

Ville Sälliluoma

## OPINNÄYTETYÖN ARVIOINTITYÖKALUN TOTEUTUS

Tietojenkäsittelyn koulutusohjelma  
2013

# OPINNÄYTETYÖN ARVIOINTITYÖKALUN TOTEUTUS

Sälliluoma, Ville  
Satakunnan ammattikorkeakoulu  
Tietojenkäsittelyn koulutusohjelma  
Helmikuu 2013  
Ohjaaja: Nieminen, Hans  
Sivumäärä: 51  
Liitteitä: 0

Asiasanat: ASP.NET MVC, arviointityökalu, toteutus

---

Opinnäytetyön toimeksiantaja oli Satakunnan ammattikorkeakoulu. Asiakasyrityksellä oli tarve uudistaa olemassa olevaa opinnäytetyön arviointityökalua. Lähtökohdat olivat tarve dynaamisemmalle työkalulle, jolla on kyky virtaviivaistaa opinnäytetyöstä annettavan lausunnon valmistusprosessia, sekä epävarmuus olemassa olevan arviointityökalun käyttöympäristön tulevaisuudesta.

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa uusi opinnäytetyön arviointityökalu, joka vastaa asiakasyrityksen tarpeisiin. Opinnäytetyössä käsitellään web-palveluissa käytettäviä tekniikoita ja niiden soveltamista järjestelmän toteuttamisessa. Näkökulmana huomioidaan myös suunnittelumalleja.

Järjestelmästä tehtiin Internet-sovellus, jota käytetään web-selaimella. Järjestelmä toteutettiin pääosin ASP.NET MVC -tekniikalla, jota opiskelin itsenäisesti. Ajax, jQuery, Entity Framework ja LINQ to Entities olivat minulle myös tuntemattomia tekniikoita. Projektilla oli myös ylläpitävä ja kehittävä vaikutus aiemmin opittuihin tekniikoihin.

## IMPLEMENTATION OF THE THESIS EVALUATION TOOL

Sälliluoma, Ville

Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences

Degree Programme in Business Information Systems

February 2013

Supervisor: Nieminen, Hans

Number of pages: 51

Appendices: 0

Keywords: ASP.NET MVC, evaluation tool, implementation

---

The employer of this thesis was Satakunta University of Applied Sciences. The client needed to remodel the existing thesis evaluation tool. There was a need for more dynamic tool with the ability to streamline the process of giving a verdict on theses and the future of the existing operating environment of the thesis evaluation tools was uncertain.

The purpose of this thesis was to design and implement a new thesis evaluation tool to respond to the client's needs. This thesis covers technologies used in web services and their utilization in system development. Software design patterns have also been covered.

A web browser based Internet application was implemented of the system. The system was primarily developed with ASP.NET MVC technology which I have been learning on my own. Ajax, jQuery, Entity Framework and LINQ to Entities were also unknown technologies to me. This project also had a maintaining and educating effect on techniques which had been included in studies.

## SISÄLLYS

1	JOHDANTO.....	6
2	VANHAN JÄRJESTELMÄN KUVAUS .....	7
2.1	Järjestelmän periaate .....	7
2.2	Järjestelmän käyttö.....	7
2.3	Puutteet ja ongelmat.....	9
3	TOTEUTUKSESSA KÄYTETTÄVÄT TEKNIIKAT.....	10
3.1	ASP.NET .....	11
3.2	ASP.NET MVC .....	11
3.2.1	Komponentit.....	12
3.2.2	Kerrokset.....	12
3.2.3	Konfigurointi.....	14
3.2.4	Reititys.....	14
3.3	ADO.NET Entity Framework.....	15
3.4	LINQ to Entities.....	15
3.5	NuGet.....	16
3.6	HTML .....	16
3.7	CSS-tyylisivukieli .....	17
3.8	JavaScript.....	18
3.9	jQuery .....	19
3.10	jQuery UI .....	20
3.11	Ajax.....	20
4	UUDEN JÄRJESTELMÄN SUUNNITTELUVAIHE.....	22
4.1	Tietokannat .....	22
4.1.1	Käyttäjätietokanta.....	22
4.1.2	Lausunto/kriteeritietokanta.....	23
4.1.3	Virhelokitietokanta.....	25
4.2	Käyttöliittymän kuvaus .....	25
4.2.1	Peruskäyttäjän käyttöliittymä .....	26
4.2.2	Pääkäyttäjän käyttöliittymä .....	29
5	JÄRJESTELMÄN TOTEUTUS JA TESTAUS .....	32
5.1	Tiedonsaantitaso .....	32
5.1.1	Käyttöliittymäluokka.....	32
5.1.2	Repository-luokka .....	33
5.1.3	Testidataluokka.....	34
5.2	Käyttäjän syötteiden validointi palvelimessa.....	34
5.3	Käyttäjän syötteiden validointi selaimessa .....	36

5.4	Tekstieditori .....	37
5.5	PDF-toiminto .....	38
5.6	Käyttäjähallinta ja tietoturva.....	38
5.6.1	Käyttäjäroolit.....	38
5.6.2	Forms-autentikointi .....	39
5.6.3	Autorisointi .....	40
5.6.4	Käyttäjän yksityisyydensuoja .....	41
5.6.5	Sivustoon kohdistuvat uhat.....	43
5.7	Julkaisutoimenpiteet .....	45
5.7.1	Code First -migraatiot lausunto/kriteeritietokannalle.....	45
5.7.2	Visual Studio Publish-toiminto .....	46
5.8	Järjestelmän testaus.....	47
6	JOHTOPÄÄTÖKSET .....	48
	LÄHTEET .....	49

# 1 JOHDANTO

Opinnäytetyön aiheena on opinnäytetyön arviointityökalun toteuttaminen. Asiakasyrityksenä toimii Satakunnan ammattikorkeakoulu, joka on monialainen, kansainvälisesti suuntautunut ja opiskelijamäärältään Suomen kahdeksanneksi suurin korkeakoulu. Koulutusta on neljällä eri paikkakunnalla: Porissa, Raumalla, Huittisissa ja Kankaanpäässä.

Asiakasyritys on minulle organisaationa tuttu – muun opiskelun lisäksi suoritin opintoihin sisältyvän työharjoittelun asiakasyrityksen Porin yksikössä. Opinnäytetyön aiheen ideoin työharjoitteluni yhteydessä.

Työn lähtökohtana oli asiakasyrityksen tarve uudistaa olemassa olevaa opinnäytetyön arviointityökalua. Uudistustarpeen laukaiseva tekijä oli epävarmuus siitä, että luovutaanko järjestelmästä, johon nykyinen opinnäytetyön arviointityökalu on rakennettu. Uudelta järjestelmältä edellytetään dynaamisempaa näkökulmaa ja kykyä virtaviivaistaa opinnäytetyöstä annettavan lausunnon valmistusprosessia. Työn tavoite on suunnitella ja toteuttaa web-selaimella käytettävä järjestelmä, joka vastaa asiakasyrityksen tarpeisiin.

Työn painopiste on suunnittelussa sekä toteutuksessa käytettävien tekniikoiden esittelyssä ja niiden soveltamisessa järjestelmän ydinosa-alueiden toteuttamiseksi. Työn luonne on enemmän tuotekehitysprojekti kuin tutkimus. Toteutuksessa otetaan kuitenkin huomioon suunnittelumalleja ja hyväksi havaittuja ratkaisuja.

Työn määrittelyosuudessa kuvataan vanhan järjestelmän toiminta sekä puutteet ja ongelmat. Toteutuksessa käytettäviä tekniikoita havainnollistetaan valmiin järjestelmän ratkaisuille. Suunnittelu- ja toteutusosuuksissa kytetään teoriaa käytäntöön syvällisemmin.

## 2 VANHAN JÄRJESTELMÄN KUVAUS

### 2.1 Järjestelmän periaate

Opinnäytetyön arviointityökalun tarkoitus on auttaa opettajia kirjallisten lausuntojen antamisessa opinnäytetöistä. Lausunto luodaan järjestelmään kovakoodattujen kriteerilauseiden avulla, mutta käyttäjä voi syöttää myös vapaamuotoisia kommentteja.

Satakunnan ammattikorkeakoulussa lausunnot arkistoidaan Tweb-asiakirjanhallintajärjestelmään PDF-muotoisina. Käyttäjä voi kopioida lausunnon kolmannen osapuolen sovellukseen, tehdä tarvittavia muokkauksia ja lopulta muodostaa valmiin PDF-dokumentin.

Opinnäytetyön arviointityökalun keskeiset ominaisuudet ovat lausuntojen käyttäjäkohtainen hallinta ja opinnäytetyön arvosanan ehdottaminen osioiden painotettuna keskiarvona. Arviointikategorioita ovat perus ja ylempi amk-tutkinto ja näiden englanninkieliset vastineet.

### 2.2 Järjestelmän käyttö

Opinnäytetyön arviointityökalun käyttöönotto tapahtuu kirjautumalla verkkopöytätyökalusta Virtualiaan, johon arviointityökalu on rakennettu. Järjestelmä tunnistaa käyttäjän Windows-todennuksella.

Käyttäjä voi luoda lausunnon haluamaansa kategoriaan. Lausunto luodaan web-lomakkeella (Kuva 1), jossa opinnäytetyön tiedot annetaan syötteinä.

[Etusivulle](#)

#### Opinnäytetyön arviointi - AMK-tutkinto

Opiskelijanumero:	<input type="text"/>
Opinnäytetyön tekijä:	<input type="text"/>
Opinnäytetyön nimi:	<input type="text"/>
Työn arvioija:	Ville Sälliluoma
Jos työllä on toinen ohjaaja:	<input type="text"/>
Yksikkö:	<valitse toimipiste>
Arviointipäivä:	28.12.2012

Kuva 1. Opinnäytetyön tiedot syötetään tekstikenttiin. Työn arvioijaksi muodostetaan tunnistettu käyttäjä. Päivämääräksi muodostetaan kuluva päiväys.

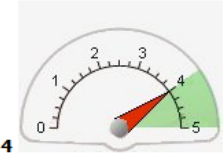
Varsinainen lausuntoteksti luodaan valintanapeilla, joiden arvot edustavat kriteerilauseita (Kuva 2). Kriteerilauseiden yhteydessä on määritelty arvosanat 0-5 sekä osioiden painokertoimet.

**Arvioinnin perusteena on käytetty seuraavia kriteereitä:**

<b>I Aiheen valinta ja lähestymistapa:</b> <b>10 %</b>	<input type="radio"/> 5 Opiskelija osoittaa aiheen olevan ajankohtainen, selkeästi työelämän tarpeeseen vastaava ja liittyvän opiskeltavaan alaan. Opiskelija asettaa työssään selkeät ja rajatut tavoitteet, ongelmat ja tehtävät ja kuvaa kehittämis- ja tutkimusmenetelmät jätäl työtavat perusteellisesti tukeutuen luotettaviin lähteisiin.
	<input type="radio"/> 4 Opiskelija osoittaa aiheen olevan työelämän tarpeeseen vastaava ja liittyvän opiskeltavaan alaan. Opiskelija asettaa työssään selkeät ja rajatut tavoitteet, ongelmat ja tehtävät ja kuvaa kehittämis- ja tutkimusmenetelmät jätäl työtavat tukeutuen luotettaviin lähteisiin.

Kuva 2. Kriteerilauseet.

Lopuksi käyttäjä voi kirjoittaa haluamansa lisäkommentit, antaa arvosanan ja tallentaa lausunnon. Järjestelmä laskee valintanapeilla annettujen arvojen perusteella osioiden painotetun keskiarvon (Kuva 3).

<b>Opinnäytetyön laskennallinen painotettu arvosana:</b> 3,60 =====>	
---	--

Kuva 3. Painotetun keskiarvon laskuri.

Lausuntoon sisällytetään kategoriakohtaiset tekstit, käyttäjän antamat opinnäytetyön tiedot, kriteerilauseet, lisäkommentit ja arvosana. Kuvassa 4 on esitetty valmis lausunto.



## SATAKUNNAN AMMATTIKORKEAKOULU

## ARVIOINTILAUSUNTO OPINNÄYTETYÖSTÄ

22.01.2013

Opinnäytetyön tekijä: Malli Opiskelija  
Opiskelijanumero: 123456

Opinnäytetyön nimi: Testi

Opinnäytetyön ohjaaja: Ville Sälliluoma

Tämä on opinnäytetyön arviointilausunto SAMK:n opiskelijan, **Malli Opiskelija**, opinnäytetyöstä **Testi**.

Lausunnon on antanut opinnäytetyön ohjaaja: Ville Sälliluoma

Opiskelija osoittaa aiheen liittyvän opiskeltavaan alaan. Hän asettaa työssään tavoitteet, ongelmat ja tehtävät, jotka vaativat täsmennystä ja kuvaa kehittämis- ja tutkimusmenetelmät ja/tai työtavat tukeutuen niukasti lähteisiin.

Opiskelija kuvaa valitsemaansa menetelmiä, mutta niiden käyttö on horjuvaa ja virheellistä. Työssä kuvataan aineiston keruu tai toimintatapa/työprosessi vain osittain eikä opiskelija perustele valintojaan.

Lisäkommentit

Opinnäytetyön arvosana: 3 Hyvä.

#### Kuva 4. Valmis lausunto.

Lausunnot tallennetaan kategoriakohtaisesti. Valitussa kategoriassa on hallintavalikko, josta pääsee lausuntojen muokkaus- ja poistotoimintoihin. Käyttäjä näkee vain oman materiaalin. Kuvassa 5 on esitetty lausuntojen hallintavalikko.

### Opinnäytetyön arviointi - AMK-opinnäytetyö

arviointipäivä	opinnäytetyö tekijä	ohjaaja
<a href="#">22.01.2013</a>	Testi	Malli Opiskelija Ville Sälliluoma
<a href="#">28.12.2012</a>		Ville Sälliluoma
<a href="#">28.12.2012</a>		Ville Sälliluoma

#### Kuva 5. Lausuntojen hallintavalikko.

### 2.3 Puutteet ja ongelmat

Järjestelmässä on muun muassa seuraavia ongelmia ja puutteita:

- 1 Käyttäjän syötteisiin liittyvät ongelmat
  - Käyttäjän syötteitä ei varmenneta ja niiden ehdollisuutta ei ole noteerattu.

- Käyttäjä ei voi vaikuttaa kaikkiin tietokenttiin. Esimerkiksi päivämääräksi muodostuu aina kuluva päivä – myös muokkausten yhteydessä.
- 2 Lausuntojen hallintaan liittyvät ongelmat
- Järjestelmässä ei ole lausuntojen etsintätoimintoa.
  - Lausuntojen järjestykseen ei voida vaikuttaa. Kategoriakohtaiset lausunnot järjestetään päivämäärän mukaisesti – ongelmaksi muodostuu päivämäärän käyttäytyminen.
- 3 Puuttuvat kokonaisuudet
- Järjestelmällä ei voi luoda PDF-dokumenttia.
  - Järjestelmällä ei voi tehdä tekstinkäsittelyä.
  - Järjestelmässä ei ole web-selaimella toteutettavaa ylläpitoa.

Järjestelmässä on myös epäloogisuuksia ja navigointiongelmia. Esimerkiksi onnistuneen kirjautumisen jälkeen käyttäjä ohjataan perus amk-tutkinto -kategorian uuden lausunnon luonti -sivulle, jossa on linkki etusivulle. Kyseinen linkki vie lausuntojen hallintavalikko -sivulle, jossa on varsinaisen etusivun linkki. Toinen mainittava esimerkki on, että järjestelmässä noteerataan arvosana 0, vaikka Satakunnan ammattikorkeakoulun linjauksen mukaan opiskelijalle ei tehdä lausuntoa ennen kuin kriteerit johtavat vähintään arvosanaan 1.

### 3 TOTEUTUKSESSA KÄYTETTÄVÄT TEKNIIKAT

Projekti toteutettiin Microsoftin Visual Studio 2010 -tuotteella, joka on täyden palvelun ohjelmistokehitysovellus. Toteutusympäristönä toimi ASP.NET MVC (versio 3). Raportin kaikki MVC-ympäristöä koskevat asiat edustavat kyseistä versiota.

Pääasiallinen ohjelmointikieli oli C#, jolla toteutettiin kaikki ASP.NET MVC:n komponentit. Kaikki kehitystyö tehtiin kotitietokoneella paikallisessa ympäristössä (localhost).

### 3.1 ASP.NET

ASP.NET (Active Server Pages for .NET) on Microsoftin kehittämä web-tuotantoalusta, jolla voidaan toteuttaa dynaamisia web-palveluja. Ohjelmointi tapahtuu nykyaikaisilla olio-ohjelmointikielillä (C# ja Visual Basic .NET). Nykyinen ASP.NET muodostuu kolmesta web-sovellusten luontiin tarkoitettusta tekniikasta: ASP.NET Web Forms, ASP.NET MVC ja ASP.NET Web Pages. (Nieminen 2012.)

### 3.2 ASP.NET MVC

ASP.NET MVC -tekniikka toteuttaa MVC-ohjelmistoarkkitehtuuria, joka perustuu norjalaisen Trygve Reenskaugin vuonna 1979 kehittämään malliin. MVC-arkkitehtuurin kantava ajatus on erottaa käyttöliittymä sovelluslogiikasta ja -datasta. Lähtökohtana on, että käyttöliittymä ilmaisee sovelluksen tilaa ja näyttää sen tarvittaessa erilaisissa näkymissä. (Bellinaso, Berardi & Katawazi 2009, 7; Koskimies & Mikkonen 2005, 142.)

MVC-suunnittelumalli (Model-View-Controller) erottaa sovelluksen kolmeen eri komponenttiin: Model (malli), View (näkymä) ja Controller (ohjain). MVC-suunnittelumalli määrittää, miten eri ohjelmointilogiikat sijoittuvat sovelluksessa. Jaottelu auttaa hallitsemaan monimutkaisia sovelluksia, koska on mahdollista keskittyä yhteen osa-alueeseen kerralla. Myös yksikkötestaus helpottuu, koska jokainen osa voidaan testata muista riippumattomasti. (Freeman & Sanderson 2011, 8; Microsoft 2012.)

ASP.NET MVC -arkkitehtuurilla saavutetaan myös muun muassa seuraavia etuja:

- Tukee kaikkia ASP.NET-ydinominaisuuksia, esimerkiksi: välimuistia ja käyttäjien autentikointia.
  - Ei sisällä viewstate- ja postback-toimintoja. Viewstate-toiminnon puuttuminen vaikuttaa muun muassa siten että, web-sivut ovat tyypillisesti pienemmän kokoisia tilavaatimukseltaan kuin Web Forms -tekniikalla toteutetut.
  - Käyttäjällä on hyvät mahdollisuudet vaikuttaa syntyvään HTML:ään.
- (Bellinaso ym. 2009, 11.)

### 3.2.1 Komponentit

Malli sisältää kaiken toiminta-, validointi- ja tiedonsaantilogiikan, joita sovellus tarvitsee. Model-oliot käyttävät tietokantaa. Esimerkiksi tuoteolio saattaa hakea tietoa tietokannasta, käsitellä sitä ja kirjoittaa päivitetyn tiedon tuotteet-tauluun SQL-palvelimelle. Pienissä sovelluksissa malli on usein käsitteellinen. (Microsoft 2012; Walther 2010, 120.)

Näkymät ovat komponentteja, jotka näyttävät sovelluksen käyttöliittymän. Tyypillisesti käyttöliittymä luodaan mallin datan pohjalta. Esimerkiksi tuote-aulun muokausnäyttö, joka esittää kontrolleja, kuten drop-down list, checkbox ja textbox. (Microsoft 2012.)

ASP.NET MVC 3 versiossa esiteltiin Razor-näkymämoottori (View Engine). Razorin tuomia uudistuksia ovat muun muassa: syntaksia on yksinkertaistettu verrattuna vanhempaan Web Forms -näkymämoottoriin sekä mahdollisuus luoda Partial View -näkymiä. Partial View on varsinaiseen näkymään upotettavaa kokonaisuus, jota voidaan käyttää useammassa näkymässä. (Allen, Galloway, Haack & Wilson 2011, 50; Nieminen 2012.)

Ohjain on komponentti, joka käsittelee käyttäjän syötteet, työskentelee mallin kanssa ja valitsee käyttöliittymän esittävän näkymän esitettäväksi. Ohjain esimerkiksi käsittelee merkkijonoja ja välittää ne mallille, joka tekee tietokantakyselyjä kyseisillä arvoilla. (Microsoft 2012.)

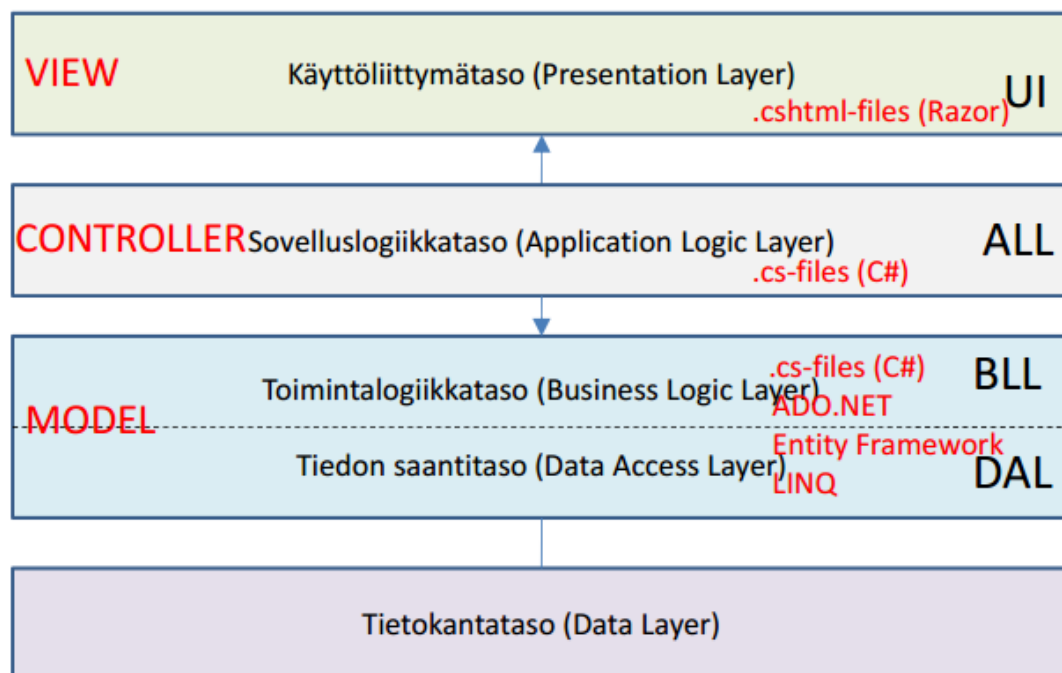
### 3.2.2 Kerrokset

ASP.NET MVC on kerrosarkkitehtuuri, joka voidaan jakaa viiteen tasoon (Kuva 6):

- 1 tiedonsaantitaso (Data Access Layer)
  - Koodi käsittelee tietokantatasolla olevaa raakadataa. Edustaa mallia.
- 2 toimintalogiikkataso (Business Logic Layer)
  - Koodi lisää toimintasäännöt ja domain-kohtaiset oliot. Edustaa mallia.
- 3 sovelluslogiikkataso (Application Logic Layer)

- Koodi huolehtii käyttöliittymätason ja toimintalogiikkatason vuorovaikutuksesta. Edustaa ohjainta.
- 4 käyttöliittymätaso (Presentation Layer)
  - Koodi määrittelee, mitä käyttäjä näkee näytöltä. Edustaa näkymää.
- 5 tietokantataso (Data Layer)
  - Yleensä relaatiotietokanta, mutta voi myös olla XML- tai tekstitiedosto.

(Bellinaso ym. 2009, 64–65.)



Kuva 6. ASP.NET MVC:n tasot ja, mitä MVC-komponenttia ne edustavat (Nieminen 2012).

Kerrosarkkitehtuuri tarkoittaa ohjelmiston rakennetta, joka koostuu loogisesti yhte-  
naisistä eri käsitetasoilla olevista kerroksista siten, että ylempi kerros käyttää alem-  
man kerroksen palveluja. Käsitetasot voidaan nähdä skaalassa laite/ihminen: laite-  
päässä olevat tasot ovat matalammassa tasossa kuin ihmisistä lähellä olevat tasot.  
Alemmat kerrokset tarjoavat laitetta tai käyttöjärjestelmää lähellä olevia toimintoja.  
Ylemmät kerrokset tarjoavat esimerkiksi graafisen käyttöliittymän palveluita. (Kos-  
kimies 2000, 212; Koskimies & Mikkonen 2005, 126.)

### 3.2.3 Konfigurointi

XML-tiedostoilla tehtävä ASP.NET-konfigurointi perustuu hierarkkiseen rakenteeseen, jossa asetukset muodostetaan lomittamalla tiedot sisimmästä solmukohdasta lähtien. Windows-asennuskansion Microsoft.NET\Framework\ .NET (versionumero)\Config -polussa on Web.config-tiedosto, joka sisältää oletusarvot ASP.NET-sovellusten omille Web.config-tiedostoille. (Nieminen 2012.)

ASP.NET MVC -sovelluksessa Web.config-tiedosto on juurikansion lisäksi myös Views-kansiossa. Juurikansion Web.config-tiedostossa määritellään sovelluksen yleisasetukset, jotka ovat voimassa, mikäli alikansion tiedostossa ei tehdä ylikirjoitusta. (Galloway 2012.)

### 3.2.4 Reititys

ASP.NET MVC -tekniikassa reititysmekanismin kantava ajatus on, että URL-osoite ei ole sidoksissa fyysiseen tiedostoon web-palvelimella. Sen sijaan URL-osoite on sidoksissa ohjaimeen. URL-pyyntöön saapuessa tutkitaan reititystaulusta, mikä ohjain vastaa URL-pyyntöä. Reititystiedot luodaan Global.asax.cs-tiedostoon. Oletuksena tiedostossa on RegisterRoutes-metodi, joka noudattaa seuraavaa reitityskaavaa: controller/action/id. (Bogard, Hexter, Hinze, Palermo & Skinner 2012, 154; Nieminen 2012.) URL-muoto on siis esimerkiksi /Etusivu/Index (Kuva 7).

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // URL with parameters
        new { controller = "Etusivu", action = "Index", id = UrlParameter.Optional }
    );
}
```

Kuva 7. Reitityksen määrittely.

### 3.3 ADO.NET Entity Framework

Entity Framework on Microsoftin kehittämä tekniikka, jonka avulla tiedonsaanti (Data Access) voidaan saavuttaa ohjelmoimalla käsitteellinen EDM (Entity Data Model) -sovellusmalli. Näin ollen sovelluslogiikan ei tarvitse olla kytköksissä suoraan tietokannan skeeman kanssa. Tämä vähentää tarvittavan koodin määrää ja ylläpitoa. (Liu 2010, 171.)

EDM-sovellusmalli voidaan toteuttaa kolmella eri menetelmällä:

#### 1 Database First

- Entity Framework pystyy automaattisesti luomaan luokista ja ominaisuuksista koostuvan sovellusmallin olemassa olevasta tietokannasta. Tiedot tietokannan skeemasta, sovellusmallista ja näiden välisestä kuvauksesta tallennetaan XML-tiedostoksi. Visual Studio -sovelluksessa on graafinen Entity Framework -suunnittelutyökalu, jolla voi käsitellä kyseistä tiedostoa.

#### 2 Model First

- Jos ei ole valmista tietokantaa, sovellusmalli voidaan luoda edellä mainitulla suunnittelutyökalulla. Kun sovellusmalli on valmis, suunnittelutyökalu toteuttaa tarvittavat DDL (Data Definition Language) -lauseet tietokannan luomiseksi. Tämä lähestymistapa käyttää myös XML-tiedostoa.

#### 3 Code First

- Tässä lähestymistavassa ohjelmoidaan itse luokat ja ominaisuudet, joita Entity Framework käyttää. Näin ollen järjestelmä ei luo XML-tiedostoa. Jos valmista tietokantaa ei ole, Entity Framework voi luoda sen kyseisistä luokista. Tietokannan skeeman ja sovellusmallin välinen kuvaus toteutetaan käyttöliittymäluokalla. Sovellusmallin muuttuessa tietokanta voidaan poistaa ja luoda uudelleen. (Dykstra 2012, 12–13.)

### 3.4 LINQ to Entities

LINQ to Entities on Entity Frameworkin yleisin tekniikka tietokantakyselyille. Kantava ajatus on, että ohjelmoijan ei tarvitse kirjoittaa matalan tason tietokantakäsitte-

lyä (esimerkiksi tietokantayhteys ja SQL-parametrit). LINQ-kyselyt tehdään EDM-mallin olioille. Entity Framework tekee tarvittavat SQL-käännökset, jotka heijastuvat konkreettiseen tietokantaan. (Barrett 2009; Nieminen, 2012.) Kuvassa 8 tehdään LINQ-kysely kriteeri-aululle.

```
public IQueryable<Kriteeri> KriteeriHaeKaikki()
{
    return _db.Kriteerit;
}
```

Kuva 8. LINQ-kysely kriteeri-aululle. Kysely palauttaa taulun kaikki alkiot.

### 3.5 NuGet

NuGet on Visual Studion laajennuksena toimiva paketinhallintajärjestelmä. NuGet kunnostautuu etenkin kolmansien osapuolien ohjelmistoratkaisujen levityskanavana .NET-ympäristössä. (NuGet 2012.)

Kun NuGet-paketti lisätään sovellukseen, järjestelmä kopioi tiedostot relevantteihin kohteisiin ja toteuttaa tarvittavat muutokset esimerkiksi Web.config-tiedostoon. NuGet-paketin poistaminen vastaavasti kumoaa tehdyt muutokset. (Bogard ym. 2012, 233.)

### 3.6 HTML

HTML (HyperText Markup Language) on kuvauskieli, jolla määritellään web-sivun rakenne ja sisältö. Web-sivupyynnö toteutetaan HTTP-protokollalla. Sivupyynnöprosessi on seuraava: web-selain lähettää HTTP-pyynnön (request) palvelimelle. Palvelin hakee pyydetyn sivun HTML-koodauksen ja lähettää sen selaimelle HTTP-vastauksena (response). Selain muodostaa HTML-koodista web-sivun. (Boehm & Ruvalcaba 2012, 37.)

HTML 5 on konseptin uusin versio, joka käytännössä korvaa aiemmat. Kantava ajatus on, että mikä tahansa HTML-variaatio toimii myös HTML 5 -dokumenttina.



HTML 5 sisältää paljon uudistuksia, esimerkiksi uusia elementtejä käyttöliittymän luomiseen, syötteiden validoimiseen, dynaamiseen grafiikkaan ja rikkaan median sovelluksiin. (Cook & Garber 2012, 6–7; Nieminen 2012.)

HTML-dokumentin perusrakenne muodostuu dokumenttityypin (DOCTYPE) määrittelystä ja dokumenttipuusta. Dokumenttipuussa sijaitsevat elementit, joilla web-sivun rakenne määritellään. Puunjuuri-elementti on <html>-tagi, jonka sisälle tehdään muut määrittelyt. HTML 5 -dokumentissa elementit voidaan kirjoittaa isoilla ja pienillä kirjaimilla ja näiden yhdistelmällä (Boehm & Ruvalcaba 2012, 42.)

### 3.7 CSS-tyylisivukieli

CSS (Cascading Style Sheets) -tyylisivukielen kantava ajatus on erottaa HTML-elementtien ulkoasun määrittely omaksi kokonaisuudekseen – HTML määrittää sivuston rakenteen, jolle CSS tuottaa visuaaliseen ilmeen. Yleisin ja tehokkain tapa on käyttää CSS-tyylejä erillisestä tiedostosta. Tällä lähestymistavalla etuina ovat muun muassa: HTML-koodi on helposti luettavaa sekä ylläpidettävää ja yhdessä paikassa tehtävät muutokset heijastuvat tarvittaessa koko sivustoon. (Grannell, Sumner & Synodinos 2012, 9; Powers 2012, 3.)

Uusin CSS-standardi on CSS3, joka on edelleen kehitysvaiheessa. Standardin kehitys perustuu useamman moduulin kokonaisuuteen yksittäisen spesifikaation sijaan. Kunkin moduuli käsittää tietyn osa-alueen, esimerkiksi fontit. Moduulit pohjautuvat aiempaan CSS2.1-spesifikaatioon, jota ne täydentävät. Konseptin tarkoitus on, että selainvalmistajat voivat vaiheittain ottaa CSS3-ominaisuuksia käyttöön. (Bradford ym. 2012, 2; Frain 2012, 24.)

CSS3 sisältää muun muassa seuraavia uudistuksia:

- pyöristetyt kulmat
  - tekstin ja elementtien varjot
  - useamman taustakuvan tuki
  - elementtien läpinäkyvyys (voidaan määritellä RGB-väriarvon kanssa).
- (Cederholm 2010, 6–8.)

CSS-syntaksi on kuvaavaa ja helposti opittavissa. Tosin tehokas hyödyntäminen vaatii paljon kokemusta. Kuvassa 9 on opinnäytetyön arviointityökalun pääotsikon määrittely CSS-tyylitiedostossa.

```
#header h1
{
  font-family: helvetica;
  font-weight: bold;
  color: #FFFFFF;
  line-height: 2em;
  font-size: 40px !important;
  text-shadow: rgb(51,51,51) 0px 4px 4px;  /* CSS3 */
}
```

Kuva 9. Opinnäytetyön arviointityökalu -otsikon tyylimäärittely Site.css-tiedostossa. Tekstin varjostus (text-shadow) on CSS3-ominaisuus.

### 3.8 JavaScript

JavaScript on skriptikieli, jolla voidaan muun muassa lisätä web-sivuille dynaamisia toimintoja, esimerkiksi hiiren käyttöön reagoivia ulkoasuelementtejä ja lomakkeiden validointitoimintoja. Avainasiana on käyttäjän ja sivuston välinen välitön vuorovaikutus. (Sawyer McFarland 2008, 1.) Kuvassa 10 on esitetty JavaScript-funktio, jolla kriteerilause viedään tekstieditoriin.

JavaScript on tulkattava kieli, mikä tarkoittaa, että sitä ei käännetä ennen ajamista. JavaScriptiä käytetään pääosin selainten sisällä, mutta sillä voidaan myös kirjoittaa ohjelmia, joita ajetaan palvelimien sisällä. JavaScriptiä voidaan kirjoittaa suoraan HTML-dokumenttiin <script>-tagin sisälle tai käyttää ulkoisesta tiedostosta. (Boumbhrey ym. 2000, 446.)

JavaScriptin kieliopillisia ominaisuuksia ovat muun muassa:

- Vakiotietotyyppejä, joita ovat esimerkiksi merkkijonot, numerot ja loogiset vakiot (boolean), ei määritellä.
- Ei ole erillisiä tietotyyppejä kokonaisluvuille ja liukuluvuille. Numero (number) -tietotyyppi edustaa kaikkia lukuja.

- Funktiolle voidaan antaa mikä tahansa määrä argumentteja. (C. Zakas 2012, 83.)

```
function lisaa() {
    var t = ""
    for (i = 0; i < document.kriteeriform.arviot.length; i++) {
        if (document.kriteeriform.arviot[i].checked == true) {
            t = t + document.kriteeriform.arviot[i].value
        }
    }

    var sisalto = tinyMCE.get('LausuntoTeksti').getContent();
    var uusisisalto = sisalto += t
    tinyMCE.get('LausuntoTeksti').setContent(uusisisalto);
}
```

Kuva 10. JavaScript-funktio, joka vie kriteerilauseen tekstieditoriin. Funktiota kutsutaan lisää lausuntoon -painikkeen onclick-tapahtumassa.

### 3.9 jQuery

jQuery on suosittu JavaScript-kirjasto, joka tarjoaa yksinkertaistettuja ratkaisuja useisiin yleisiin käyttötapauksiin. Kantava ajatus on tarvittavan koodimäärän supistaminen – yhdellä koodirivillä voi saavuttaa tuloksen, joka muilla JavaScript-ratkaisulla vaatisi huomattavasti enemmän töitä. Konseptilla tähdätään myös koodin toimivuuden varmistamiseen eri selaimissa. (Sawyer McFarland 2012, 4.)

jQueryn muita etuja ovat muun muassa:

- avoin lähdekoodi
- pieni koko, varsinkin pakattuna versiona
- suuri kehittäjäyhteisö
- kattava dokumentointi. (Murach 2012, 197; Rutter 2011, 12.)

jQuery otetaan käyttöön lataamalla jQuery-kirjastot joko projektin sivuilta tai linkittämällä virallisen palveluntarjoajan, esimerkiksi Googlen, jakelukanavan (Content Delivery Network) URL-osoite web-järjestelmään. (Larsen & Otero 2012,4.) jQuery-kirjastot ovat saatavilla myös NuGet-pakettina.

### 3.10 jQuery UI

jQuery UI on jQuery:n lisäosa, joka on keskittynyt nimenomaan käyttöliittymän toimintoihin. jQuery UI tarjoaa erilaisia toimintoja rikkaan käyttöliittymän luomiseksi. Toiminnot jaetaan kolmeen kategoriaan: valmiisiin komponentteihin, efekteihin ja vuorovaikutteisiin toimintoihin. (Blakeley Silver 2010, 193–194.) Kuvassa 11 haetaan lausunnon päivämäärä interaktiivisesta kalenterista.

Valmiita jQuery UI -komponentteja ovat esimerkiksi:

- tekstikentän syötteiden automaattitäyttö
- interaktiivinen kalenteri
- välilehdet
- prosessipalkki. (Sarrion 2012, 2.)

**Arviointipäivä:**



Kuva 11. jQuery UI interaktiivinen kalenteri -komponentti (Nieminen 2012). Kalenteri ilmestyy, kun arviointipäivä-kenttää klikataan.

### 3.11 Ajax

Ajax (Asynchronous JavaScript and XML) on useamman tekniikan kombinaatio, jonka kantava ajatus on mahdollistaa web-sivun osittainen päivittäminen asynkroni-

sella tiedonsiirrolla selaimen ja palvelimen välillä. Tämä tarkoittaa, että tiedonsiirto-prosessi ei noudata perinteistä konseptia – postback/vastaus sykliä, jossa palvelin palauttaa kokonaisen HTML-sivun selaimelle. (Grieb, Moroney & Pars 2007, 7.)

Ajax-toteutuksessa keskeisessä roolissa on XMLHttpRequest-objekti, jolla toteutetaan selaimen ja palvelimen välinen kommunikaatio. Palvelin lähettää esimerkiksi XML-muotoista dataa selaimelle, jossa toiminnallisuus tuotetaan JavaScriptillä, ja datan esittäminen HTML ja CSS-tekniikoilla. (Holzner 2006, 11–12.)

Konseptilla saavutetaan muun muassa seuraavia etuja:

- Parantaa verkon ja palvelimen suorituskykyä, koska ei suoriteta täyttä postback-toimintoa, jolloin liikkuva datamäärä on pienempi.
- Parantaa käyttöliittymää, koska käyttäjä saa nopeammin vasteen toiminnalleen. (Cate, Glavich, McClure & Shoemaker 2006, 6.)

ASP.NET MVC -ympäristössä Ajax-toimintoja voidaan suorittaa muun muassa Ajax helper -metodeilla. Metodit mahdollistavat asynkronisen toiminnan ilman erillistä koodausta. Ajax helper -metodit ovat riippuvaisia jQuery-kirjastosta, joten niiden käyttö edellyttää viittausta jquery.unobtrusive-ajax -skriptiin. (Allen ym. 2011, 187.) Kuvassa 12 haetaan Ajax ActionLink -metodeilla sisältöä jQuery UI Menu -komponentin välilehtiin.

```
<div id="tabsyp">
  <ul>
    <li><a href="#tabs-1">OPINNÄYTETYÖN ARVIOINTI</a></li>
    <li>@Ajax.ActionLink("BACHELOR'S THESIS",
      "_IndexEng",
      new AjaxOptions
      {
        UpdateTargetId = "tabs-2",
        InsertionMode = InsertionMode.Replace,
        HttpMethod = "GET"
      })
    </li>
    <li>@Ajax.ActionLink("YLEMPI AMK-TUTKINTO",
      "_IndexYlempiAmk",
      new AjaxOptions
      {
        UpdateTargetId = "tabs-3",
        InsertionMode = InsertionMode.Replace,
        HttpMethod = "GET"
      })
    </li>
  </ul>
</div>
```

Kuva 12. Ylläpito-osion jQuery UI Menu -komponentin koodia. Välilehtiin haetaan sisältöä Ajax ActionLink -metodeilla.

## 4 UUDEN JÄRJESTELMÄN SUUNNITTELUVAIHE

Opinnäytetyön arviointityökalun suunnittelun lähtökohtana oli toteuttaa helppokäyttöinen, web-selaimella käytettävä sovellus, joka ratkaisee vanhan järjestelmän ongelmia. Suunnitteluprosessi sisälsi tietokantojen ja käyttöliittymän hahmottelua ja osittaista toteuttamista.

Tässä luvussa kuvataan järjestelmän tietokantojen luontia sekä rakennetta. Luvussa viisi käsitellään tarkemmin tietokantojen integroimista ja sovelluslogiikan ohjelmointia tietokantojen ja järjestelmän välillä. Käyttöliittymästä kuvataan muun muassa keskeinen rakenne ja miten tämä heijastuu perus- ja pääkäyttäjille. Havainnollistamisessa käytetään kuvia, jotka ovat valmiista järjestelmästä.

### 4.1 Tietokannat

Tietokantakäytäntöinä käytettiin Microsoftin SQL Server Express- ja SQL Server Compact Edition -tekniikoita. Hallintatyökaluna toimi SQL Server Management Studio Express. Opinnäytetyön arviointityökalu sisältää kolme eri tietokantaa seuraaville kokonaisuuksille: käyttäjät, lausunnot/kriteerit ja virheloki.

#### 4.1.1 Käyttäjätietokanta

Käyttäjätietokannassa on toiminnot muun muassa käyttäjätilien tallentamiseen, autentikoimiseen ja käyttäjäroolien hallintaan. Käyttäjätietokantana käytetään ASP.NET Membership -tietokantaa, joka luodaan SQL Server Express -instanssiin.

Windows-asennuskansion Microsoft.NET\Framework\\*.NET (versionumero) -polussa on asp\_net\_regsql.exe -ohjelma, jolla ASP.NET Membership -tietokanta asennetaan haluttuun SQL Server -instanssiin. Tietokannan dbo-skeemaan luodaan useita tietokantaobjekteja: tauluja, talletettuja proseduureja, näkymiä, skeemoja ja tietokantaroolleja. (Nieminen 2012).

ASP.NET sisältää valmiin Membership-rajapinnan. Membership-rajapinnan pääluokka on System.Web.Security.Membership, joka sisältää useita metodeja sekä ominaisuuksia käyttäjätilien hallintaan. (Bellinaso ym. 2009, 89.) Kyseistä rajapintaa käyttämällä toteutettiin toiminnallisuus sovelluksen ja käyttäjätietokannan välillä.

ASP.NET MVC sisältää myös valmiin pohjan (Internet Application template), jossa on paljon valmiita elementtejä käyttäjähallinnan toteuttamiseksi. Pohjaan luodaan muun muassa valmis käyttäjätili-ohjain. (Allen ym. 2011, 143.)

Tällä lähestymistavalla Visual Studiolla luotava sovellus käyttää oletusasetuksena SQL Membership provider -luokkaa, jolla otetaan yhteys SQL Server Express -instanssiin. SQL Server Express tukee user instance databases -ominaisuutta, jolloin ei tarvitse tehdä tietokannan asetuksia – avattu yhteys luo automaattisesti aspnetdb-nimisen ASP.NET Membership -tietokannan. (Bellinaso ym. 2009, 93; Freeman & Sanderson 2011, 740–741.)

Toteutuksessa kuitenkin päädyttiin siihen, että asennettiin itse halutun niminen tietokanta. Tällä ratkaisulla saavutti myös täyden kontrollin MVC-komponentteihin, jotka assosioituvat käyttäjätietokantaan.

#### 4.1.2 Lausunto/kriteeritietokanta

Tietokanta lausunnoille ja kriteereille toteutettiin Entity Framework Code First- tekniikalla. Tietokantakäytäntönä toimii SQL Server Compact Edition. Tietokanta on lähtökohtaisesti melko yksinkertainen, koska taulut eivät sisällä yhteyksiä muihin tauluihin. Tämä ei poissulje sitä, että saadaan assosioitua käyttäjän ja lausunnon välinen yhteys. Tästä kerrotaan tarkemmin luvussa viisi.

Tietokanta sisältää kahdeksan taulua – yhden lausunto- ja kriteeri-aulun arviointikategorialle kohden. Jokaisen kategorian taulut ovat rakenteeltaan identtisiä muiden kategorioiden kanssa. Taulujen sarakkeet kirjoitetaan ominaisuuksiksi (property) malliluokkiin.

Kriteeri-aulun tietoina ovat tunnus (id), otsikko sekä kriteerilauseet arvosanaa kohden. KriteeriID-ominaisuus toimii taulun perusavaimena. Oletusarvoisesti Entity Framework tulkitsee id-loppuiset ominaisuudet perusavaimiksi (Dijkstra 2012, 20). Kuvassa 13 on ohjelmoitu kriteeri-aulun sarakkeet.

```
public class Kriteeri
{
    public int KriteeriID { get; set; }
    public string Otsikko { get; set; }
    public string A5 { get; set; }
    public string A4 { get; set; }
    public string A3 { get; set; }
    public string A2 { get; set; }
    public string A1 { get; set; }
}
```

Kuva 13. Kriteeri-aulun sarakkeet.

Lausunto-aulun tietoina ovat tunnus, opinnäytetyön tiedot sekä arvosana, ohjaajan sekä mahdollisen toisen ohjaajan nimi, varsinainen lausuntoteksti, arviointiaika ja omistaja, joka on lausunnon assosiaatiosuhde käyttäjään. Kuvassa 14 on ohjelmoitu lausunto-aulun sarakkeet.

```
public partial class Lausunto
{
    public int LausuntoID { get; set; }
    public int? Arvio { get; set; }
    public string Omistaja { get; set; }
    public string OpiskelijaNro { get; set; }
    public string Tekija { get; set; }
    public string TyonNimi { get; set; }
    public string Ohjaaja { get; set; }
    public string ToinenOhjaaja { get; set; }
    public string LausuntoTeksti { get; set; }
    public DateTime Aika { get; set; }
}
```

Kuva 14. Lausunto-aulun sarakkeet.



#### 4.1.3 Virhelokitietokanta

Virhelokin kerääminen toteutettiin avoimen lähdekoodin ELMAH-sovelluksella, johon liittyvät komponentit haettiin NuGet-pakettina. ELMAH MVC -projektin sivuilla on DDL-skripti, jolla ELMAH-virhelokitietokanta pystytetään sekä ohjeet sovelluksen integroimiseksi (Beletsky, haettu 12.1.2012). Virhelokitietokanta toteutettiin SQL Server Express -instanssiin.

#### 4.2 Käyttöliittymän kuvaus

Opinnäytetyön arviointityökalun käyttöliittymän kantava ajatus on, että oleellisimmat päätoiminnot kuten sivuston navigointi ja käyttäjätinhallinta ovat aina saatavilla samassa muodossa. Lisäksi käyttäjälle tarjotaan tarvittavaa sisältöä sivukohtaisiin valikkolohkoihin. Kuvassa 15 on ulkoasun taittokaava.



Kuva 15. Ulkoasun taittokaava.

Käyttöliittymän suunnittelun lähtökohtana oli luoda helppokäyttöinen ja looginen käyttöliittymä, joka vastaa vanhan järjestelmän puutteisiin ja epäloogisuuksiin. Värimaailman suunnittelu perustuu Satakunnan ammattikorkeakoulun web-materiaaleissa käytettävään linjaukseen.

Järjestelmän käyttöliittymä toteutettiin pääasiallisesti CSS- ja jQuery UI -tekniikoilla. Käyttöliittymän ratkaisut, joihin päädyttiin, ovat helppoja toteuttaa ASP.NET MVC -ympäristössä.

Yhtenäisen ulkoasun toteutus perustuu \_Layout.cshtml-tiedostoon (Kuva 16), jossa on määritelty käytettävät CSS-tyylitiedostot, skriptit ja HTML-rakenne sivuston eri osa-alueille, esimerkiksi pääsisältö- ja navigointilohkoille. Kyseinen ulkoasutiedosto toimii kaavana näkymille, joissa voidaan määrittää haluttua sisältöä lohkoihin. Lohkojen esiintyminen voidaan myös ehdollistaa. (Allen ym. 2011, 60; Chadwick, Panda & Snyder 2012, 28.)

```
<div id="sivupalkki">
  <div id="logindisplay">
    <h4>Käyttäjätili</h4>

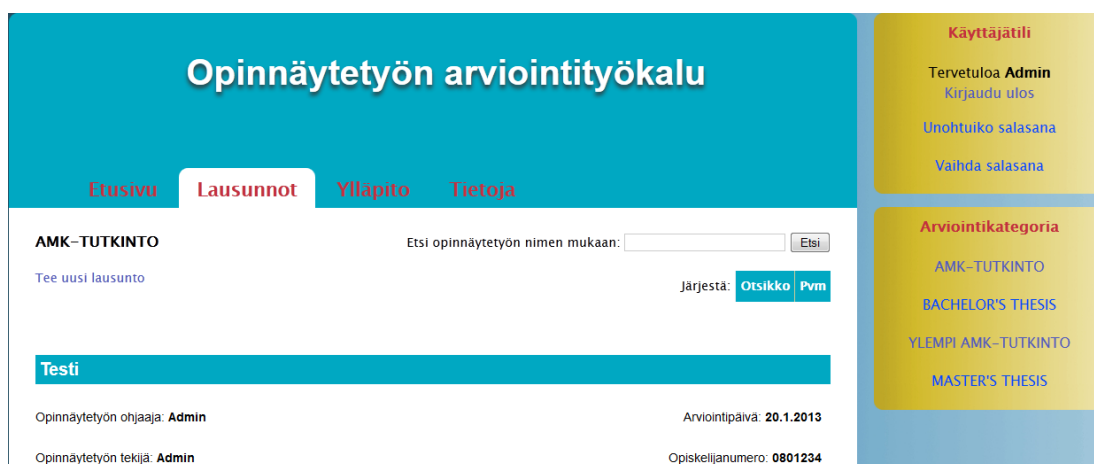
    @Html.Partial("_Kirjaus")
    <p>
      @Html.ActionLink("Unohtuiko salasana", "UnohtSalasana", "Kayttaja")
    </p>
    <p>
      @Html.ActionLink("Vaihda salasana", "MuutaSalasana", "Kayttaja")
    </p>
  </div>
</div>

<div id="sivupalkki2">
  @RenderSection("sidebar2", false)
</div>
```

Kuva 16. Ulkoasutiedoston koodia, jossa on määritelty sivulohkojen HTML-rakenne.

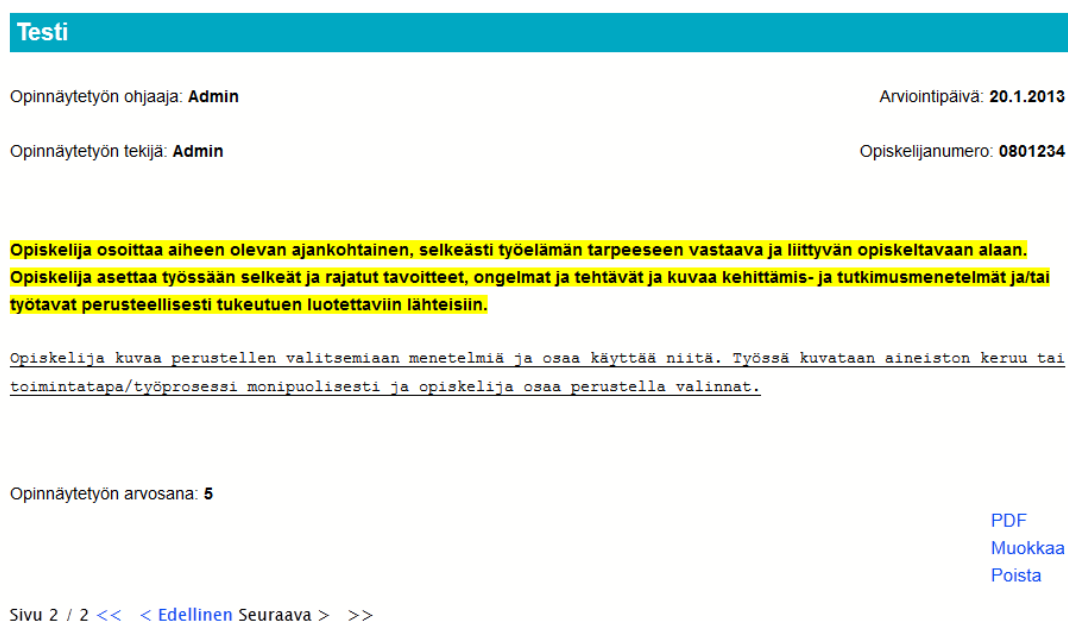
#### 4.2.1 Peruskäyttäjän käyttöliittymä

Peruskäyttäjä pääsee muihin päänavigointivalikon sivuihin paitsi ylläpitoon. Peruskäyttäjälle oleellisin sivu on lausunnot (Kuva 17), josta kontrolloidaan opinnäytetöistä annettavia lausuntoja. Sivulohkosta valitaan haluttu kategoria, johon lausunnot luodaan. Kategoriakohtaiset näkymät ovat toiminnaltaan identtisiä.



Kuva 17. Päänavigointivalikosta on valittu lausunnot-sivu.

Lausunnot-sivulle tallennetaan lausuntojen esikatselut (Kuva 18). Sivulla näytetään kolme esikatselua – kontekstin navigointitoiminto aktivoituu, jos niitä on enemmän. Muita lausunnot-sivun toimintoja ovat etsintä, järjestely sekä muokkaus-, poisto- ja PDF-linkit.



Kuva 18. Lausunnon esikatselu.



Web-lomakkeella toteutetaan myös painotetun keskiarvon laskeminen ja syötetään opinnäytetyön tiedot. Virheelliset syötteet noteerataan virheilmoitusviesteillä. Lausunnon esikatselun PDF-linkki vie näkymään, jossa lausunnosta luodaan PDF-tiedosto (Kuva 21).

SATAKUNNAN AMMATTIKORKEAKOULU

ARVIOINTILAUSUNTO OPINNÄYTETYÖSTÄ

20.1.2013

Opinnäytetyön tekijä: Admin

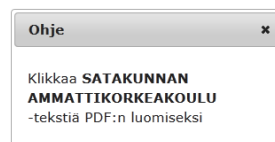
Opiskelijanumero: 0801234

Opinnäytetyön nimi: Testi

Opinnäytetyön ohjaaja: Admin

Tämä on opinnäytetyön arviointilausunto SAMK:n opiskelijan, Admin, opinnäytetyöstä.

Lausunnon on antanut opinnäytetyön ohjaaja: Admin



Opiskelija osoittaa aiheen olevan ajankohtainen, selkeästi työelämän tarpeeseen vastaava ja liittyvän opiskeltavaan alaan. Opiskelija asettaa työssään selkeät ja rajatut tavoitteet, ongelmat ja tehtävät ja kuvaa kehittämis- ja tutkimusmenetelmät ja/tai työtavat perusteellisesti tukeutuen luotettaviin lähteisiin.

Kuva 21. PDF-tiedoston luonti -näkö. Ohjeikkuna on toteutettu jQuery UI Dialog -komponentilla.

#### 4.2.2 Pääkäyttäjän käyttöliittymä

Pääkäyttäjän käyttöliittymällä hallinnoidaan tietokantojen tietoja. Pääkäyttäjän oleellisin sivu on ylläpito (Kuva 22), jossa voidaan kontrolloida järjestelmän kriteereitä, käyttäjätilejä ja virhelokia. Ylläpitosivun oletusnäkö on kriteerien kontrollointi. Sivulohkosta valitaan ylläpidon osa-alue.

##### Kriteerit

OPINNÄYTETYÖN ARVIOINTI		BACHELOR'S THESIS	YLEMPI AMK-TUTKINTO	MASTER'S THESIS
Tee uusi kriteeri				
Otsikko		Toiminnot		
Aiheen valinta ja lähestymistapa		<a href="#">Tiedot</a> <a href="#">Muokkaa</a> <a href="#">Poista</a>		



Kuva 22. Ylläpitosivun kriteerit-näkö.

Kriteerien kontrollointi tapahtuu jQuery UI Menu -komponentilla. Kriteeriotsikkoon assosioituvia toimintoja ovat muokkaus, tiedot poisto. Tiedot-toiminnolla näkee kriteeriotsikon käsittämät kriteerilauseet ja arvosanat. Tällä ratkaisulla saadaan kokonaisuus pienempään ja selkeämpään tilaan.

Käyttäjähallinnan toimintoja ovat muun muassa käyttäjän etsintä, käyttötilin poisto, käyttäjän salasanan vaihto ja käyttäjäroolienhallinta. Kuvassa 23 on käyttäjähallinnan päävalikko.

**Käyttäjähallinta**

Etsi käyttäjä (tunnus tai sähköpostiosoite):

**Käyttäjät**

- Admin**  
[admin@dot.net](#)  
**Paikalla**  
Pääkäyttäjä. Tässä voidaan esittää käyttäjäkohtaisia kommentteja.
- Kayttaja**  
[kayttaja@dot.com](#)  
Poissa 1 päivä.

< Edellinen
1
Seuraava >

**Roolit**

- [admins](#)
- [users](#)

Kuva 23. Käyttäjähallinnan päävalikko.

Päävalikon käyttäjien- ja roolien nimet edustavat linkkejä, jotka johtavat niihin assosioituviin näkymiin. Kuvassa 24 on esitetty käyttäjätiedot-näkymä.

## Käyttäjätiedot: Admin [Poissa]

- [Päävalikko](#)
- [Roolitus](#)
- [Salasana](#)

### Käyttäjätili

Käyttäjänimi: Admin

Sähköpostiosoite: [admin@dot.net](mailto:admin@dot.net)

Viimeksi aktiivinen: 20, tammikuu, 2013 10:46:12

Viimeksi kirjautunut: 20, tammikuu, 2013 10:46:12

Luotu: 01, lokakuu, 2012 14:10:12

Poista tili tietokannasta

### Sähköpostiosoite ja kommentit

Sähköpostiosoite:

Kommentit:

Pääkäyttäjä. Tässä  
voidaan esittää

Kuva 24. Käyttäjätiedot-näkymä. Ylälaidan navigointivalikosta pääsee muihin toimintoihin, jotka assosioituvat käyttäjään – navigointilogiikka pysyy samana.

Virhelokissa (Kuva 25) käyttöliittymä on englanninkielinen. Virhelokin toimintoja ovat muun muassa näytettävien virheilmoitusten määrän kontrollointi ja lokin konvertointi CSV-tiedostoksi.

## Error Log for OpnTesti on -PC

Host	Code	Type	Error	User	Date	Time
-PC	404	Http	The controller for path '/OpnTesti/bar' was not found or does not implement IController. <a href="#">Details...</a>	admin	24.1.2013	14:01
PC	404	Http	The controller for path '/OpnTesti/foo' was not found or does not implement IController. <a href="#">Details...</a>	admin	24.1.2013	14:01

Kuva 25. Virheloki.

## 5 JÄRJESTELMÄN TOTEUTUS JA TESTAUS

### 5.1 Tiedonsaantitaso

Opinnäytetyön arviointityökalun tiedonsaantitaso (Data Access Layer) käsittää seuraavat luokat:

- Käyttöliittymäluokka, joka toimii tietokantakäytävänä.
- Repository-luokka, jonka kautta käytetään Entity Frameworkin luomia oliokokoelmia.
- Testidataluokka, jolla tietokantaan voidaan luoda muun muassa kriteerilauseet testaustarkoitukseen.

#### 5.1.1 Käyttöliittymäluokka

Käyttöliittymäluokka periytetään System.Data.Entity.DbContext-luokasta. Luokassa määritellään DbSet<T>-tietotyyppiset ominaisuudet. T edustaa mallia, jonka Entity Framework assosioi tietokantataulun riveiksi. Ominaisuuden nimi edustaa kohdetaulua. (Allen ym. 2011, 75; Freeman & Sanderson 2011, 174.)

Opinnäytetyön arviointityökalussa käyttöliittymäluokka (Kuva 26) sisältää kahdeksan lausunto- ja kriteeri-malleista johdettua ominaisuutta. Mukana on myös metodi, jolla vältetään taulunimien monikkomuodot. Entity Framework johtaisi taulunimen ominaisuuden nimestä, mutta tällä lähestymistavalla nimi johdetaan mallin nimestä (Dykstra 2012, 23).

```
public class OpnArviointiEntities : DbContext
{
    public DbSet<Kriteeri> Kriteerit { get; set; }
    public DbSet<Criterion> Criteria { get; set; }
    public DbSet<KriteeriYlempiAmk> KriteeritYlempiAmk { get; set; }
    public DbSet<CriterionYlempiAmk> CriteriaYlempiAmk { get; set; }
    public DbSet<Lausunto> Lausunnot { get; set; }
    public DbSet<LausuntoEng> LausunnotEng { get; set; }
    public DbSet<LausuntoYlempiAmk> LausunnotYlempiAmk { get; set; }
    public DbSet<LausuntoYlempiAmkEng> LausunnotYlempiAmkEng { get; set; }

    //Taulunimet eivät generoidu monikollisiksi
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    }
}
```



Kuva 26. Entity Framework Code First -käyttöliittymäluokka.

Jotta tietokantayhteys toimii, Web.config-tiedostoon määritellään yhteysmerkkijono (connection string) (Kuva 27). Entity Framework tunnistaa tietokannan name-attribuutin avulla, joten on oleellista, että sen arvo on sama kuin käyttöliittymäluokan nimi (Freeman & Sanderson 2011, 174–175).

```
<connectionStrings>
  <add name="ApplicationServices"
    connectionString="Data source=.\SQLEXPRESS;Initial Catalog=OpnArviointi;Integrated Security=True"
    providerName="System.Data.SqlClient" />
  <add name="OpnArviointiEntities"
    connectionString="Data Source=|DataDirectory|Opn.sdf"
    providerName="System.Data.SqlClient" />
  <add name="elmah"
    connectionString="Data source=.\SQLEXPRESS;Initial Catalog=elmah;Integrated Security=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

Kuva 27. Konfiguraatio järjestelmän tietokantayhteyksien muodostamiseksi.

### 5.1.2 Repository-luokka

Repository edustaa suunnittelumallia, jonka kantava ajatus on, että sovelluslogiikkataso (ohjaimet) olisivat mahdollisimman vähän kytköksissä tiedonsaantitason kanssa – ohjainten koodi on täten yksinkertaisempaa. Muita etuja ovat muun muassa: tiedonsaantilogiikkaa on helpompi muokata myöhemmin ja yksikkötestaus helpottuu. (Conery, Guthrie, Haack & Hanselman 2009, 26.)

Repository-luokassa on opinnäytetyön arviointityökalun LINQ to Entities -metodit, joita käytetään ohjain-luokissa. Konseptia on hyödynnetty muun muassa seuraavasti: kuvan 8 esimerkissä haettiin Repository-luokasta kaikki kriteeri-taulun alkiot LINQ-kyselyllä. Kriteeri-ohjaimessa kutsutaan kyseistä metodia ja välitetään data kriteerit-näkymälle, joka on Partial View -tyyppinen. Kriteerit-näkymässä käsitellään ohjaimen välittämä data – tehdään kontekstin oliolle foreach-toistosilmukka jQuery UI Accordion -komponentissa. Ratkaisun tuotos on esitetty kuvassa 19.

### 5.1.3 Testidataluokka

Entity Framework mahdollistaa tietokannan poiston ja uudelleenluomisen jokaisella sovelluksen käynnistyskerralla. Entity Framework vertaa olemassa olevaa tietokantaa EDM-sovellusmalliin. Jos sovellusmalliin on tehty muutoksia, tietokanta poistetaan ja luodaan uudelleen. Voidaan myös määritellä, että tässä yhteydessä tietokantaan kirjoitetaan testidataa. (Lerman & Miller 2012, 28 ja 149.)

Opinnäytetyön arviointityökalussa kyseistä ominaisuutta on hyödynnetty etenkin kriteerien osalta. Kuvan 28 esimerkissä luodaan edellä mainitulla periaatteella uusi kriteeri ylempi amk-tutkinto -kategoriaan.

```
public class EsimerkkiData : DropCreateDatabaseIfModelChanges<OpnArviointiEntities>
{
    protected override void Seed(OpnArviointiEntities context)
    {
        var yamk = new List<KriteeriYlempiAmk>
        {
            new KriteeriYlempiAmk { Otsikko = "Ulkoasu",
                                   A5 = "Työ on huolitellusti aseteltu ja tarkoituksenmukaisesti havainnollistettu kokonaisuus.",
                                   A4 = "Työ on siisti ja opinnäytetyön ohjeen mukainen.",
                                   A3 = "Ulkoasu on pääosin opinnäytetyön ohjeen mukainen.",
                                   A2 = "Ulkoasussa on paljon poikkeamia opinnäytetyön ohjeesta.",
                                   A1 = "Ulkoasu on viimeistelemätön ja opinnäytetyön ohjetta ei ole noudatettu."}
        };

        yamk.ForEach(s => context.KriteeriYlempiAmk.Add(s));
        context.SaveChanges();
    }
}
```

Kuva 28. Testidatan luonti -metodi.

Tämän lisäksi pitää määritellä, että sovelluksen käynnistyessä ajetaan Entity Framework -metodi, joka laukaisee kyseisen prosessin. Tämä määrittely tehdään Global.asax.cs-luokan Application\_Start -metodiin. (Dykstra 2012, 26–27.)

## 5.2 Käyttäjän syötteiden validointi palvelimessa

Palvelinpuolella toteutettava käyttäjän syötteiden validointi perustuu ajatukseen, että käyttäjän syötteiden oikeellisuuteen ei kannata koskaan luottaa. Vaikka validointia toteutettaisiin myös selaimen puolella, käyttäjä voi esimerkiksi kytkeä JavaScriptin pois päältä tai tehdä odottamattoman toiminnon, johon selainpuolen validointi ei reagoi. (Allen ym. 2011, 117; Bogard ym. 2012, 93.)

ASP.NET sisältää Data Annotations -rajapinnan käyttäjän syötteiden validoimiseen. Lähtökohtana on, että malliin voidaan lisätä erilaisia attribuutteja, joilla validointi toteutetaan. Mikäli käyttäjä rikkoo validointisääntöjä, esitetään näkymän Validation-Summary helper -metodissa Data Annotations -rajapinnan luomat virheilmoitukset englanninkielellä. Virheilmoitusviestiin voidaan vaikuttaa ErrorMessage-ominaisuudella. (Allen ym. 2011, 101; Chadwick ym. 2012, 63–64.) Kuvassa 29 on esitetty lausunto-mallin ominaisuuksia, joihin on lisätty validointilogiikkaa.

```
public partial class Lausunto
{
    public int LausuntoID { get; set; }

    //int? jotta ei ole oletusarvoa ja voidaan jättää tyhjäksi.
    [Range(1, 5, ErrorMessage = "Arvioinnin on oltava kokonaisluku väliltä 1-5")]
    [RegularExpression("\\d", ErrorMessage = "Arvioinnin on oltava kokonaisluku väliltä 1-5")]
    [DisplayName("Arviointi (jätä tyhjäksi, jos harkinnassa):")]
    public int? Arvio { get; set; }

    //String koska opiskelijanumero voi alkaa 0:lla.
    [Required(ErrorMessage = "Pakollinen")]
    [DisplayName("Opiskelijanumero:")]
    [StringLength(30, ErrorMessage = "Opiskelijanumero ei voi olla suurempi kuin 30 merkkiä")]
    public string OpiskelijaNro { get; set; }

    [DataType(DataType.MultilineText)]
    [DisplayName("Lausunto:")]
    [MaxLength]
    [AllowHtml]
    public string LausuntoTeksti { get; set; }

    [Required(ErrorMessage = "Pakollinen")]
    [DataType(DataType.Date)]
    [DisplayName("Arviointipäivä:")]
    public DateTime Aika { get; set; }
}
```

Kuva 29. Lausunto-mallin validointilogiikkaa.

Validointi noteerataan ohjainten Action-metodien suoritusprosessissa. ASP.NET MVC Framework populoi ModelState-olion, joka välitetään ohjaimelle. Ohjain reagoi kyseiseen olioon ja tekee tarvittavat toimenpiteet, esimerkiksi tallentaa validin datan tietokantaan, tai palauttaa käyttäjän takaisin näkymään, jossa korjataan syötteiden virheet. (Chadwick ym. 2012, 62.) Kuvan 30 esimerkissä lausunto-ohjain käsittelee uuden lausunnon syötteet kyseisellä logiikalla.

```

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Uusi(Lausunto lausunto)
{
    try
    {
        if (ModelState.IsValid)
        {
            //Merkitään omistajaksi
            lausunto.Omistaja = User.Identity.Name;

            rp.LausuntoUusi(lausunto);
            return RedirectToAction("Index");
        }
        else
        {
            return View(lausunto);
        }
    }
    catch
    {
        return View();
    }
}

```

Kuva 30. Lausunto-ohjaimen Action-metodi (HTTP POST), jolla luodaan uusi lausunto.

### 5.3 Käyttäjän syötteiden validointi selaimessa

Selainpuolen validoinnin kantava ajatus on tarjota palvelinpuolen validointia nopeampi lähestymistapa, jossa käyttäjälle näytetään välittömästi mahdolliset virheilmoitukset. Selainpuolen validointi perustuu jQuery Validate -kirjastoon ja on oletusasetuksena päällä ASP.NET MVC -projektissa, mutta siihen voidaan vaikuttaa Web.config-tiedostossa. Selainpuolen validointi osaa hyödyntää malleihin määriteltyjä Data Annotations -attribuutteja, joten käyttöönotto on helppoa. (Chadwick ym. 2012, 79–81.)

Näkymille määritellään skriptit, jotka viittaavat jQuery Validate- ja jQuery ydin -kirjastoihin (Bogard ym. 2012, 99). jQuery ydin -kirjaston viite kannattaa määritellä \_Layout.cshtml-tiedostossa, jotta viittaus periytyy kaikkiin näkymiin. Kuvassa 31. on uuden lausunnon luonti -näkymän validointimäärittelyä.

```

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)
    <fieldset>
    <legend>Tiedot (pakolliset merkitty punaisella)</legend>
    <div class="editor-label-required">
        @Html.LabelFor(model => model.OpiskelijaNro)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.OpiskelijaNro)
        @Html.ValidationMessageFor(model => model.OpiskelijaNro)
    </div>
}

```

Kuva 31. Uuden lausunnon luonti -näkymän validointimäärittelyä. Muun muassa opiskelijanumero-kenttään assosioituu validointiviesti.

#### 5.4 Tekstieditori

Opinnäytetyön arviointityökalun tekstieditorina toimii Moxiecode Systemsin TinyMCE-sovellus. TinyMCE on avoimen lähdekoodin JavaScript-pohjainen WYSIWYG (What You See Is What You Get) -editori, jolla voidaan muuttaa muun muassa HTML text area -kentät editori-instansseiksi (Moxiecode Systems 2012). Tekstieditorin komponentit ladattiin NuGet-pakettina. Ohjeet integrointiin löytyvät projektin sivuilta.

Toteutuksessa luotiin helper-metodi (Kuva 32), jossa suoritetaan JavaScript-koodi, mikä tuottaa tekstieditorin. Konfigurointi toteutetaan metodin arvoja muuttamalla. Metodia kutsutaan näkymissä, joissa editoria käytetään.

```

@helper WYSIWYG()
{
    <script type="text/javascript" src="/Scripts/tinymce/tiny_mce.js"></script>
    <script type="text/javascript">
        tinyMCE.init({
            // General options
            mode: "textareas",
            language: "fi",
            theme: "advanced",
            plugins: "autolink,lists,style,layer,table,advhr,inlinepopups,insertdatetime,

            // Theme options

```

Kuva 32. Osa metodista, jolla tekstieditori tuotetaan.

## 5.5 PDF-toiminto

Opinnäytetyön arviointityökalun PDF-toiminto toteutettiin avoimen lähdekoodin Rotativa-ratkaisulla. Käyttöönotto tapahtuu seuraavasti: asennetaan Rotativa NuGet -paketti ja tehdään ohjaimeen Action-metodi, joka palauttaa konvertoitavan näkymän ActionAsPdf-metodin argumenttina (Bozio 2012). Kuvassa 33 on esitetty opinnäytetyön arviointityökalun Action-metodi, jolla konvertoidaan valmis lausunto PDF-tiedostoksi.

```
public ActionResult PrintLausunto(int id)
{
    return new ActionAsPdf(
        "_Tiedot",
        new { id = id }) { FileName = "Lausunto_opinnaytetyon tekija_opinnaytetyon nimi, koulutusohjelma, vuosiluku.pdf" };
}
```

Kuva 33. Lausunto-ohjaimen Action-metodi, jolla tuotetaan PDF-tiedosto. Metodin tarvitsema argumentti (id-tunnus) välitetään kontekstin näkymästä.

## 5.6 Käyttäjähallinta ja tietoturva

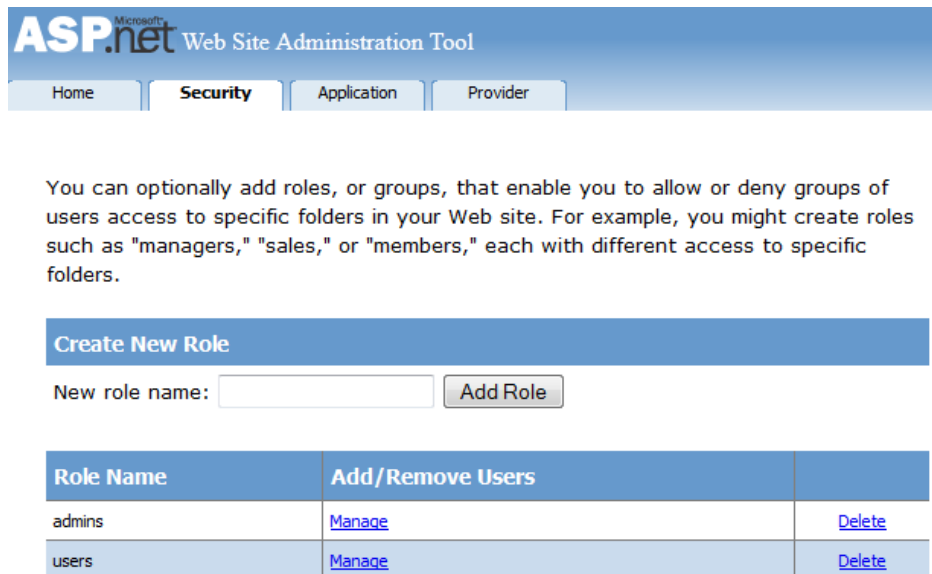
Opinnäytetyön arviointityökalussa on monipuoliset käyttäjähallinnan toiminnot. Myös tietoturva-asiat on noteerattu sekä käyttäjän yksityisyydensuojan että sivustoon kohdistuvien ulkoisten uhkien näkökulmista.

### 5.6.1 Käyttäjäroolit

Käyttäjäroolien kantava ajatus on keskittää käyttäjät ryhmiin, joille voidaan määritellä käyttöoikeuksia. Näin ollen jokaiselle käyttäjälle ei tarvitse erikseen määritellä, mihin toimintoihin voi vaikuttaa järjestelmässä. (Bellinaso ym. 2009, 100.) Käyttäjähallinnan toteutusprosessi kannattaa aloittaa testikäyttäjien- ja roolien tekemisellä, mikä on helppoa ASP.NET MVC -ympäristössä.

Opinnäytetyön arviointityökalu sisältää kaksi erilaista roolia: perus- ja pääkäyttäjät (users ja admins). Roolit luotiin Visual Studion Web Site Administration Tool

(WAT)- työkalulla (Kuva 34). WAT on web-pohjainen käyttöliittymä Membership-rajapinnan hallintaan (Freeman & Sanderson 2011, 743).



ASP.NET Web Site Administration Tool

Home Security Application Provider

You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders.

**Create New Role**

New role name:

Role Name	Add/Remove Users	
admins	<a href="#">Manage</a>	<a href="#">Delete</a>
users	<a href="#">Manage</a>	<a href="#">Delete</a>

Kuva 34. WAT -työkalu, jolla luotiin kaksi roolia.

Jatkossa käyttäjärooleihin ja käyttäjätileihin liittyvät toiminnot toteutettiin ohjelmallisesti. Ajatuksena on, että lopullisessa tuotantoympäristössä kaikki kontekstin toiminta voidaan toteuttaa sovelluksen omalla käyttöliittymällä.

### 5.6.2 Forms-autentikointi

Autentikointi tarkoittaa käyttäjän tunnistamista käyttäjänimen ja salasanan perusteella. ASP.NET MVC -sovelluksessa oletus autentikointitapa on Forms-autentikointi, jota käytetään Internet-sovelluksissa. Toinen vaihtoehto on Windows-autentikointi, jonka käyttökohde on Intranet-sovellukset. Autentikointitapa määritellään Web.config-tiedostossa. (Nieminen, 2012.) Opinnäytetyön arviointityökalussa käytetään Forms-autentikointia, jonka asetukset on määritelty kuvassa 35.

```
<authentication mode="Forms">
  <forms loginUrl="~/Kayttaja/Sisaan" timeout="2880" requireSSL="false" />
</authentication>
```

Kuva 35. Forms-autentikoinnin määrittely, jossa kerrotaan muun muassa, että sisäänkirjautumiseen käytettävän sivun URL-osoite on /Kayttaja/Sisaan.

Forms-autentikoinnin turvallisuus perustuu salakirjoitettuun evästeeseen (Freeman & Sanderson 2011, 734). Esimerkiksi käyttäjän rekisteröintiprosessi on seuraava: käyttäjä täyttää rekisteröintilomakkeen, jonka arvot välitetään käyttäjä-ohjaimella. Mikäli kontekstin mallin syötearvot ovat valideja, luodaan tietokantaan uusi käyttäjä, jolle asetetaan pysyvä autentikointieväste.

Kuvassa 36. on esitetty käyttäjä-ohjaimen rekisteröintimetodi, jossa toimitaan edellä mainitulla tavalla. Lisäksi toimenpiteinä ovat käyttäjän lisäys users-rooliin ja tervetulosähköpostin lähettäminen.

```
[HttpPost]
public ActionResult Rekisteroi(KayttajaModels.RekisterointiModel model)
{
    if (ModelState.IsValid)
    {
        // Yritetään rekisteröidä käyttäjä
        MembershipCreateStatus createStatus = MembershipService.CreateUser(model.KayttajaNimi, model.Salasana, model.Email);

        if (createStatus == MembershipCreateStatus.Success)
        {
            //Lisätään käyttäjä users-rooliin - luodaan pysyvä eväste
            Roles.AddUserToRole(model.KayttajaNimi, "users"); FormsAuthentication.SetAuthCookie(model.KayttajaNimi, false);
            FormsService.SignIn(model.KayttajaNimi, false);

            //Tervetuloviesti (Käyttäjänimi ja salasana)
            MailClient.LahetaTervetuloa(model.Email, model.KayttajaNimi, model.Salasana);
            FormsAuthentication.SetAuthCookie(model.KayttajaNimi, false);

            return RedirectToAction("Index", "Etusivu");
        }
        else
        {
            ModelState.AddModelError("", KayttajaModels.AccountValidation.ErrorCodeToString(createStatus));
        }
    }
    // Jos päädyttiin tähän, jotain meni väärin, näytä uudelleen formi
    return View(model);
}
```

Kuva 36. Käyttäjä-ohjaimen rekisteröinti-metodi.

### 5.6.3 Autorisointi

Autorisointi tarkoittaa käyttöoikeuksien hallintaa – määrittelee, mitä toimintoja tunnistettu käyttäjä voi järjestelmässä tehdä. ASP.NET MVC -ympäristössä autorisointi voidaan toteuttaa yksinkertaisella Authorize-suodatinattribuutilla. (Bogard ym. 2012, 136.)



Authorize-attribuutti voidaan määritellä koko ohjain-luokalle tai yksittäisiin Action-metodeihin. Attribuutin käyttö tarkoittaa, että toiminnon suoritus edellyttää autentikointia, jolloin tunnistamaton käyttäjä ohjataan järjestelmän kirjautumis-sivulle. Attribuuttiin voidaan lisätä parametreja, joilla voidaan määritellä käyttöoikeus rooleille tai yksittäisille käyttäjille. Tämä tarkoittaa, että autentikoinnin lisäksi käyttöoikeus edellyttää parametrien kriteerien täyttämistä. (Esposito 2011, 228–229.)

Opinnäytetyön arviointityökalussa autorisointi on toteutettu siten, että käyttöoikeudet on huomioitu roolitasolla. Kuvan 37 esimerkissä on esitetty ylläpitosivun autorisoinnin toteutus.

```
[Authorize(Roles = "admins")]
public class YllapitoController : Controller
```

Kuva 37. Ylläpidon autorisoinnin toteutus. Kaikki ylläpito-ohjaimen toiminta edellyttää, että käyttäjä on autentikoitu ja edustaa admins-roolia.

#### 5.6.4 Käyttäjän yksityisyydensuoja

Aiemmin esitettyjen käyttäjähallinta-asioiden mielekkyys konkretisoituu vasta sitten, kun käyttäjällä on vaikutusmahdollisuudet vain omaan materiaaliin. Tässä luvussa käsitellään prosessi, jolla saavutetaan kyseinen päämäärä.

Lausunnon ja käyttäjän välinen assosiaatio muodostetaan lausunto-mallin omistaja-ominaisuuden avulla. Lausunto-ohjaimessa määritellään, että uuden lausunnon luonnin yhteydessä omistajaolion arvoksi muodostetaan tunnistetun käyttäjän nimi (Kuva 30) – tieto tallennetaan tietokantaan.

Lausunto-mallissa on myös metodi (Kuva 38), jolla verrataan omistaja-merkkijonon ja argumenttina annettavan käyttäjänimen arvoja. Mikäli nimet täsmäävät, palauteaan arvoksi tosi.

```
public bool IsOmistaja(string kayttajanimi)
{
    return String.Equals(Omistaja, kayttajanimi, StringComparison.OrdinalIgnoreCase);
}
```

Kuva 38. Metodi, jota käytetään lausunnon ja käyttäjän välisen yhteyden toteuttamisessa (Conery ym. 2009, 117).

Lausunto-näkymässä suoritetaan foreach-toistosilmukka (Kuva 39) lausunto-mallin olioille. Silmukan sisällä on ehtolause, joka tutkii edellä mainitun metodin avulla, omistaako tunnistettu käyttäjä lausunnon. Mikäli ehto on tosi, luodaan Partial View -tyyppinen näkymä.

```
@foreach (var item in Model)
{
    if (item.IsOmistaja(User.Identity.Name))
    {
        @Html.Partial("_Lausunto", item)
    }
}
```

Kuva 39. Ohjelmakoodi luo lausuntosivulle kaikkien lausuntojen esikatselut, jotka käyttäjä omistaa.

Tällä hetkellä konstruktiossa on vakava tietoturva-aukko. Asiantunteva käyttäjä voi nähdä vieraan käyttäjän lausuntoja manipuloimalla URL-osoitteita. Kun käyttäjä valitsee lausunnon muokkaustoiminnon, URL-osoitteeksi muodostetaan esimerkiksi /Lausunto/Muokkaa/1.

Osoitteen lopussa oleva numero edustaa lausunnon id-tunnusta, joka muodostetaan automaattisesti, kun lausunto luodaan. Käyttäjä voi vaihtaa kyseisen numeron, jolloin URL-osoite viittaa eri lausuntoon, kuin alun perin – lausunto voi olla eri käyttäjän.

Kyseinen skenaario vältetään luomalla tarvittaviin lausunto-ohjaimen Action-metodeihin (HTTP GET) ehtolause, jolla tutkitaan omistaako käyttäjä lausunnon. Mikäli ei omista, palautetaan näkymä, jossa ilmoitetaan, että käyttäjällä on mahdollisuus vaikuttaa vain omaan materiaaliin.

### 5.6.5 Sivustoon kohdistuvat uhat

Opinnäytetyön arviointityökalussa on noteerattu seuraavat sivustoon kohdistuvat uhkatekijät: XSS-hyökkäys (Cross-Site Scripting) ja CSRF (Cross-Site Request Forgery).

XSS-hyökkäyksen lähtökohta on esimerkiksi, että hyökkääjä yrittää lähettää haitallisen JavaScript-koodin ja/tai HTML:n sivuston syötteistä. Hyökkäyksen onnistuessa on muun muassa mahdollista varastaa eväste (käyttäjätunnus ja salasana), muuttaa selaimen asetuksia ja asentaa haittaohjelma. (Nieminen 2012.)

Oletuksena Razor-näkymämoottori koodittaa (encode) kaiken renderoitavan tekstin. Tämä tarkoittaa, että erikoismerkit muutetaan HTML-muotoon. Esimerkiksi lainausmerkki muutetaan muotoon &quot;. Tämä estää esimerkiksi haitallisen JavaScript-koodin tulkkauksen, jolloin koodi näkyy vain tekstinä. (Freeman & Sanderson 2011, 717.)

Kooditus voidaan ohittaa näkymissä Html.Raw-metodilla. Mikäli syöte liittyy malli-kontekstiin, tarvitaan myös AllowHtml-attribuutti (Kuva 29) mallin ominaisuudessa. (Bogard ym. 2012, 143–144.)

Opinnäytetyön arviointityökalussa tekstieditorin toiminta edellyttää HTML-syötteiden sallimista, mutta potentiaalinen XSS-hyökkäyksen uhka on otettu huomioon. Lähtökohtaisesti kaikki syötettävä teksti muodostetaan <p>-tagien sisälle. Jos esimerkiksi kirjoitetaan skripti, jolla pyritään luomaan ponnahdusikkuna, tämä tulos-tuu vaarattomana tekstinä (Kuva 40).

#### XSS-Hyökkäys

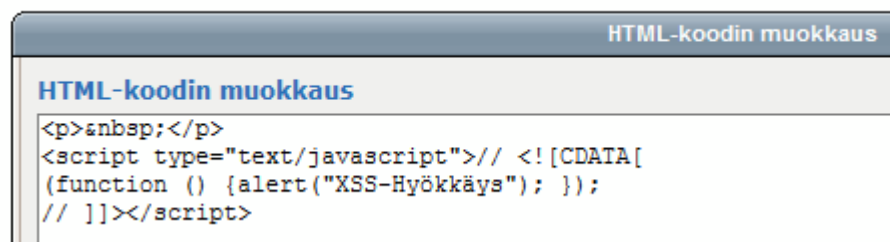
Opinnäytetyön ohjaaja: **Admin**

Opinnäytetyön tekijä: **Admin**

```
<script>(function () {alert("XSS-Hyökkäys"); })</script>
```

Kuva 40. XSS-hyökkäystä edustava skripti <p>-tagien sisällä (Nieminen 2012).

Ominaisuutta saatetaan yrittää kiertää, esimerkiksi kytkemällä JavaScript pois päältä. Mikäli näin on toimittu ja syötetään kyseinen skripti, tekstieditori muuttaa sen vaarattomaksi (Kuva 41).



Kuva 41. XSS-hyökkäystä edustava skripti on neutralisoitu.

Lähtökohtaisesti kannattaa toteuttaa myös palvelinpuolelle toiminto, jolla puhdistetaan vaaralliseksi epäilty merkit käyttäjän syötteestä. Tämä voidaan toteuttaa esimerkiksi Microsoftin AntiXSS-kirjastolla. (Chadwick ym. 2012, 198.) Tämä ei kuitenkaan toimi TinyMCE-sovelluksen kanssa.

CSRF-hyökkäyksessä on kyse seuraavasta tilanteesta:

- Käyttäjän selaimelle lähetetään kirjautumisen yhteydessä autentikointieväste, joka kulkee saman sivuston sivupyynnöissä istunnon (session) ajan.
- Vihamielinen palvelin yrittää lähettää käyttäjän selaimelle formin, jossa on sama rakenne ja URL kuin evästeen lähettäneessä palvelimessa.
- Kun käyttäjä suorittaa submit-toiminnon tälle formille, vihamielinen palvelin saa käyttäjän evästeen, koska käyttäjän palvelin ei reagoi siihen, mistä pyyntö saapuu. (Nieminen 2012.)

CSRF-hyökkäys voidaan estää MVC-Frameworkin AntiForgeryToken (Kuva 31) helper -metodin ja ValidateAntiForgeryToken-attribuutin (Kuva 30) kombinaatiolla. AntiForgeryToken-metodi luo formiin piilokentän (Kuva 41), jossa on arvo palvelimen istunnon identifioimiseksi. Sama arvo tallennetaan myös selaimelle lähetettävään evästeeseen. (Nieminen 2012.)

```
<form action="/Lausunto/Muokkaa/3" method="post"><input name="__RequestVerificationToken" type="hidden" value="2v
<legend>Tiedot (pakolliset merkitty punaisella)</legend>
```

Kuva 41. AntiForgeryToken-metodin luoma piilokenttä.

Action-metodiin liitetty `ValidateAntiForgeryToken`-attribuutti toteuttaa prosessin, jossa verrataan piilokentän ja evästeen arvoja validointitoimenpiteenä. Mikäli arvot eivät täsmää, syntyy poikkeus (exception) – hyökkäys epäonnistuu. (Freeman & Sanderson 2011, 726.)

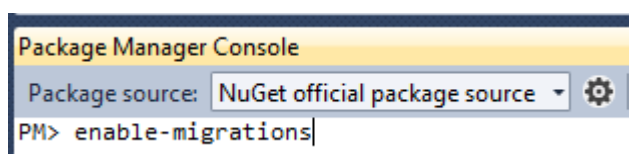
## 5.7 Julkaisutoimenpiteet

Opinnäytetyön arviointityökalun julkaisualustana toimii Windows-palvelin, jossa palvelinohjelmistona on IIS (Internet Information Services). Julkaisu vaatii useita toimenpiteitä, esimerkiksi IIS-palvelinohjelmiston ja sovelluksen (`Web.config`) konfigurointia. Tässä luvussa käsitellään Code First -migraatiot lausunto/kriteeritietokannalle sekä Visual Studio Publish-toiminto.

### 5.7.1 Code First -migraatiot lausunto/kriteeritietokannalle

Luvussa 5.1.3 esitettiin konsepti, jossa tietokanta poistetaan ja luodaan uudelleen testidatan kanssa, mikäli EDM-sovellusmalli on muuttunut. Tämä ei luonnollisesti ole ideaaliratkaisu tuotantoympäristössä, koska olemassa oleva data menetetään.

Tuotantoympäristön tietokanta luodaan Code First -migraatioilla. Voidaan myös määrittää, mitä dataa tietokantaan sisällytetään. Data säilyy, vaikka tietokantaan tehtäisiin muutoksia. Code First -migraatiot otetaan käyttöön Visual Studio Package Manager Console -valikosta. (Dykstra 2011.) Kuvassa 42 on esitetty komento, jolla Code First -migraatiot otetaan käyttöön.



Kuva 42. Code First -migraatioiden käyttöönottokomento.

Code First -migraatioiden käyttöönotto luo projektiin Configuration.cs-tiedoston Migrations-kansioon. Tiedostossa on Seed-metodi, johon voidaan määritellä koodi, mikä lisää halutun datan tietokantaan. (Dysktra 2011.) Opinnäytetyön arviointityökalussa luotiin kriteerilauseet tuotantoympäristöön kyseisellä periaatteella.

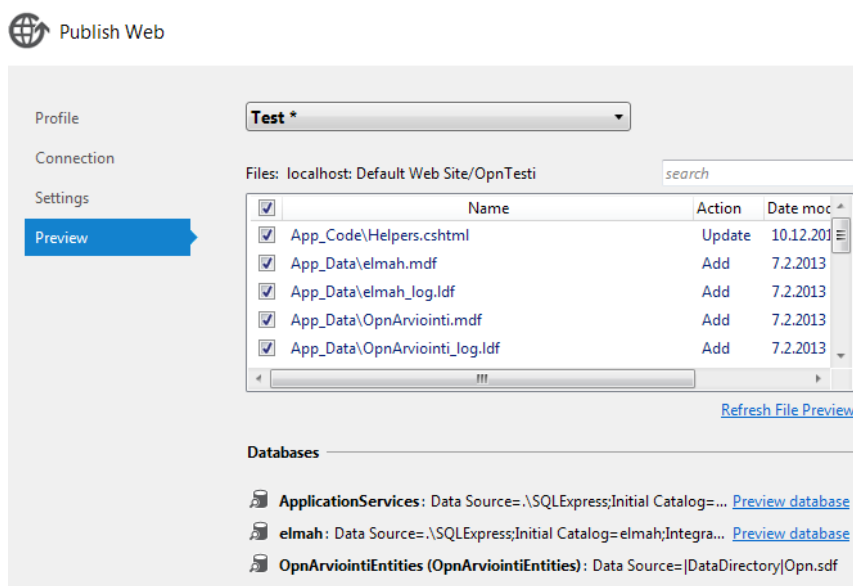
Lisäksi toteutetaan migraatio, jolla luodaan varsinainen tuotantoympäristön tietokantaskema – prosessi noteeraa tiedonsaantitason käyttöliittymäluokan määrittelyt. Mikäli tietokannan rakenteeseen tehdään tulevaisuudessa muutoksia, nämä noteerataan uusilla migraatioilla. Kuvassa 43 on esitetty koodi, jolla prosessissa luodaan lausunto-taulu.

```
CreateTable(
    "dbo.Lausunto",
    c => new
    {
        LausuntoID = c.Int(nullable: false, identity: true),
        Arvio = c.Int(),
        Omistaja = c.String(maxLength: 50),
        OpiskelijaNro = c.String(nullable: false, maxLength: 30),
        Tekija = c.String(nullable: false, maxLength: 50),
        TyonNimi = c.String(nullable: false, maxLength: 256),
        Ohjaaja = c.String(nullable: false, maxLength: 50),
        ToinenOhjaaja = c.String(maxLength: 50),
        LausuntoTeksti = c.String(),
        Aika = c.DateTime(nullable: false),
    })
    .PrimaryKey(t => t.LausuntoID);
```

Kuva 43. Koodi, jolla Code First -migraatioilla luodaan lausunto-taulu. Kuvasta voidaan todeta muun muassa luvussa 5.1.1 mainittu taulunimen käyttäytyminen.

### 5.7.2 Visual Studion Publish-toiminto

Visual Studio sisältää Publish-toiminnon (Kuva 44) sivuston julkaisemiseksi. Opinnäytetyön arviointityökalun julkaiseminen toteutettiin kyseisen toiminnon uusimmalla versiolla. Uudistettu versio virtaviivaistaa julkaisuprosessia ja lisää ominaisuuksia, kuten Code First -migraatioiden tuen ja uudistetun käyttöliittymän (Hashimi 2012).



Kuva 44. Visual Studion Publish -toiminto. Asetuksista voidaan määrittellä muun muassa, mitä materiaalia julkaistaan.

Julkaisuprofiilin nimeksi on annettu Test. Tällä viitataan projektiin lisättyyn Web.Test.config-tiedostoon, jolla ohitetaan Web.config-tiedoston määrittelyjä. Esimerkiksi lausunto/kriteeritietokannan yhteysmerkkijonoa on muutettu ja sovelluksen debug-ominaisuus otettu pois käytöstä.

## 5.8 Järjestelmän testaus

Opinnäytetyön arviointityökalun testausta suoritettiin pääosin Visual Studion ajoympäristössä, jolloin selaimena toimi Mozilla Firefox. Testikäyttäjiä luotiin useampia muun muassa käyttäjähallinnan toimivuuden varmistamiseksi. Luvussa 5.1.3 esitetty konsepti testidatan muodostamiseksi osoittautui erittäin käyttökelpoiseksi, esimerkiksi ulkoasun suunnittelun yhteydessä.

Järjestelmää testattiin myös paikalliseen ympäristöön julkaistuna. Palvelinsovelluksena toimi IIS. Tässä skenaariossa testausta toteutettiin myös Internet Explorer, Opera, Safari ja Chrome -selaimilla. Sovelluksen käyttäytymisessä ei ollut selainten välillä toiminnallista eroa. Ulkoasussa ilmeni paikoin pieniä eroavaisuuksia, mutta näihin toteutettiin ratkaisut.

## 6 JOHTOPÄÄTÖKSET

Opinnäytetyön arviointityökalun toteuttaminen oli ammatillisen osaamisen kehittämisen kannalta erittäin tärkeä projekti. Opintoihini on sisältynyt web-järjestelmien toteuttamisen osalta ASP.NET Web Forms -tekniikka, joka on vaihtoehtoinen toteutusfilosofia ASP.NET MVC -tekniikalle. Opiskelin ASP.NET MVC -tekniikkaa itsenäisesti lukemalla kirjallisuutta sekä tekemällä esimerkkisovelluksia. Ajax, jQuery, Entity Framework ja LINQ to Entities olivat minulle myös tuntemattomia tekniikoita. Lisäksi projektilla oli ylläpitävä ja kehittävä vaikutus aiemmin opittuihin tekniikoihin.

Opinnäytetyön arviointityökalun tekeminen oli haastava ja mielenkiintoinen projekti. Sisäistin MVC-arkkitehtuurin kantavan ajatuksen melko nopeasti ja hahmotin mil-laista potentiaalia sillä on web-järjestelmien toteuttamiseksi. Pidän lähestymistapaa mielekkäämpänä kuin Web Forms -tekniikkaa. Suurimmat vaikeudet liittyivät kriteeridatan esittämiseen niin, että käyttäjällä on helposti saatavilla kaikki kategorian kriteerilauseet. Partial View -näkymien käyttäminen ja niiden esiintymisen ehdollistaminen ratkaisi useita ongelmia.

Olen projektin lopputulokseen todella tyytyväinen. Mielestäni asettamani tavoitteet saavutettiin erittäin hyvin, koska opinnäytetyön arviointityökalulla voidaan toteuttaa opinnäytetyöstä annettavan lausunnon valmistusprosessi kokonaisuudessaan – käyttäjä ei tarvitse kolmannen osapuolen sovelluksia.

Järjestelmällä on myös kehityspotentiaalia. Esimerkiksi autentikointi voitaisiin toteuttaa Windows-autentikoinnilla, mikä olisi paremmin linjassa asiakasyrityksen muiden järjestelmien kanssa. Päädyin Forms-autentikointiin, koska se oli helpommin toteutettavissa ja testattavissa olosuhteisiin nähden. Toinen mainittava esimerkki on, että lausunnoista luotavat PDF-tiedostot tallennettaisiin järjestelmään käyttäjäkohtaisesti. Käyttäjä voisi esimerkiksi lähettää tiedoston sähköpostilla arkistoinnista vastaavalle taholle.



## LÄHTEET

- Allen, S., Galloway, J., Haack, P. & Wilson, B. 2011. Professional ASP.NET MVC 3. Indianapolis: John Wiley & Sons, Inc.
- Barrett, K. 2009: Introducing the ADO.NET Entity Framework. Viitattu 2.1.2013. Saatavissa: <http://fyi.oreilly.com/2009/02/introducing-the-adonet-entity.html>
- Beletsky, A: ELMAH.MVC. Viitattu 12.1.2013. Saatavissa: <https://github.com/alexanderbeletsky/elmah.mvc>
- Bellinaso, M., Berardi, N. & Katawazi, A. 2009. ASP.NET MVC 1.0 Website Programming: Problem – Design – Solution. Indianapolis: Wiley Publishing, Inc.
- Blakeley Silver, T. 2010. Wordpress 3.0 jQuery. Birmingham: Packt Publishing Ltd.
- Boehm, A. & Ruvalcaba, Z. 2012. Murach's HTML5 and CSS3. Mike Murach & Associates, Inc.
- Bogard, J., Hexter, E., Hinze, M., Palermo, J. & Skinner, J. 2012. ASP.NET MVC 4 In Action. Shelter Island: Manning Publications Co.
- Boumphrey, F., Greer, C., Raggett, D., Raggett, J., Schnitzenbaumer, S. & Wugofski, T. 2000. XHTML Ohjelmoijan Käsikirja. Helsinki: Oy Edita Ab.
- Bozio, G. 2012: Rotativa, how to print PDF in ASP.NET MVC. Viitattu 23.1.2013. Saatavissa: <http://www.codeproject.com/Articles/335595/Rotativa-how-to-print-PDF-in-Asp-Net-MVC>
- Bradford, A., Casario, M., Congleton, A., Improta, F., Reid, J., Saltzman, D. & Wormser, N. 2012. CSS3 Solutions: Essential Techniques for CSS3 Developers. Friends of ED – Apress.
- Cate, S., Glavich, P., McClure, W. & Shoemaker, C. 2006. Beginning Ajax with ASP.NET. Wiley Publishing, Inc.
- Cederholm, D. 2010. CSS3 for Web Designers. Jeffrey Zeldman.
- Chadwick, J., Panda, H. & Snyder, T. 2012. Programming ASP.NET MVC 4. O'Reilly Media, Inc.
- Conery, R., Guthrie, S., Haack, P. & Hanselman, S. 2009. Professional ASP.NET MVC 1.0. Indianapolis: John Wiley & Sons, Inc.
- Cook, C. & Garber, J. 2012. Foundation HTML5 with CSS3. Friends of ED – Apress.
- C. Zakas, N. 2012. Professional JavaScript for Web Developers. 3. painos. Indianapolis: John Wiley & Sons, Inc.

Dykstra, T. 2011: Deploying an ASP.NET Web Application to a Hosting Provider using Visual Studio or Visual Web Developer. Viitattu 10.2.2013. Saatavissa: <http://www.asp.net/mvc/tutorials/deployment/deployment-to-a-hosting-provider/deployment-to-a-hosting-provider-introduction-1-of-12>

Dykstra, T. 2012. Getting Started with the Entity Framework 4.1 Using ASP.NET MVC. Microsoft.

Esposito, D. 2011. Programming Microsoft ASP.NET MVC. 2. painos. Microsoft Press.

Frain, B. 2012. Responsive Web Design with HTML5 and CSS3. Birmingham: Packt Publishing Ltd.

Freeman, A. & Sanderson, S. 2011. Pro ASP.NET MVC 3 Framework. 3. painos. Apress.

Galloway, J. 2012: 10 Things ASP.NET Developers Should Know About Web.config Inheritance and Overrides. Viitattu 28.12.2012. Saatavissa: <http://weblogs.asp.net/jgalloway/archive/2012/01/17/10-things-asp-net-developers-should-know-about-web-config-inheritance-and-overrides.aspx>

Grannell, C., Sumner, V. & Synodinos, D. 2012. The Essential Guide to HTML5 and CSS3 Web Design. Apress.

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

Hashimi, S. 2012: Visual Studio 2010 Web Publish Updates. Viitattu 13.2.2013. Saatavissa: <http://blogs.msdn.com/b/webdev/archive/2012/06/15/visual-studio-2010-web-publish-updates.aspx>

Holzner, S. 2006. Ajax For Dummies. Wiley Publishing, Inc.

Koskimies, K. 2000. Oliokirja. Jyväskylä: Satku – Kauppakaari.

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

Larsen, R. & Otero, C. 2012. Professional jQuery. Indianapolis: John Wiley & Sons, Inc.

Lerman, J. & Miller, R. 2012. Programming Entity Framework: Code First. O'Reilly Media, Inc.

Liu, M. 2010. WCF 4.0 Multi-tier Services Development with LINQ to Entities. Birmingham: Packt Publishing Ltd.

Microsoft: ASP.NET MVC Overview. Viitattu 20.12.2012. Saatavissa: <http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview>

Moxiecode Systems. Viitattu 24.1.2013. Saatavissa: <http://www.tinymce.com/index.php>

Murach, M & Ruvalcaba, Z. 2012. Murach's JavaScript and jQuery. Mike Murach & Associates, Inc.

Nieminen, H. 2012. Web-järjestelmän ohjelmointi.

NuGet. Viitattu 2.1.2013. Saatavissa: <http://nuget.codeplex.com/>

Powers, D. 2012. Beginning CSS3. Apress.

Rutter, J. 2011. Smashing jQuery. John Wiley & Sons, Inc.

Sarrion, E. 2012. jQuery UI. O'Reilly Media, Inc.

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

Sawyer McFarland, D. 2012. JavaScript & jQuery The Missing Manual. 2. painos. O'Reilly Media, Inc.

Walther, S. 2010. ASP.NET MVC Framework Unleashed. Sams.