossa data

sovitetaan web-sovelluksen tieto-olioihin. Lopulta tietokantayhteys katkaistaan heti kun yhteys ei ole enää tarpeen.

Kutakin projektia varten on luoto ”Strong key”-salausavain, jotta DLL tiedostojen väärennös ei olisi mahdollista. Näin varmistetaan, että kaikki järjestelmään tehtävät muutokset tehdään aina yhden lähdekoodiversion sisällä eikä DLL:iä voi koodata erikseen.

3.2 Käyttöliittymä

Käyttöliittymä on toteutettu tekemällä kaikki sivut ASP.NET mallia käyttäen. Kaikki yksittäiset sivut ovat ”.aspx”–päätteisiä tiedostoja. Suurin osa palvelinpäässä suoritettavasta koodista on erillisessä tiedostossa ”Codebehind”-attribuuttiin määritetyssä tiedostossa. Näin on helpompi lukea html-merkkikieltä sen sijaan, että seassa olisi palvelimella ajettavaa koodia. Mahdollisimman vähän on käytetty kuitenkin varsinaista html–merkkikieltä ja on annettu web-palvelimen ja .NET Framework:n hoitaa sivujen muunnos puhtaaksi html:ksi. Esimerkiksi syötekentissä käytetään ASP.NET web-komponentteja, sen sijaan että käytettäisiin ”input”-tageja. Näitä ovat mm. asp:TextBox ja asp:Button.

3.2.1 Päänäkymät

Käyttöliittymä on jaettu kolmeen päänäkymään, joita kutakin varten on luotu oma alikansio. Päänäkymät ovat jaettu järjestelmän käyttäjäroolien mukaan, jotka ovat opiskelija, ohjaaja ja koordinaattori. Neljäs rooli, hallinto tai johto, joka alun perin oli myös määrittelyissä, on rajattu tästä opinnäytetyöstä pois.

Kun käyttäjä on kirjautunut järjestelmään ja järjestelmä on tunnistanut käyttäjän, suorittaa se auktorisoinnin eli ONTin tapauksessa se ohjaa käyttäjän roolin perusteella oikeaan näkymään. Seuraavassa on esimerkki metodista, joka suorittaa päättelyn ja palauttaa kutsujalle oikean sivun suhteellisen osoitteen.

```
private string PaatteleAloitussivu(Rooli rooli)
{
    switch (rooli)
    {
        case Rooli.Hallinto:
            return string.Empty;
        case Rooli.Koordinaattori:
            return NayttojenNimet.KoordinaattoriEtusivu;
        case Rooli.Ohjaaja:
            return NayttojenNimet.OhjaajaEtusivu;
        case Rooli.Opiskelija:
            return NayttojenNimet.OpiskelijaEtusivu;
        default:
            return null;
    }
}
```

Kuva 3. Päättely näkymästä roolin perusteella

Käyttöliittymässä on näiden kolmen päänäkymän lisäksi yleisiä sivuja, kuten esimerkiksi sivu istunnon päättymiselle, yleinen virhesivu, tiedoston lataaminen (joka käytännössä palauttaa näytölle HTTP:n välityksellä tiedoston binäärivirran).
Käyttöliittymässä on myös CSS-tiedosto (Cascading Style Sheet) tyylejä varten ja yksi kuva (organisaation logo).

3.2.2 Master page

Jokaisella päänäkymällä on oma Master page eli Master-sivu. Master-sivussa on määritetty elementit, joiden tulee näkyä jokaisella sivulla. Näitä ovat mm. kirjautunut käyttäjä, viimeisin kirjautuminen, kielen valintalinkit ja linkki uloskirjautumiseen ja logo. Tämän lisäksi on erikseen sivulla myös kullekin näkymälle oma päävalikko. Päävalikosta pääsee valitsemaan käyttäjän tarvitseman osion ja näin avautuu osiokohtainen alavalikko näytön vasempaan reunaan. Tosin etusivulla alavalikko vastaa myös päävalikon linkkejä.

Kuva 4. Päävalikko ja alavalikko

3.2.3 Web User Control -elementit

Päävalikko, alavalikko ja muut erilliset elementit ovat käytännössä toteutettu ”Web User Control” –tyyppisinä käyttäjän toteuttamina palvelinkontrolleina (.ascx tiedostot), joita myös kutsutaan web-kontrolleiksi. Käytännössä sivuille, kuten myös Master–sivulle, palvelinkontrollit rekisteröidään ja sijoitetaan ne sivuille tiettyyn kohtaan. Seuraavassa esimerkki rekisteröinnistä:

```
<%@ Register TagPrefix="uc" TagName="HeaderYleinenOpp" Src="HeaderYleinenOpp.ascx" %>
```

Ja seuraavassa on esimerkki miten vastaava Web User Control sijoitetaan sivulle:

```
<uc:HeaderYleinenOpp ID="ucHeader" runat="server"></uc:HeaderYleinenOpp>
```

Master-sivuun palvelinkontrollien lisäksi täytyy lisätä paikka, mihin varsinainen haettu sivu tulee. Master-sivu toimii ainoastaan kehyksenä haetulle sivulle, eikä siihen tehdä minkäänlaista erityistä toiminallisuutta vaan sivun toiminnallisuus on ”aspx”-sivussa ja sen sisältämissä web-kontrolleissa. Sivun paikka Master-sivussa ilmaistaan seuraavasti:

```
<asp:ContentPlaceHolder ID="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
```

Jotta sivu löytäisi oikean paikkansa Master-sivussa, sivun sisällön tulee olla ”asp:Content” -tagin sisällä ja sen tulee viitata oikeaan ”ContentPlaceHolder”-elementtiin.

Seuraavassa esimerkki ”LisaaTiedosto.aspx” –sivusta, joka on sijoitettu Master-sivun sisään.

```
<asp:Content ID="Lisaatiedosto" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
…
</asp:Content>
```

Kuten aiemmin tuli esille, sivuille on upotettu alavalikon palvelinkontrolli. Näin voidaan sivukohtaisesti valita mitä alavalikkoa voidaan käyttää. Alavalikon lisäksi sivuilla on web-kontrolli kontekstiohjetta varten. Lisää kontekstiohjeesta aliluvussa Lokalisointi.

ONTissa on kahdentyyppisiä sivuja: listaussivuja ja lomakesivuja. Lomakesivuilla web-kontrollia käytetään myös syöttökontrollien esittämiseen. Syy miksi joillakin lomakesivuilla näin on tehty, johtuu yksinkertaisesti siitä, että samaa web-kontrollia saattaa käyttää kaksi eri sivua. Esimerkkinä ovat sivut *LisaaOpinnaytetyonKokous.aspx* ja *MuokkaaKokousta.aspx*. Molemmat näistä sivuista käyttävät samoja syötekontrolleja ja näin vältytään turhalta syötekontrollien toistamiselta, jotka muutenkin toimisivat samalla tavalla molemmilla sivuilla. Tämäntyyppisissä web-kontrolleissa sivulla ei ole oletuksena pääsyä web-kontrollin syötekontrolleihin, koska syötekontrollien näkyvyys on oletuksena *protected*, eli ne näkyvät ainoastaan perintäketjussa oleville luokille. Siksi web-kontrollien sisällön hallintaa varten on luotava *public accessor*, eli julkinen

ominaisuus, jonka kautta sivu voi syöttää tietoa web-kontrollille tai lukea tietoa siitä. Näin web-kontrollin vastuulle jää tiedon käsittely ja sen oikea näyttäminen selaimessa.

Hyvänä esimerkkinä tästä on *KokousKentat.ascx* web-kontrolli. Siinä on erikseen kaksi (tai oikeastaan kolme) syötekontrollia, joista toinen on web-kalenteri tai toinen jakautuu kahden pudotusvalikon kanssa kellonajaksi. Web-kontrolli sisältää public accessor:n Aika, joka ottaa vastaan (set-lohko) *DateTime*–luokan ilmentymän. Set-lohko *public accessor*:ssa hoitaa sen, että *DateTime*:n päivämääräosa asetetaan kalenterikontrolliin ja kellonaikaosa asetetaan kauniisti pudotusvalikkoihin. Samaan tapaan get-lohko parsii kalenterista päivämäärän ja kellonajan pudotusvalikoista. Seuraavassa on esimerkki siitä, miten tämä tehdään.

```
public DateTime Aika
{
    get
    {
        DateTime temp = new DateTime(calAika.SelectedDate.Year,
                                     calAika.SelectedDate.Month,
                                     calAika.SelectedDate.Day,
                                     int.Parse(ddlKloTunti.SelectedValue),
                                     int.Parse(ddlKloMinuutti.SelectedValue),0);
        return temp;
    }
    set
    {
        //Kalenteri ei näytä valittuja päiviä, jos kalenterissa on kellonaika. Tämä on
Microsoftin bugi, jota ei ole korjattu vuosiin.
        calAika.SelectedDate = PaivamaaraAvustaja.KarsiKellonAika(value);
        calAika.VisibleDate = PaivamaaraAvustaja.KarsiKellonAika(value);
        ListItem item = ddlKloTunti.Items.FindByValue(value.Hour.ToString());
        item.Selected = true;
        ListItem item2 = ddlKloMinuutti.Items.FindByValue(value.Minute.ToString());
        item2.Selected = true;

    }
}
```

Sivuhuomiona tähän kalenterikontrolliin - jota käytetään vain yhdessä paikassa ONTissa - on se, että jos kalenterikontrollille syöttää päivämäärän, jossa on myös kellonaika, se ei näytä valittua päivämäärää kontrollissa. Nopean Google-haun jälkeen vaikuttaisi siltä, että kyseessä on Microsoftin bugi tuossa kontrollissa.

3.2.4 Räätälöidyt syötekontrollit ja lokalisointi

ONTia varten on tehty muutama räätälöity syötekontrolli. Tämä on tehty lähinnä lokalisointia ajatellen. Kullekin syötekontrollille on annettu ”Lokalisointiavain” -ominaisuus, jonka arvo löytyy kielikohtaisissa resurssitiedostoissa kuten esimerkiksi *KieliAvaimet_fi-FI.resx* ja *OhjetekstiAvaimet_fi-FI*. Lokalisointiavain annetaan

syötekontrollille jo syötekontrollin ”Design”–tilassa Visual Studiossa. Räätälöidyt syötekontrollit ovat peritty ASP.NETin perussyötekontrolleista, joissa näytetään lähinnä staattista tekstiä esimerkiksi asp:Label syötekontrollissa.

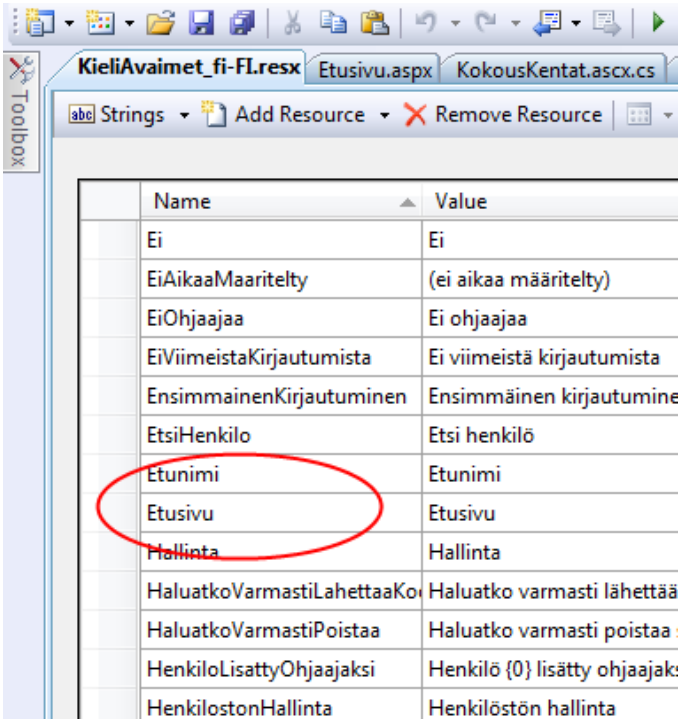
Räätälöidyt syötekontrollit sijaitsevat HaagaHelia.Ont.KL–projektin alikansiossa WebKomponentit (eli nimiavaruudessa HaagaHelia.Ont.KL.WebKomponentit) ja ne rekisteröidään sivuille tai web-kontrolleihin seuraavasti:

```
<%@ Register TagPrefix="CustomWeb" Assembly="HaagaHelia.Ont.KL"
Namespace="HaagaHelia.Ont.KL.WebKomponentit" %>
```

Räätälöity syötekontrolli asetetaan sivulle samaan tapaan kuin tavallinenkin syötekontrolli eli seuraavasti:

```
<CustomWeb:CustomLabel ID="lblEtusivu" runat="server" LokalisointiAvain="Etusivu">
</CustomWeb:CustomLabel>
```

Lokalisointiavain viittaa CustomLabelin tapauksessa KieliAvaimet_fi-FI resurssitiedostoon, joka löytyy HaagaHelia.Ont.Yleinen –projektista. Lokalisointi vastaa avainta, joka on resurssitiedostossa.



Kuva 5. Lokalisointiavain ja sitä vastaava arvo löytyvät resurssitiedostosta

Varsinainen lokalisointi suoritetaan syötekontrollin elinkaaren ”OnPreRender” -tapahtumassa. Resurssiavaimen haun suorittaa ”Lokalisoija”–luokan ”Kaanna”-metodi. ”Kaanna”-metodille syötetään lokalisointiavain ja se palauttaa käännetyn tekstin oikeasta resurssitiedostosta. ”Lokalisoija”–luokan ilmentymä on asetettu istuntoon, jossa on myös tieto käytetystä kulttuurista.

Kulttuurin vaihtaminen tehdään näin:

```
public void VaihdaKulttuuri(CultureInfo uusiKulttuuri)
{
    _edellinenKulttuuri = _nykyinenKulttuuri;
    _nykyinenKulttuuri = uusiKulttuuri;

    Thread.CurrentThread.CurrentCulture = uusiKulttuuri;

    ...
}
```

Ja tekstin kääntäminen tehdään näin:

```
public string Kaanna(string avain)
{
    return _resurssikasittelija.GetString(avain);
}
```

3.2.5 Lomakkeiden validointi

Lomaketyyppisille sivuille on asetettu validaattoreja. Osa näistä validaattoreista ovat nekin räätälöityjä, jotta lokalisointi toimisi. Lokalisointi hoidetaan perusvalidaattoreissa niin, että virheilmoitus asetetaan ”Page_Load” –tapahtumassa. Useimmiten on ollut tarpeen käyttää CustomValidatorista perittyä validaattoria, jotta monimutkaisemmat tarkistukset saataisiin tehtyä. Validaattoreita käytetään enimmäkseen opiskelijanäkymässä, koska siellä syötetään suurin osa järjestelmän tiedoista.

Validointeja tehdään mm. aihe-ehdotuksen vaadittavissa kentissä, joissa on käytetty ASP.NETin standardivalidaattoria *RequiredFieldValidator*. Toisissa tilanteissa on käytetty *CustomValidator*:a, josta on peritty *CustomValidaattori* (huomaa ero suomen ja englannin kielessä). Tätä on mm. käytetty tarkistamaan, että yhdellä opinnäytetyöllä voi olla vain yksi aloituskokous ja yksi päätöskokous.

Seuraavassa on esimerkki, miten validaattorin toiminta on toteutettu:

```
<CustomWeb:CustomValidaattori LokalisointiAvain="AloituskokousOlemassaTarkistus"
ID="valTarkistaAloituskokous" runat="server"
OnServerValidate="TarkistaOnkoAloituskokousOlemassa"
ValidationGroup="Tallennus"></CustomWeb:CustomValidaattori>
```

Codebehind:ssa tarkistusmetodi on seuraavanlainen:

```
protected void TarkistaOnkoAloituskokousOlemassa(object source, ServerValidateEventArgs
args)
{
    if (ucKentat.Vaihe == KokousVaihe.Aloituskokous)
    {
        valTarkistaAloituskokous.IsValid  =!
KokousBo.OnkoOpinnaytetyollaTietynVaiheenKokous(
                IstuntoAvustaja.KasittelyssaOlevaOpinnaytetyo.Kentat.ID,
                KokousVaihe.Aloituskokous,
                null);
        args.IsValid = valTarkistaAloituskokous.IsValid;
    }
}
```

Jos tarkistus ei mene läpi, annetaan validointiargumenttien ”IsValid” -ominaisuuden arvoksi asetetaan false. Näin sivu on automaattisesti ”Ei validi”-tilassa, jolloin tallennus keskeytyy.

```
protected void btnTallenna_Click(object sender, EventArgs e)
{
    Validate("Tallennus");
    if (IsValid)
    {
        KokousBo kokousBo = AsetaTiedot();

        try
        {
            kokousBo.Tallenna();
            Response.Redirect(NayttojenNimet.OpiskelijaKokousTallennettu);
        }
        catch (ApplicationException)
        {
            VirheAvustaja.OhjaaVirhesivulle();
        }
    }
}
```

Validaattorilla on oma validointiryhmä (ValidationGroup). Tällä tavalla voidaan erotella eri validaattorit ja niiden laukeamiset tietyssä kontekstissa. Aiemmassa koodiesimerkissä validointiryhmä on sidottu tallennustapahtumaan, jolloin nämä validaattorit laukeavat ainoastaan tallennustapahtumassa. Tällä tavalla ei käy niin, että jos käyttäjä peruuttaa tallennuksen, valitsemalla ”Peruuta”-painikkeen, validaattoreita ei käytetä, eivätkä ne estä sovelluksen käyttöä tarkoituksen mukaisesti.

3.2.6 Istunnon hallinta

Kirjautunut käyttäjä, käsittelyssä olevat liiketoimintaobjektit ja lokalisoija on talletettu käyttäjän istuntoon. Istunnon pituutta voidaan säätää web-sovelluksen konfigurointitiedostosta *web.config*:sta, joka ilmaistaan minuutteina. Oletuksena ONTiin on asetettu 120 minuuttia, mutta sitä voi muuttaa. *Web.config*:ssa istunnon pituus ja istuntotietojen tallennuspaikka ilmaistaan *sessionState* tagissa. Seuraavassa esimerkki:

```
<sessionState timeout="120" mode="InProc"></sessionState>
```

Mode attribuutti ilmaisee istunnon tietojen tallennuspaikan. Esimerkissä on arvona InProc eli tarkoittaa In-Process istunnon moodia, jossa istunnon tiedot pysyvät palvelimen muistissa niin kauan kun ASP.NET:n ”Working Process” -prosessin suoritusta ei katkaista tai kierrätetä (recycle).

Istunnon tietoja käsitellään käyttöliittymän ”IstuntoAvustaja” -apuluokassa. Jokaista istunnon julkista ominaisuutta luettaessa tarkistetaan, ettei istunto ole katkennut. Seuraavassa on esimerkki siitä miten tarkistus käytännössä tehdään:

```
/// <summary>
/// Käsittelyssä oleva aihe-ehdotus
/// </summary>
public static AiheEhdotusBo KasittelyssaOlevaAiheEhdotus
{
    get
    {
        TarkistaIstunto();
        return HttpContext.Current.Session["KasittelyssaOlevaAiheEhdotus"] as
AiheEhdotusBo;
    }
    set
    {
        HttpContext.Current.Session["KasittelyssaOlevaAiheEhdotus"] = value;
    }
}
```

Get -lohkossa kutsutaan metodia *TarkistaIstunto*.

```
private static void TarkistaIstunto()
{
    if (HttpContext.Current.Session.IsNewSession ||
HttpContext.Current.Session["KirjautunutHenkilo"] == null)
    {
        HttpContext.Current.Response.Redirect(NayttojenNimet.VirheSivutIstuntoPaattynyt);
    }
}
```

Istunto katkeaa jos istunnon omaava käyttäjä ei tee yhtäkään palvelinpyyntöä määräaikaan (timeout) mennessä. On otettava huomioon, että vaikka käyttäjä omasta mielestään tekisikin jotain sovelluksessa (kuten esimerkiksi syöttäisi tekstiä tekstiruutuun) se ei ole palvelimen mielestä palvelinpuolen aktiivisuutta, jolloin palvelin ei noteeraa käyttäjän tekemisiä selaimessa.

Jos istunto havaitaan päättyneeksi, käyttäjä ohjataan sivulle, jossa käyttäjälle ilmoitetaan istunnon päättymisestä ja kehotetaan oheisen linkin kautta palaamaan kirjautumissivulle ja kirjautumaan uudelleen järjestelmään.

Istunnossa on mukana myös lokalisointitieto eli istunto pitää yllä tietoa siitä mitä kieltä (kulttuuria) ollaan käyttämässä. Näin jokainen räätälöity syötekontrolli voi kääntää tekstin tämän tiedon mukaisesti.

3.2.7 JavaScript ja selaimet

ONTissa käytetään JavaScriptiä hyvin vähän. Näin saadaan eliminoitua selainriippuvaisuutta mahdollisimman paljon, koska palvelin hoitaa lähes kaiken tiedonkäsittelyn ja asiakasohjelman eli selaimen tehtävänä on vain näyttää käyttäjälle sovelluksen grafiikka.

ONTissa on käytetty kuitenkin JavaScriptin laajennusta JQuery:a, jonka avulla on saatu mm. päävalikkoon kauniit tehosteet painikkeiden valinnasta.
Viittaus JQuery:n käyttöön tulee jokaiseen Master-sivuun html:n ”head”-tagin sisään:

```
<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.5.2/jquery.min.js"></script>
                            <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jqueryui/1.8.12/jquery-ui.min.js"></script>
```

JQueryä käytetään aktivoimaan päävalikon linkit ”button”–tyyppisiksi käyttämällä CSS –tyylitiedostoa.

```
<script type="text/javascript">
    $(document).ready(function() {
        $('.jqSubmitButton').button();
        $('#button').button();
        $('#button1').button();
        $('#button2').button();
        $('#button3').button();
    });
</script>
```

3.2.8 Tyylitiedostot ja grafiikka

ONTissa käytetään yhtä tyylitiedostoa, joka on lisätty Master–sivuihin. Tyylitiedossa ovat sekä omat että JQuery:n tyylit, joita käytetään mm. painikkeiden graafiseen muuttamiseen.

Päävalikon painikkeisiin on käytetty omia tyylejä, joihin on myös käytetty kuvia, jotka ovat ”kuvat” –kansiossa. Näin on saatu aikaan kaunis sävytys painikkeisiin kun hiiren osoitin on niiden päällä.

Joissakin ”div”-elementeissä on käytetty *border radius* attribuuttia pyöristämään ”div”-elementtien kulmat, mutta nämä ominaisuudet eivät näy Internet Explorer 8:lla tai sitä vanhemmalla selaimella.

Tähän sovellukseen ei ole uhrattu grafiikkaan ylen määrin, sillä se ei olisi tarkoituksen mukaista. Järjestelmä palvelee käyttäjiä paremmin yksinkertaisella grafiikalla. Grafiikkaa tärkeämpiä seikkoja ovat turvallisuus, käytettävyys ja suorituskyky.

3.3 Sovelluslogiikka

Sovelluslogiikkasäännöt ovat pääasiassa luokilla, jotka sijaitsevat ”HaagaHelia.Ont.Liiketoiminta” –projektissa. Käyttöliittymällä on sovelluslogiikka lähinnä navigoinnin, istunnon hallinnan ja validoinnin suhteen. Voidaan sanoa, että silloin kyseessä ei ole liiketoimintalogiikasta vaan käyttöliittymäprosessilogiikasta.

3.3.1 Tieto-olion sisältävät liiketoimintaluokat

Liiketoimintaluokat on jaettu karkeasti kahteen tyyppiin: Tietoa sisältäviin ja muihin. Tietoa sisältävillä luokilla on ”–Bo” suffiksi; lyhenne joka tulee sanoista Business Object. Tietoa sisältävillä luokilla on kullakin oma tieto-olio. Viittaus tähän tieto-olioon löytyy ominaisuudesta *Kentat*. Esimerkiksi *AiheEhdotusBo* liiketoimintaluokalla on oma tieto-olio luokkansa ”Kentat” –ominaisuudessa: *AiheEhdotusTietoOlio*. UML–mallinnuksessa (katso liite 1: Luokkakaavio) *AiheEhdotusBo*:n suhde *AiheEhdotusTietoOlioon* on *composite* eli sitä ei jaeta muiden liiketoimintaolioiden kesken. Muutoin se olisi *aggregate*. Toisin sanoen, tieto-olio on osa liiketoimintaoliota. Luokkakaaviosta voi nähdä selkeästi tämän suhteen. Seuraavassa on esimerkki koodista, josta näkee suhteen toteutuksen.

```
public class AiheEhdotusBo : LiiketoimintaKanta, ITallennettava<AiheEhdotusTietoOlio>
{
    private AiheEhdotusTietoOlio _kentat;

    public AiheEhdotusTietoOlio Kentat
    {
        get { return _kentat; }
        set { _kentat = value; }
    }
...
```

Liiketoimintaolion konstruktorissa luodaan myös tieto-olio ja näin vältetään mahdolliset tilanteet, jossa null–tarkistus on jäänyt tekemättä.

Tietoa sisältävät liiketoimintaoliot perivät ”Liiketoimintakanta” –kantaluokan sekä toteuttavat ”ITallennettava” -rajapinnan. *LiiketoimintaKanta* sisältää yhden julkisen ominaisuuden *OlionTila*, joka ilmaisee onko liiketoimintaoliota muutettu, onko sen tiedot haettu tietokannasta vai onko luokan ilmentymä uusi.

”ITallennettava”–rajapinnalla on aiemmin mainittu julkinen ominaisuus *Kentat* toteutettavana sekä kolme paluuarvotonta metodia *AlustaLiiketoimintaluokka*, *Tallenna*, ja *Poista*. Jokaisen tietoa sisältävän liiketoimintaolion täytyy toteuttaa vähintään nämä metodit.

Liiketoimintaluokan tarkoituksena on tehdä sovelluksen päätelmät, laskemiset ja tarkistukset. Sen vuoksi jokaisella liiketoimintaluokalla on omat metodit, jotka sisältävät näitä edellä mainittuja toimintoja. Liiketoimintaluokan tarkoituksena on myös olla välikappaleena tietokantasovittajille ja säädellä miten dataa haetaan tietokannasta ja mitä sinne tallennetaan milloinkin eri tilanteissa. Pääpiirteissään kuitenkin tietoa sisältävät liiketoimintaluokat toimivat samaan tapaan.

Esimerkkinä otetaan aihe-ehdotuksen tallennus. Aihe-ehdotuksen tiedot kohdistetaan käyttöliittymäkerroksessa liiketoimintaluokan tieto-olion julkisiin ominaisuuksiin. Kuten aiemmin on tullut esille, tieto-olio sisältää tietokannan tietueen vastaavat tiedot sovelluskoodin muodossa. Esimerkiksi tietokannan ”nvarchar” -tietotyyppi vastaa ”string” -tyyppiä, tietokannan ”int”-tietotyyppi vastaa myös ”int” –tietotyyppiä sovelluskoodissa jne.

Näin käyttöliittymäkerroksella on suora pääsy tieto-olion julkisiin ominaisuuksiin, koska tieto-olion ilmentymä on julkinen (public) ominaisuus liiketoimintaoliolla. Käyttöliittymältä saadaan käyttäjän syöttämät syötteet ja ne voidaan asettaa tieto-olioon liiketoimintaolion kautta. Seuraavassa on esimerkki siitä miten tämä tapahtuu. Tässä tapauksessa tallennus tehdään aihe-ehdotukselle, jota ollaan lähettämässä ja jota ei ole tallennettu aiemmin.

```
private AiheEhdotusBo AsetaTiedot()
{
    AiheEhdotusBo aiheEhdotusBo = new AiheEhdotusBo();
    aiheEhdotusBo.Kentat.Nimi = ucKentat.Nimi;
    aiheEhdotusBo.Kentat.SisaltoKuvaus = ucKentat.SisaltoKuvaus;
    aiheEhdotusBo.Kentat.MerkitysKuvaus = ucKentat.MerkitysKuvaus;
    aiheEhdotusBo.Kentat.Aikataulu = ucKentat.Aikataulu;
    aiheEhdotusBo.Kentat.Menetelmat = ucKentat.Menetelmat;
    aiheEhdotusBo.Kentat.Tyovalineisto = ucKentat.Tyovalineisto;
    aiheEhdotusBo.Kentat.Toimeksiantaja = ucKentat.Toimeksiantaja;
    aiheEhdotusBo.Kentat.Edellytykset = ucKentat.Edellytykset;
    aiheEhdotusBo.Kentat.Opiskelija.Id = IstuntoAvustaja.KirjautunutHenkilo.Kentat.Id;

    return aiheEhdotusBo;
}
```

Sisäinen (private) metodi palauttaa aihe-ehdotuksen ilmentymän mukanaan myös aihe-ehdotuksen tunniste, (id) joka on jo olemassa tietokannassa.
”AsetaTiedot”–metodia on kutsuttu painikkeen tapahtumasta. Seuraavassa on esimerkki:

```
protected void btnLahetaAiheEhdotus_Click(object sender, EventArgs e)
{
    AiheEhdotusBo aiheEhdotusBo = AsetaTiedot();
    aiheEhdotusBo.Kentat.AiheEhdotusTila = AiheEhdotusTila.TallennettuKeskenerainen;

    try
    {
        aiheEhdotusBo.Tallenna();
        IstuntoAvustaja.KasittelyssaOlevaAiheEhdotus = aiheEhdotusBo;
        Response.Redirect(NayttojenNimet.OpiskelijaAiheEhdotusVahvistaLahetys);
    }
    catch (ApplicationException)
    {
        VirheAvustaja.OhjaaVirhesivulle();
    }
}
```

Kun liiketoimintaolion ”Tallenna” -metodia kutsutaan, se tekee päättelyn sisäisesti siitä, onko objekti uusi vai jo olemassa. Päättelyn perusteella se kutsuu ”AiheEhdotusSovittaja” –luokan ilmentymän (sovittajista enemmän seuraavassa luvussa) metodia, jonka tehtävänä on välittää tiedot tietokantapalvelimelle. Alla esimerkki liiketoimintaluokan sisäisestä päättelystä:

```
public void Tallenna()
{
    AiheEhdotusSovittaja sovittaja = new AiheEhdotusSovittaja();
    if (OlionTila == Tila.Uusi)
    {
        _kentat.Luomisaika = DateTime.Now;
        int id = sovittaja.LisaaUusiJaPalautaID(_kentat);
        _kentat.Id = id;
    }
    else
    {
        sovittaja.Muokkaa(_kentat);
    }
    OlionTila = Tila.EiMuutettu;
}
```

Viestiyhteyskaaviosta (katso Liite 3. Viestiyhteyskaavio) voi nähdä aihe-ehdotuksen lähetyksen toiminnallisuuden suhteessa eri kerrosten luokkiin.

Tietojen haku toimii hieman samaan tapaan kuin tiedon tallennus ja poisto, mutta tietojen haussa ei kutsuta ilmentymän vaan luokan staattisia (static) metodeja.

Seuraavassa on esimerkki luokkakohtaisesta metodista, jossa haetaan lista tieto-olioita ja sovitetaan ne liiketoimintaoliolistaan.

```
/// <summary>
/// Hakee opiskelijan aihe-ehdotukset opiskelijan ID:n mukaan.
/// </summary>
/// <param name="id">Kirjautuneen henkilön ID</param>
/// <returns>Palauttaa listan opiskelijan aihe-ehdotuksista</returns>
public static List<AiheEhdotusBo> HaeOpiskelijanAiheEhdotukset(int id)
{
    AiheEhdotusSovittaja sovittaja = new AiheEhdotusSovittaja();
    return
SovitaTietoOliotLiiketoimintaOlioihin(sovittaja.HaeOpiskelijanAiheEhdotukset(id));
}

...

private static List<AiheEhdotusBo>
SovitaTietoOliotLiiketoimintaOlioihin(List<AiheEhdotusTietoOlio> oliot)
{
    List<AiheEhdotusBo> aiheEhdotukset = new List<AiheEhdotusBo>();

    foreach (AiheEhdotusTietoOlio item in oliot)
    {
        AiheEhdotusBo aiheEhdotusBo = new AiheEhdotusBo();
        aiheEhdotusBo.OlionTila = Tila.EiMuutettu;
        aiheEhdotusBo.Opiskelija = new HenkiloBo(item.Opiskelija);
        aiheEhdotusBo.Kentat = item;
        aiheEhdotukset.Add(aiheEhdotusBo);
    }
    return aiheEhdotukset;
}
```

3.3.2 Muut liiketoimintaluokat

Muita liiketoimintaluokkia ovat *EdeltavyysTarkastaja*, *EmailLahettaja* ja *ICalLuoja*.

EdeltavyysTarkastajan tarkoituksena on tehdä tarvittavat tarkistukset opiskelijoiden pakollisten opintojen edellytysten täyttymisestä. Luokassa on staattinen metodi *OnkoSuoritusOK*, jolle välitetään opiskelijan käyttäjätunnus ja opintojakso, joka on opiskelijan koulutusohjelman pakollinen opinto tietyssä koulutusohjelman toteutuksessa. Tämän metodin toteutus ei kuulunut tähän opinnäytetyöhön, koska sen toiminnallisuus liittyy Winha Willen rajapintaan. *EdeltavyysTarkastaja* käy läpi kaikki opiskelijan pakollisten opintojen toteutukset ja palauttaa tiedon siitä, onko opiskelija suorittanut kaikki pakolliset opinnot vai ei. Tämä on edellytys opiskelijan ONTin käytölle. Tämä tarkistus voidaan kuitenkin ohittaa, mikäli koordinaattori omasta näkymästään antaa siitä merkinnän kyseiselle opiskelijalle.

EmailLahettaja nimensä mukaisesti lähettää kuittausviestejä eri tilanteissa ONTia käytettäessä. Toiminnallisuus on valmis, mutta tässä kehitysvaiheessa ei ole osattu vielä

päättää milloin ja millaisia viestejä lähetetään, joten käyttötapauksissa, jossa tätä luokkaa käytettäisiin, eivät kuitenkaan käytä sitä.

ICalLuojan ainoana tarkoituksena on luoda ”KokousBo” -liiketoimintaluokasta ICS-tiedosto, joka on formaatti iCalendar-standardille. Tätä formaattia käytetään monissa kalenterisovelluksissa kuten Microsoft Outlook:ssa, Lotus Notes:ssa ja Mozilla Thuderbird:ssa eli niiden avulla voidaan luoda uusia kalenterimerkintöjä. Luokkaa kutsutaan kun käyttäjä (ohjaaja tai koordinaattori) valitsee kokouksen ”Lataa” -linkin, hän saa automaattisesti dialogi-ikkunan, jossa häntä kehotetaan tallentamaan kalenterimerkintä.

Tämä on kätevä ominaisuus, jonka avulla erityisesti opinnäytetöiden ohjaajat voivat seurata helposti opinnäytetöiden kokousten alkamisaikoja- ja paikkoja. *ICalLuoja* on räätälöity toimimaan ainoastaan GMT +02:00 (kesäaika) aikavyöhykkeellä.

3.4 Tietokanta ja sovittajat

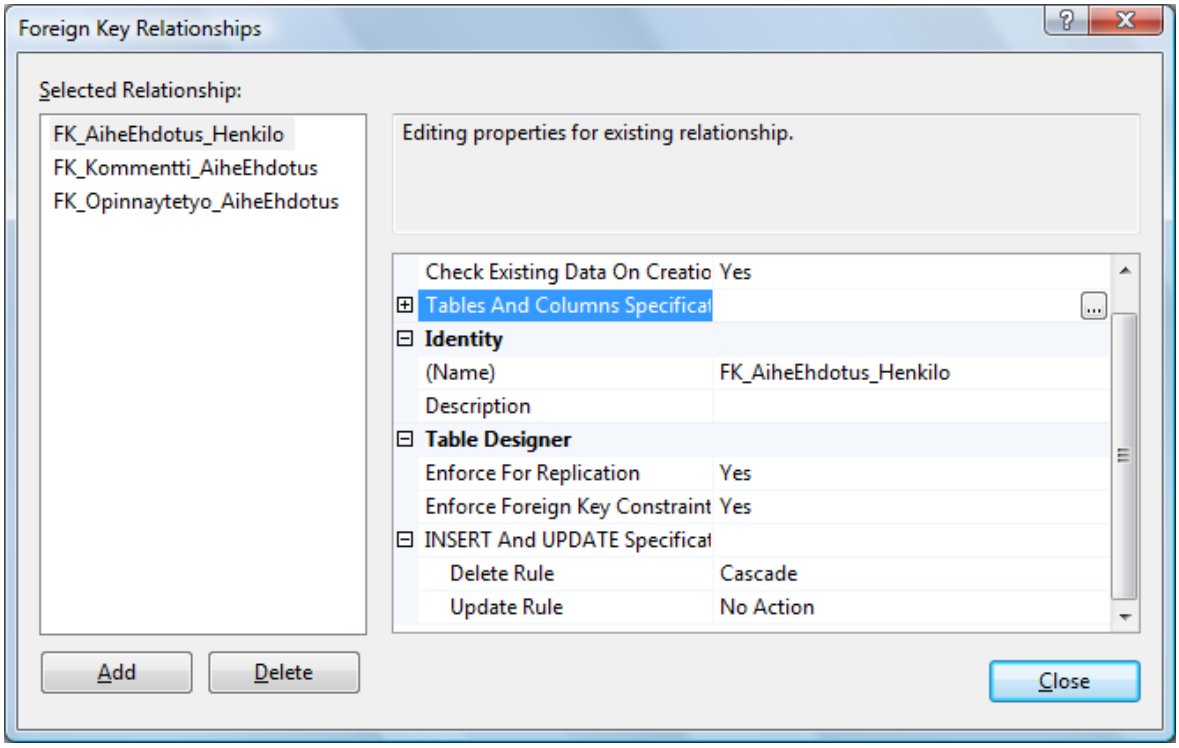
3.4.1 Tietokannan rakenne ja ominaisuudet

Tietokanta on asennettu SQL Server 2005 tietokannanhallintajärjestelmään (DBMS). Tietokanta sisältää 14 taulua, yhden näkymän ja yli 80 ”Stored procedure” –komentoa, jotka toimivat tietokannan haku- ja käsittelytehtävissä. Tarkoituksena oli, että tietokantarajapinta (ei sovittaja) päättää siitä, miten dataa tallennetaan ja miten sitä käsitellään yms. Tällä tavalla vaikka tietokannan rakenteeseen tulee muutoksia, tulokset sovelluksen kannalta pysyvät samana ja sovittajarajapintaan tehtävät muutokset ovat olemattomia tai erittäin vähäisiä.

Tietokannan kollaatioksi on määritetty *Finnish_Swedish_CI_AI*. Näin suomen- ja ruotsinkieliset aakkoset saadaan oikeaan järjestykseen, kun tehdään kyselyjä. CI tarkoittaa *Case Insensitive* eli isoja ja pieniä kirjaimia ei erotella kun merkkijono järjestetään, vaan iso ja pieni kirjain ovat samanarvoiset. AI tarkoittaa *Accent Insensitive*, eli vokaalien aksenttimerkillä ei ole merkitystä järjestystä tehdessä. Suomen, ruotsin ja englannin kielessä käytetään vain vähän aksenttimerkkejä.

Tietokannan *Recovery Model* on syytä olla *Full*, jotta tietokannan palautustilanteessa voidaan päästä mahdollisimman tarkkaan kellonaikaan ja palauttaa transaktiolokista puuttuva data. Loput tietokantakohtaiset asetukset määrittää järjestelmän tilaaja omien tarpeittensa mukaiseksi.

Tauluihin on määritetty myös viite-eheydet eri taulujen välillä. Liitteestä 4 voi nähdä taulujen väliset relaatiot ja viite-eheydet. Viite-eheys määritetään SQL Serverissä niin, että mennään taulun ”Design”–tilaan, valitaan muokattava sarake, valitaan *Relationships*, kontekstivalikosta ja näin saadaan auki viite-eheys valintojen dialogi.



Kuva 6. Viite-eheyksien määrittäminen SQL Serverissä

”Tables And Columns Specification” –osioista valitaan taulujen ja sarakkeiden väliset suhteet. Toinen sarakkaista on aina *Primary key* ja toinen voi olla *Primary* tai *Foreign Key*. *INSERT And UPDATE Specification* kertoo, mitä tapahtuu kun pääavaimen taulusta poistetaan tietty rivi. *Delete Rule: Cascade* ilmaisee, että kaikki rivit, joilla on suhde kyseiseen pääavaimen riviin, poistetaan.

Tauluissa merkkijonosarakkeissa on käytetty lähinnä ”nvarchar”–tietotyyppiä. Tämä johtuu puhtaasti siitä, että tietokantaan saadaan myös Unicode-tekstiä, vaikka yksi

merkki viekin kaksi tavua. Järjestelmää tällöin voidaan käyttää kielillä, jotka eivät käytä latinalaisia aakkosia.

”Stored Procedure” -komentoja kutsutaan ohjelmakoodista ADO.NET Framework:n kautta. ”Stored Procedure” -komennoilla on nimi ja parametrit, jotka välitetään ADO.NET:n rajapintaluokkien kautta. ”Stored Procedure” -komennoisa on erilaisia kyselyjä ja tietokannan käsittelykomentoja eri järjestelmän tarpeeseen. Jossain kyselyssä on tarvittu taulujen välillä moni-moneen suhdetta ja johon on tarvittu useampaa taulua. Taulut *Edellytys*, *Toteutus* ja *EdelleytysToteutus* toteuttavat tämän moni-moneen suhteen, joka ilmaisee että yhdellä Edellytyksellä (pakollisella opinnolla) voi olla useampi Toteutus (eli sama pakollinen opinto saattaa olla useammassa koulutusohjelman toteutuksessa eri opintojaksotunnuksella). Ja toisaalta yhdellä Toteutuksella voi olla ja onkin useampi Edellytys eli pakollinen opinto. Seuraavassa on esimerkki miten kysely hakee kaikki tietyn koordinaattorin määrittämät opintojaksojen toteutukset eri pakolliselle opinnolle.

```
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:                  Marko Ruotsalainen
-- Create date: 02.09.2011
-- Description:             Hakee opintojaksot, jotka toteuttavat pakolliset
-- opinnot koordinaattorin id:n mukaan, järjestettynä edellytyksen id:n mukaan
-- =============================================
ALTER PROCEDURE [dbo].[EdellytysToteutusHaeKoordinaattorinMukaanJarjestaEdellytyksenMukaan]
              -- Add the parameters for the stored procedure here
              @KoordinaattoriID int
AS
BEGIN
              -- SET NOCOUNT ON added to prevent extra result sets from
              -- interfering with SELECT statements.
              SET NOCOUNT ON;

    SELECT T.ToteutusID, T.Nimi AS ToteutusNimi, T.KoordinaattoriID AS Koordinaattori,
    E.EdellytysID, E.Nimi AS EdellytysNimi, ED.Opintojakso, ED.OpintojaksoID

    FROM Edellytys E
    JOIN EdellytysToteutus ED ON E.EdellytysID = ED.EdellytysID
    JOIN Toteutus T ON ED.ToteutusID = T.ToteutusID
    WHERE T.KoordinaattoriID = @KoordinaattoriID
    AND E.KoordinaattoriID = @KoordinaattoriID

              ORDER BY E.EdellytysID
END
```

Kuten koodista näkyy, tähän on tarvittu kahden muun taulun yhdistämistä, jotta saadaan haluttu joukko tuloksia tietokannasta.

3.4.2 Tietokantasovittajat

Dataa ei voi noutaa kannasta ilman ”Stored Procedure” -komentoa, mutta tiedon saattamiseksi sovellukselle sopivaksi tarvitaan sovittajia. Sovittajan tehtävänä on avata tietokantayhteys palvelimelle niin lyhyeksi ajaksi kuin on mahdollista, sovittaa tarvittavat tieto-olion tiedot tietokantakyselylle sopivaksi ja toisinpäin ja vapauttaa käytetyt tietokantaresurssit, kuten tietokannan lukijan (SqlDataReader). Tämä on hyvin tärkeää, sillä tietokannan lukija, jota ei ole suljettu, useamman säikeen käyttäessä samoja tietokannan resursseja voi aiheuttaa tietokantapalvelimen ”dead lock” -tilanteen, jossa mikään säie ei pääse varattuun resurssiin (kuten esimerkiksi tauluun). Tätä ongelmaa ei tapahdu tietokantaan kirjoitettaessa, mutta sieltä lukiessa se on kriittinen asia.

Tietokantasovittaja perii ”SqlKanta”-kantaluokan, joka mm. lukee ja pitää sisällään tietokantayhteyden merkkijonoa (connection string). Näin kukin sovittajaluokka tietää minne tietokantaan operaatiot tehdään. Sovittajat myös toteuttavat taulukohtaisen rajapinnan sekä yleisen ”ISovittaja”–rajapinnan. Rajapinnoissa on määritetty julkiset metodit, jotka tulee toteuttaa. Taulukohtaiset rajapinnat on tehty siitä syystä, että tietokantarajapinta saataisiin tietokantariippumattomaksi. Näin on mahdollista käyttää samoja rajapintoja toteuttamaan sovittajat eri tietokannanhallintajärjestelmille kuten MySQL:lle tai Oraclelle. Tietokantasovittajat löytyvät projektista HaagaHelia.Ont.Data. Nimiavaruus on HaagaHelia.Ont.Data.Sql, juuri sen vuoksi, että kyseisessä nimiavaruudessa sijaitsevat Sql Server 2005:n sovittajat.

Seuraavassa on datan käsittelyä varten tehty sovittajametodi:

```
/// <summary>
/// Lisää uuden aihe-ehdotuksen tietokantaan.
/// </summary>
/// <param name="tietoOlio">Aihe-ehdotuksen tiedot, jotka tallennetaan tietokantaan</param>
public void LisaaUusi(AiheEhdotusTietoOlio tietoOlio)
{
    try
    {
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            con.Open();
            SqlCommand command = new SqlCommand();
            command.Connection = con;
            command.CommandType = CommandType.StoredProcedure;
            command.CommandText = "AiheEhdotusLisaaUusi";
command.Parameters.Add(new SqlParameter("OpiskelijaID", tietoOlio.Opiskelija.Id));
            command.Parameters.Add(new SqlParameter("Nimi", tietoOlio.Nimi));
command.Parameters.Add(new SqlParameter("SisaltoKuvaus", tietoOlio.SisaltoKuvaus));
command.Parameters.Add(new SqlParameter("MerkitysKuvaus", tietoOlio.MerkitysKuvaus));
            command.Parameters.Add(new SqlParameter("Aikataulu", tietoOlio.Aikataulu));
command.Parameters.Add(new SqlParameter("Tyovalineisto", tietoOlio.Tyovalineisto));
            command.Parameters.Add(new SqlParameter("Menetelmat", tietoOlio.Menetelmat));
            command.Parameters.Add(new SqlParameter("Toimeksiantaja",
tietoOlio.Toimeksiantaja));
command.Parameters.Add(new SqlParameter("Edellytykset", tietoOlio.Edellytykset));
command.Parameters.Add(new SqlParameter("AiheEhdotusTila", tietoOlio.AiheEhdotusTila));
            command.Parameters.Add(new SqlParameter("Luomisaika", tietoOlio.Luomisaika));
            command.Parameters.Add(new SqlParameter("AiheEhdotusID", SqlDbType.Int));
            command.ExecuteNonQuery();
        }
    }
    catch (SqlException ex)
    {
        Lokitus.LokitaVirhe(ex);
        throw new ApplicationException();
    }
}
```

Metodin sisällä using avainsana viittaa ”IDisposable”–rajapinnan toteuttavaan objektiin, toisin sanoen ”SqlConnection”-luokan ilmentymän resurssit vapautetaan (tässä tarkoittaa tietokantayhteyden katkaisemista), kun koodin suoritus pääsee using –lohkon loppuun. Metodissa luodaan uusi yhteys, määritetään komentotyypiksi *Stored Procedure* ja annetaan komennon nimi. Komennon parametreille annetaan liiketoimintaluokan ilmentymän välittämä tieto-olio objekti, joka sisältää tallennettavan datan. On kätevää välittää parametrina ainoastaan tieto-olio, koska datarakenteen muuttuessa ei ole tarpeen muuttaa metodin parametreja (argumentteja), vaan tieto-olion ilmentymä kelpaa aina sovittajalle.

”ExecuteNonQuery” -metodi suorittaa komennon eli välittää datan ”Stored Procedure” -komennolle, joka suorittaa tietokantapalvelimen sisällä kyseisen komennon.

Datan hakeminen kannasta tehdään hieman samaan tapaan:

```
/// <summary>
/// Hakee olion kannasta id:n perusteella
/// </summary>
/// <param name="id">Aihe-ehdotuksen id</param>
/// <returns>Yhden aihe-ehdotuksen tiedot</returns>
public AiheEhdotusTietoOlio HaeOlio(int id)
{
    try
    {
        AiheEhdotusTietoOlio aiheEhdotus = new AiheEhdotusTietoOlio();
        using (SqlConnection con = new SqlConnection(_connectionString))
        {
            con.Open();
            SqlCommand command = new SqlCommand();
            command.Connection = con;
            command.CommandType = CommandType.StoredProcedure;
            command.CommandText = "AiheEhdotusHaeYksi";
            command.Parameters.Add(new SqlParameter("AiheEhdotusID", id));

            using (SqlDataReader reader = command.ExecuteReader())
            {
                if (reader.Read())
                {
                    aiheEhdotus.Id = int.Parse(reader["AiheEhdotusID"].ToString());
                    aiheEhdotus.Opiskelija.Id =
int.Parse(reader["OpiskelijaID"].ToString());
                    int i;
                    if (int.TryParse(reader["OhjaajaID"].ToString(), out i))
                    {
                        aiheEhdotus.Ohjaaja.Id =
int.Parse(reader["OhjaajaID"].ToString());
                    }
                    else
                    {
                        aiheEhdotus.Ohjaaja = null;
                    }


                    aiheEhdotus.Nimi = reader["Nimi"].ToString();
                    aiheEhdotus.SisaltoKuvaus = reader["SisaltoKuvaus"].ToString();
                    aiheEhdotus.MerkitysKuvaus = reader["MerkitysKuvaus"].ToString();
                    aiheEhdotus.Aikataulu = reader["Aikataulu"].ToString();
                    aiheEhdotus.Tyovalineisto = reader["Tyovalineisto"].ToString();
                    aiheEhdotus.Menetelmat = reader["Menetelmat"].ToString();
                    aiheEhdotus.Toimeksiantaja = reader["Toimeksiantaja"].ToString();
                    aiheEhdotus.Edellytykset = reader["Edellytykset"].ToString();
                    aiheEhdotus.AiheEhdotusTila = (AiheEhdotusTila)
Enum.Parse(typeof(AiheEhdotusTila), reader["AiheEhdotusTila"].ToString());
                }
            }
        }
        return aiheEhdotus;
    }
    catch (SqlException)
    {
        throw new ApplicationException();
    }
}
```

Erona aiempaan sovittajan metodiin on siinä, että ”SqlConnection” –luokan ilmentymä suorittaa kyselyn, jonka paluuarvo on ”SqlDataReader” -tyyppinen joka palautetaan tietokannan lukijaobjektille. Metodi luo uuden tieto-olion ja sovittaa tietokannan lukijan datan tieto-oliolle sopivaksi. Lopuksi tieto-olio palautetaan kutsujalle eli liiketoimintaluokan ilmentymälle.

Palataksemme vielä hetkeksi käyttöliittymäkerrokseen, tilanteissa, joissa tarvitaan lista näytettävistä liiketoimintaolioiden tiedoista, käyttöliittymä pyytää liiketoimintakerrokselta ”List” -kokoelman, jotka sisältävät tietyntyyppisiä liiketoimintaolioita. Oliot sidotaan ”Repeater” web-komponenttiin kutsumalla ”DataBind” –metodia. Seuraavassa on esimerkki:

```
private void LataaOhjaajat()
{
    try
    {
        rptOhjaajat.DataSource = HenkilostoBo.HaeKaikkiOhjaajat();

        rptOhjaajat.DataBind();
    }
    catch (ApplicationException)
    {
        VirheAvustaja.OhjaaVirhesivulle();
    }
}
```

3.5 Muut tekniset ratkaisut

3.5.1 Virhekäsittely

Mikäli sovelluksessa tapahtuu hallitsematon virhe (kaikki yleisimmät virhetilanteet, on pyritty hallitsemaan sovelluksesta käsin), niin siitä tulee virhelokiin (Event Log) kirjaus. Jotta kirjaus toimisi, NETWORK SERVICE (ASP.NETin ajokäyttäjätunnus) –tunnuksella tulee olla oikeus kirjata Windows-lokiin virheilmoituksia. Näin mahdollisten hallitsemattomien virheilmoitusten tutkiminen onnistuu vain niille käyttäjille, joilla on oikeus kirjautua palvelimen käyttöjärjestelmään.

Järjestelmän käyttäjälle näytetään hallitsemattomassa virheessä yleinen virhesivu, jossa käyttäjä voi enintään palata edelliselle sivulle. Näin sovelluksen arkaluonteista dataa ei näytetä käyttäjälle, josta mm. voisi nähdä pinomuistin tiedot eli *StackTrace:n*.

Virhetilanne voisi sattua esimerkiksi tietokantaoperaatiossa. Tällöin poikkeus kirjataan lokiin.

```
try
{
    using (SqlConnection con = new SqlConnection(_connectionString))
    {
        …

        command.ExecuteNonQuery();

        return int.Parse(param.Value.ToString());
    }
}
catch (SqlException ex)
{
    Lokitus.LokitaVirhe(ex);
    throw new ApplicationException();
}
```

Virheestä heitetään kutsujalle uusi ”ApplicationException” -poikkeusobjekti ja se otetaan kiinni käyttöliittymäkerroksessa. Käyttöliittymäkerroksessa *VirheAvustaja* ohjaa käyttäjän yleiselle virhesivulle

```
try
{
    IstuntoAvustaja.KasittelyssaOlevaIlmoitus.Tallenna();
    lblHuomautus.Text =
IstuntoAvustaja.IstunnonLokalisoija.Kaanna("JarjestelmaIlmoitusTallennettuOnnistuneesti");
}
catch (ApplicationException)
{
    VirheAvustaja.OhjaaVirhesivulle();
}
```

3.5.2 Cross-Site-Scripting injektioiden torjuminen

Kaksi yleisintä verkkohyökkäyksen muotoa ovat XSS eli ”Cross-Site-Scripting” –injektiot ja SQL-injektiot. Usein hyökkääjät syöttävät huomaamattomia skriptejä tavallisen tekstin sekaan, joten muut käyttäjät eivät välttämättä huomaa olleensa hyökättyinä. ONTissa XSS-hyökkäysten torjuminen on hoidettu ilmaisella Microsoftin *AntiXSSLibrary*:n ominaisuuksilla, joiden ansiosta käyttäjien syötteet, jotka näytetään web–sovelluksessa, koodataan niin, ettei käyttäjän syöttämää haitallista (malicius) koodia suoriteta sovelluksessa. Näin saadaan myös selville, onko jokin käyttäjistä tarkoituksellisesti halunnut aiheuttaa vahinkoa muille käyttäjille.

Käyttäjän syötteitä XSS:n suhteen ei suodateta, vaan syötteiden näyttämisvaiheessa (eli ennen kuin data renderöidään HTML:ksi) *AntiXSSLibrary*:n ”Encoder”-luokka suorittaa koodaukseen HTML:ään sopivaksi

3.5.3 Datan kryptaus

Käyttäjien tunnukset ja sovelluksen konfigurointitiedostot kryptataan. Järjestelmäkuvauksessa on aiheesta enemmän.

3.5.4 Active Directory

Active Directory (myöhemmin AD) on olennainen osa järjestelmää. Sitä käytetään mm. käyttäjien tunnistamiseen HAAGA-HELIAn verkossa. ”HaagaHelia.Ont.ActiveDirectory”–projektissa oleva ainoa luokka *AdAvustaja* sisältää järjestelmän tarvittavat toiminnot. Tunnistautuminen järjestelmään tehdään niin, että *AdAvustajan* ”TunnistaKayttaja”–metodille annetaan käyttäjätunnus ja salasana, joka”PrincipalContext”–luokan ilmentymällä tarkistaa, onko käyttäjää olemassa ja onko vastaava salasana oikein. Metodi palauttaa totuusarvon eli onnistuiko tunnistaminen vai ei. Tätä metodia käytetään kun ollaan kirjautumassa järjestelmään.

Seuraavassa esimerkki siitä miten tunnistautuminen tapahtuu

```
public bool TunnistaKayttaja(string kayttajatunnus, string salasana)
{
    PrincipalContext pc = new PrincipalContext(ContextType.Domain, ADDomain);
    bool onValidi = false;
    onValidi = pc.ValidateCredentials(kayttajatunnus, salasana);

    return onValidi;

}
```

AD–rajapintaa käytetään myös näyttämään järjestelmässä käyttäjätunnuksen sijaan pelkästään käyttäjän nimi ja sukunimi. Tästä huolehtii ”HaeKayttajanKokoNimi” –metodi.

Rajapinnan avulla tehdään myös ohjaajien haku koordinaattorin näkymässä. Haun perusteella on käyttäjän etunimi ja/tai sukunimi tai niiden osa sekä AD:n hierarkisessa muodossa oleva AD-ryhmä, josta käyttäjiä haetaan. Metodi palauttaa tieto-oliot, jotka edustavat löydettyjä henkilöiden tietoja hakukriteerien perusteella.

Seuraavassa on esimerkki miten haku käytännössä on toteutettu:

```
public List<HenkiloTietoOlio> HaeKayttajaNimenJaRyhmanMukaan(string groupName, string name,
string lastname)
{
    SearchResultCollection results;

    DirectorySearcher search = new DirectorySearcher();

    search.Filter =
"(&(objectClass=User)(givenName="+name+")(sn="+lastname+")(memberOf="+groupName+"))";

    search.PropertiesToLoad.Add("member");
    results = search.FindAll();

    ArrayList userNames = new ArrayList();

    List<HenkiloTietoOlio> henkilosto = new List<HenkiloTietoOlio>();

    if (results != null)
    {

        foreach (SearchResult result in results)
        {
            DirectoryEntry entry = result.GetDirectoryEntry();

            HenkiloTietoOlio henkilo = new HenkiloTietoOlio();

            henkilo.KokoNimi = ((string)entry.Properties["givenName"][0]) + " " +
((string)entry.Properties["sn"][0]);
            henkilo.Kayttajatunnus = (string)entry.Properties["sAMAccountName"][0];
            henkilosto.Add(henkilo);
        }
    }
    return henkilosto;
}
```

Haun voi suorittaa ainoastaan käyttäjä, jolla on oikeus selata AD:tä. Siksi ”web.config” -konfigurointitiedostossa on parametrit, jotka määräävät mikä käyttäjä voi tehdä kyseisen haun. Järjestelmäkuvauksessa (liite 5) on enemmän siitä, miten kyseinen konfigurointi tehdään.

3.5.5 Web-sovelluksen konfigurointi

Web-sovelluksen konfiguroinnista ja sen parametreista kerrotaan enemmän järjestelmäkuvauksessa, eikä siitä puhuta tässä raportissa, koska kyseessä on salaista tietoa.

4 Työn arviointi ja pohdintaa

4.1 Tausta työlle

HAAGA-HELIAn ammattikorkeakoulu tarvitsi tietojärjestelmää, joka mahdollistaisi helpomman ja kätevämmän tavan seurata opiskelijoiden opinnäytetöitä ja niiden etenemistä. Tätä tarkoitusta varten aloin kehittämään ASP.NET –tekniikkaa käyttäen web-sovellusta, joka täyttäisi tämän tarpeen käyttämällä aiempaa laadittua määrittelykuvastoa, joka oli tehty eräällä Ohjelmistoprojekti –kurssin toteutuksella.

4.2 Tulosten analysointi

Lähtökohtana sovelluksen kehittämiselle oli, että sitä olisi helppo jatkokehittää ja laajentaa. ASP.NET-kehitysmalli auttoi luomaan erilliset ”Visual Studio”–projektit kutakin arkkitehtuurikerrosta varten. Näin oli helppo pitää erillään tietyt kokonaisuudet ja pitää kokonaisuuksien väliset suhteet eheänä. Jos joku muu alkaisi kehittää tehtyä koodia, sovelluksen logiikka on melko helposti ymmärrettävissä. Lyhyet ja selkeät koodikokonaisuudet olivat mahdollisia juuri ASP.NET:n olio-ohjelmointimallin vuoksi. Koodi on hyvin ymmärrettävää, metodien pituudet ovat lyhyitä, muuttujien ja metodien nimet ovat yksiselitteisiä ja koodin kommentitkin tukevat luettavuutta. Koodi on helposti ylläpidettävää, koska koodikokonaisuudet on jaettu ja pilkottu järkevästi. Tämä tietenkin edellyttää, että samaa ohjelmointi- ja kehitystapaa noudatetaan myös jatkokehitysvaiheessa.

Sovellukseen on mahdollista lisätä uusia kieliä helposti ja kielikäännökset ovat helppo toteuttaa ilman, että kääntäjän tulisi osata ohjelmoida. Käännetyt tekstit ovat yksinkertaisissa XML-tiedostoissa, joita on helppo ylläpitää.

Sovelluksesta pyrin tekemään mahdollisimman turvallisen ja suorituskykyisen ja omien testien ja erään HAAGA-HELIAn henkilöstöön kuuluvan koordinaattorin kanssa yhdessä saimme testattua sovellusta parhaamme mukaan. Eri asia on sitten käyttöönottotestaus ja pilotointivaiheet, jolloin käyttötilanteet tosi elämässä realisoituvat. Valitettavasti itselläni ei ollut aikaa ja resursseja suorittaa kauttaaltaan laaja järjestelmätestaus. Siksi tämä osa rajattiin opinnäytetyöstäni pois. Tulos omasta

mielestäni on hyvää jälkeä ja siitä voi olla ylpeä. Tulos täyttää asiakkaan vaatimat määritykset ja siinä on lisäksi ominaisuuksia, jotka nostavat järjestelmän arvoa.

Sovellus on mahdollisimman turvallinen sovelluskehittäjän näkökulmasta, mutta kaikkeen sovelluskehittäjäkään ei voi vaikuttaa kun puhutaan tietoturvasta. Tietoturva on käsitteenä hyvin laaja, mutta sovelluksen osalta tietoturva on hyvin kunnossa. XSS- ja SQL-injektioiden tekeminen torjutaan tehokkaasti ja data kryptataan niissä sovelluksen osissa, joissa se on järkevää tehdä. Ottaen huomioon myös järjestelmän kriittisyysasteen, se on huomattavasti turvallisempi kuin useat muut sovellukset, joiden näkyvyys on Internet. ONT on tällä hetkellä Intranet-sovellus, mutta tulevaisuudessa siitä saattaa tulla Extranet-sovellus, HAAGA-HELIAn Extranettiin. Tästä tietenkin päättää koulu itse.

Tietokannasta tuli mahdollisimman eheä ja suorituskykyinen kokonaisuus. Järjestelmän suorituskyvyn tehokkuuden huomaa sitten kun se on asennettu aitoon ympäristöön. Olen huomannut, että on aivan eri asia ajaa koodia Visual Studiolla Yhdysvalloissa olevaa tietokantapalvelinta vasten kuin ajaa käännettyjä binääritiedostoja IIS–palvelimen päällä, jonka tietokantapalvelin on samassa sisäverkossa.

Sovelluksessa on tiettyjä kohtia, joissa koodinpätkät toistuvat ja myönnän, että ajan puute ei mahdollistanut koodin optimointia. Mielellään olisin kokeillut myös eri tietokantoja järjestelmässä, mutta pitäydyin tutussa turvallisessa SQL Serverissä.

Sovelluksen kehityksessä olisi voinut käyttää myös ”ASP MVC” -mallia ”Web Forms” -mallin sijaan. Web Forms alkaakin olla pikkuhiljaa vanhentunut, vaikka se onkin edelleen suosittu ja sitä pidetään ASP.NET–tekniikan ytimenä ja sydämenä. MVC–malli olisi antanut joustavuutta käyttöliittymän dynaamisuuteen, vaikka siihen ei mielestäni tällaisessa järjestelemässä olisi tarvetta. Sellainen sopisi enemmän kaupalliselle tuotteelle. Valitettavasti MVC-kehitysmalli ei ole vielä minulle kovin tuttua, vaikka aiheena se on minusta kiinnostava. Odotan työelämässä käyttäväni kyseistä mallia tulevaisuudessa. Tällä kertaa pitäydyin tutussa ja turvallisessa.

Sovellusta oli hankala välillä testata ja kehittää, koska sen täysi toiminta vaati olemaan HAAGA-HELIAn sisäverkossa. Tästä syystä usein jouduin tekemään testejä fyysisesti istuen HAAGA-HELIAn Pasilan toimipisteessä. Olin kuitenkin tehnyt sovellukseen konfigurointiparametrin, jonka avulla pystyin kehittämään ja testaamaan suurinta osaa koko järjestelmän toiminnasta. Myös minun oli vaikeaa saada tukea IT-palveluilta, luultavasti koska pyyntöni tukeen olivat epätavallisia verrattuna opiskelijaan, joka esimerkiksi ei vaan pääse omaan sähköpostitiliinsä. Mm. ei ole tavanomaista pyytää käyttöönsä käyttäjätunnusta, jolla pääsisi selaamaan koko koulun verkkoa. Kaikesta huolimatta pääsin yli näistäkin haasteista. Lopulta löytyi käyttäjätunnus, jonka avulla oli mahdollista selata koulun verkkoa.

Kaiken kaikkiaan käytin työn tekemiseen suuren määrän aikaa. Järjestelmän kehitys ei ollut mikään pieni tehtävä, mutta ajan kanssa sain kaikki osuudet tehtyä. Avonaiseksi kysymykseksi tietysti jää, aikooko HAAGA-HELIA koskaan ottaa käyttöön tätä järjestelmää.

4.3 Kehitysehdotukset ja jatkotoimenpiteet

ONT ei ole käyttövalmis luovutuksen yhteydessä. Se vaatii vähintään ”Winha Wille” -rajapinnan toteutuksen (rajapinta käytännössä edellyttää ONTin lähdekoodin muokkaamista) sekä asennuksen ja oikean konfiguroinnin web-palvelimelle. Asennusta ja konfigurointia varten olen laatinut järjestelmäkuvauksen (Liite 5, salainen), jossa kuvaillaan mm. järjestelmän keskeisiä ominaisuuksia ja sitä mitä asioita asennuksessa ja konfiguroinnissa tulee ottaa huomioon.

Sähköpostikuittauksia varten tehty toiminnallisuus on olemassa ONTissa, mutta sitä ei ole vielä sovellettu mihinkään järjestelmän käyttötapaukseen. Jatkokehitysvaiheessa voisi miettiä sitä mihin käyttötapauksiin sähköpostitoiminnallisuutta sovelletaan.

ONTiin tallennetut opinnäytetyöt jäävät järjestelmään ja ajan kanssa ne jäävät koordinaattorin ja ohjaajien näkymiin. Tätä varten pitäisi toteuttaa Arkisto–toiminnallisuus, jonka avulla päättyneet opinnäytetyöt voidaan pitää erillään aktiivisista

opinnäytetöistä. Näin saadaan koordinaattorin ja ohjaajien näkymät mahdollisimman selkeiksi ja järjestelmän suorituskyky pysyy hyvänä.

ONTissa on asioita, joita voi jatkokehittää. Jatkokehitystehtäviä varten tarvitaan HAAGA-HELIAn henkilöstöstä joku, joka osaa koodata C#:lla ja jolla on kokemusta sovelluskehityksestä ja mm. Winha Willen rakenteesta ja toiminnallisuuksista. Myös itse olen valmis jatkokehittää järjestelmää erillisellä sopimuksella.

4.4 Oma ammatillinen kasvu

Minulla on kokopäiväinen ansiotyö, jossa kehitän suurta ASP.NET sovellusta julkishallinnon puolella. Minulle on ASP.NET kehitys tästä syystä varsin tuttua. Olen ollut sovelluskehittäjänä yli 10 vuotta ja .NET kehittäjänä olen ollut yli 4 vuotta. Tämän opinnäytetyön tekeminen sovelluskehityksen näkökulmasta toi jotain uutta ammatillista osaamista ja se samalla vahvisti jo sitä osaamista, jota minulla on. Jossain kohdin kehitystä oivalsin uusia malleja ja koodirakenteita, joista voisi olla myös hyötyä työelämässä.

Teknisessä mielessä Active Directory:n tuntemus on vahvistunut tämän opinnäytetyön jälkeen valtavasti, koska olen joutunut tutustumaan aiheeseen perin pohjin.

Kaikkein eniten olen kasvanut ammatillisesti projektinhallinnan suhteen. En usko, että jatkan sovelluskehittäjänä eläkepäiviin asti, ja varmasti jossain vaiheessa joudun enemmän tai vähemmän olemaan tekemisissä projektinhallinta rutiinien kanssa. Tosin, tällä hetkellä olen osa projektia ja tuen projektiryhmää, mutta projektin vetäminen on aivan eri juttu ja vaatii paljon paperityötä. Mielestäni opinnäytetyö on aivan erityisellä tavalla kasvattanut minua siinä miten projekti viedään läpi alusta loppuun ja antaa esimakua työelämään.

Lähteet

Bean Software. Classic ASP vs. ASP.NET. Luettavissa: http://www.beansoftware.com/ASP.NET-Tutorials/Classic-ASP-vs-ASP.NET.aspx Luettu 23.9.2011

Microsoft 2011. What is ASP.NET Web Forms? Luettavissa: http://www.asp.net/web-forms/what-is-web-forms. Luettu 23.9.2011

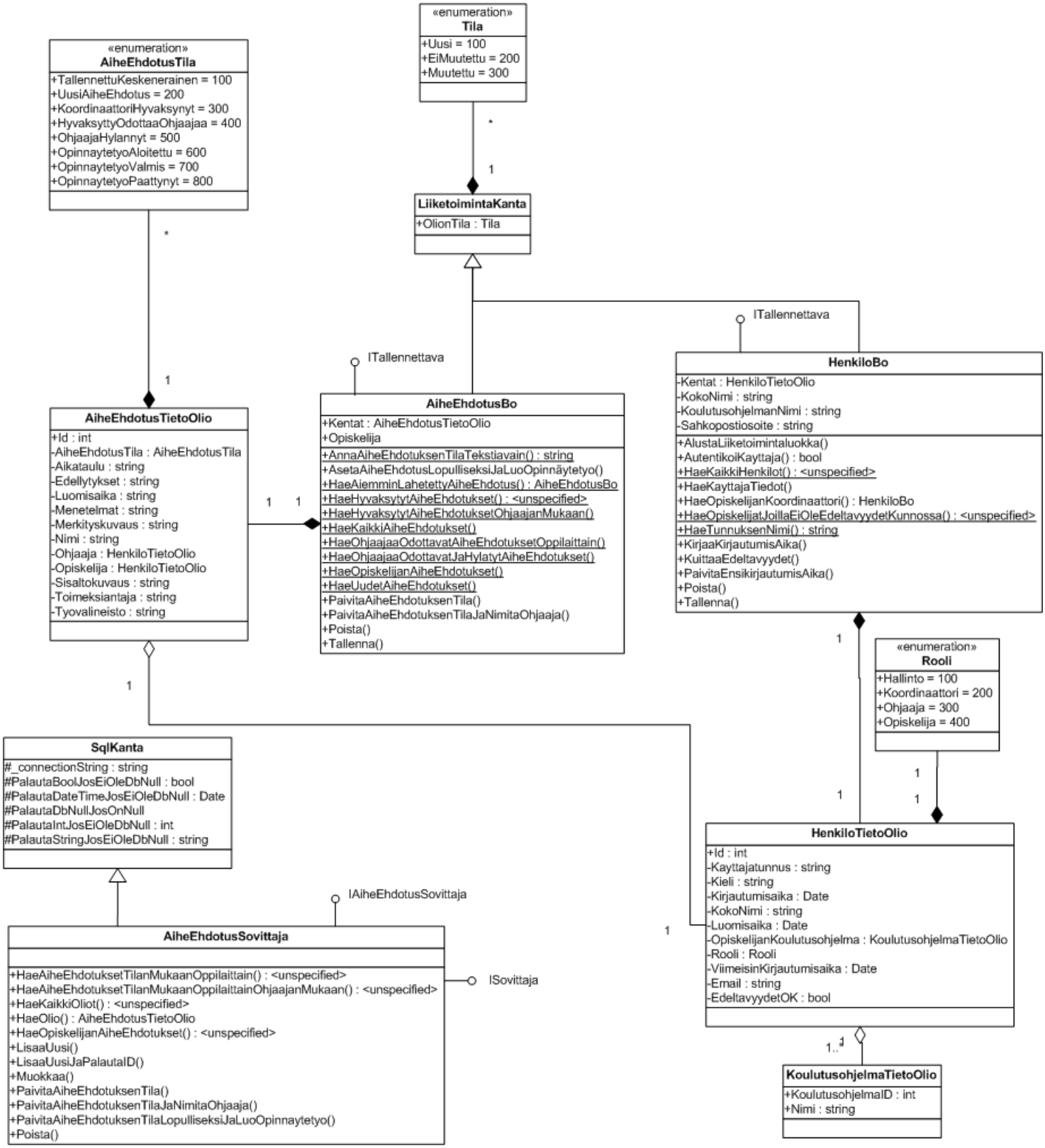
Microsoft 2011. ASP.NET Page Life Cycle Overview. Luettavissa: http://msdn.microsoft.com/en-us/library/ms178472.aspx. Luettu 23.9.2011

Microsoft 2011. ADO.NET Overview. Luettavissa: http://msdn.microsoft.com/en-us/library/h43ks021(v=VS.90).aspx. Luettu 23.9.2011

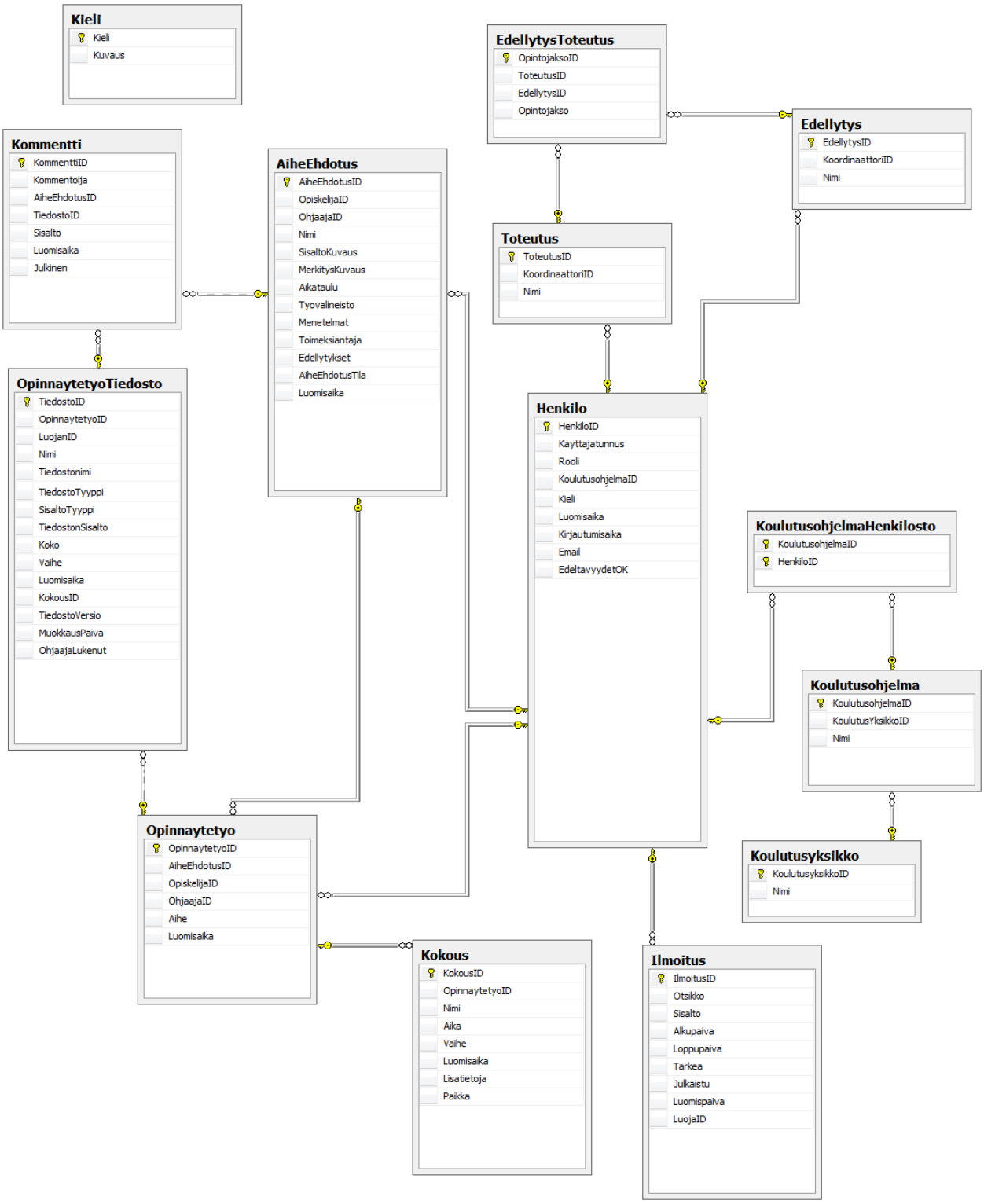
Pinal Dave 2007. SQL SERVER – Stored Procedures Advantages and Best Advantage. Luettavissa: http://blog.sqlauthority.com/2007/04/13/sql-server-stored-procedures-advantages-and-best-advantage/. Luettu 23.9.2011

Liitteet

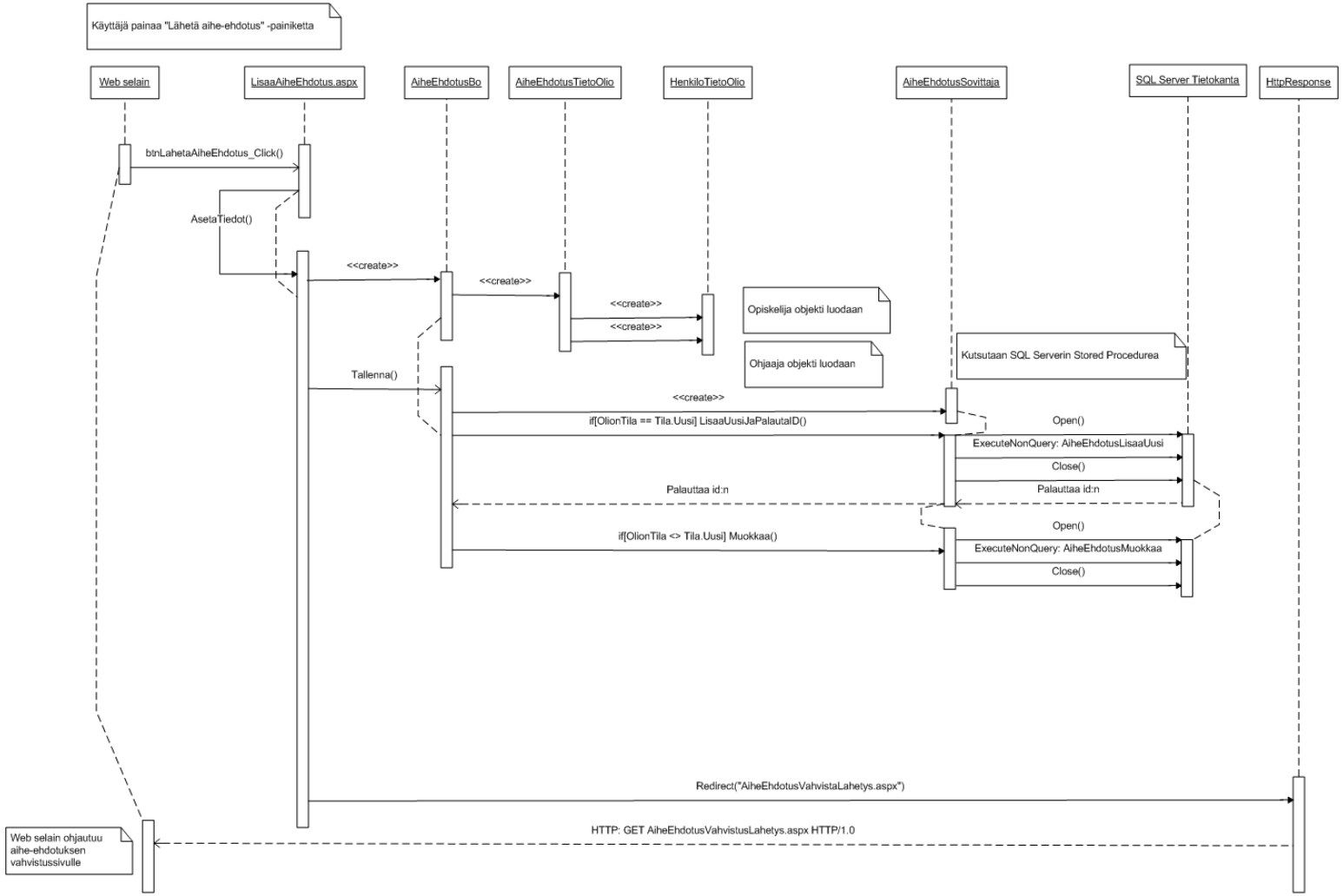
Liite 1. Luokkakaavio



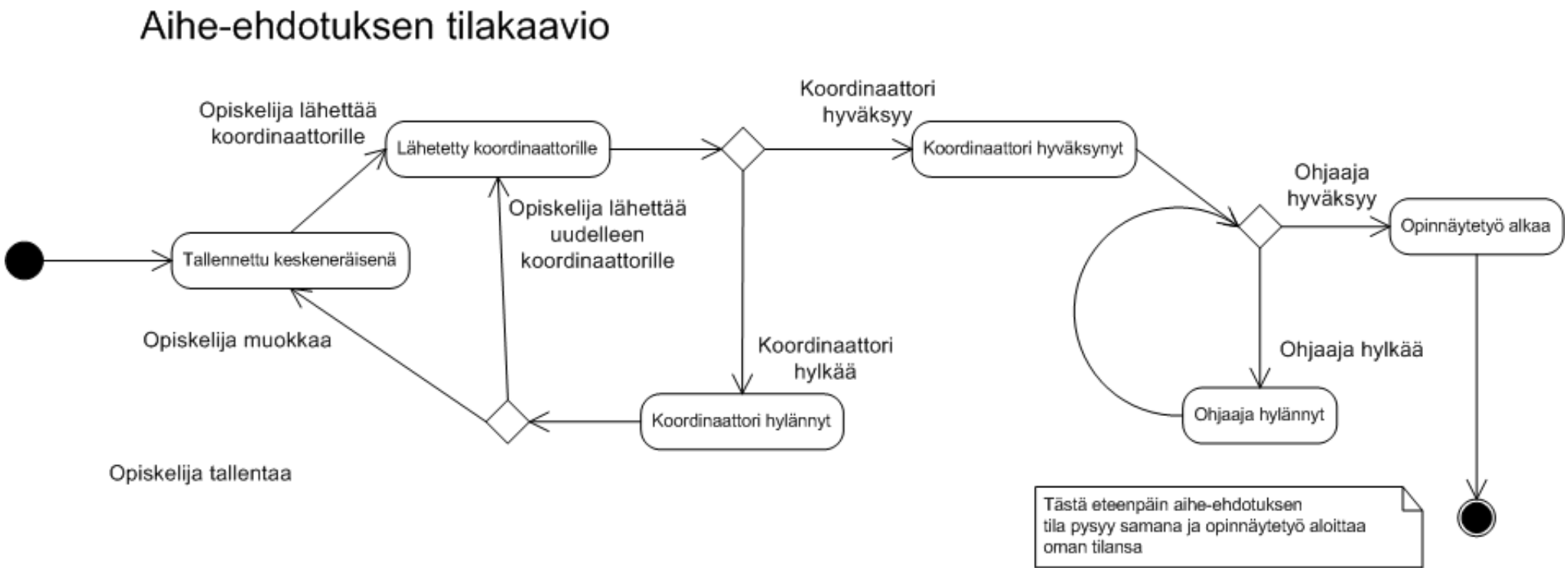
Liite 2. Relaatiokaavio



Liite 3. Viestiyhteyskaavio



Liite 4. Tilakaavio



Liite 5. Järjestelmäkuvaus

Liite on salainen.

Liite 6. Loppuraportti