Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Sähköisen liiketoiminnan järjestelmät

2016

Arvin Mahmudi

VALJAKKOAJON TULOSPALVELUOHJELMAN KEHITTÄMINEN ASP.NET-YMPÄRISTÖSSÄ

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutusohjelma | Sähköisen liiketoiminnan järjestelmät

Lokakuu 2016 | 33

Ohjaaja: Päivi Nygren

Arvin Mahmudi

VALJAKKOAJON TULOSPALVELUOHJELMAN KEHITTÄMINEN ASP.NET-YMPÄRISTÖSSÄ

Tämä opinnäytetyö käsittelee Hepoteqille tehtyä valjakkoajon kestävyyskokeen estepalveluiden tulospalveluohjelmaa. Projektin päätavoitteena oli luoda selainpohjainen sovellus, joka helpottaisi valjakkoajon kestävyyskokeen estepalveluiden aikojen kirjaamista, sekä toteuttaa reaaliaikaisesti päivittyvä tulospalvelunäyttö katsojille.

Toimeksiantajalta saatiin projektiin pakolliset ja vaihtoehtoiset määrittelyt. Kehityksessä hyödynnettiin Microsoftin työkaluja, pääasiallisesti C# -ohjelmointikieltä Microsoft Visual Studion .NET -alustalla, sekä Microsoft SQL -tietokannanhallintajärjestelmää. Ennen kehityksen aloittamista oli jo selvää mitä teknologioita tullaan käyttämään sivuston kehityksessä aikaisemman kokemuksen perusteella ja soveltuvuuden kannalta. Ohjelmistoa testattiin satunnaisella tiedonsyötöllä käyttöliittymän kautta.

Sovellusta luotaessa käytettiin hyväksi laajalti tekijän aikaisempaa osaamista, useita keskustelufoorumeita ja alan kirjallisuutta. Ongelmia tuotti suurimmalta osin asiakaspuolen ohjelmointimetodit, joiden opiskeluun laitettiin paljon aikaa. Lopputuloksena saatiin toimiva selainpohjainen sovellus. Sovelluksen kaikkia pakollisia ominaisuuksia ei saatu valmiiksi opinnäytetyön kirjoittamisen aikana, kuitenkin suurin osa toimeksiantajan määritellyistä ominaisuuksista saatiin kumminkin toteutettua. Sovellusta ei ole vielä julkaistu julkiseen käyttöön vielä. Kehitystä jatketaan tulevaisuudessa.

ASIASANAT:

Microsoft SQL Server, ASP.NET, C#, WWW

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology | e-Business System

October 2016| 33

Instructor: Päivi Nygren

Arvin Mahmudi

DEVELOPING CARRIAGE DRIVING SCOREBOARD APPLICATION IN ASP.NET ENVIROMENT

This thesis deals with the development of carriage driving endurance test scoreboard service program made for Hepoteq. The main objective of the project was to create a browser-based application that would facilitate time logging of carriage driving endurance test, as well as to implement in real-time updating scoreboard for viewers.

The client provided the mandatory and optional features for the project. Project itself was developed by utilizing Microsoft tools, mainly C# programming language on Microsoft Visual Studio .NET platform and Microsoft SQL database management system. Before starting the actual developing, it was already clear which technologies will be used for site development on the basis of previous experience and suitability. The software was tested with random data feed via the user interface.

Author's prior knowledge, several discussion forums and field literature were used in creating the application. Client-side programming methods produced many problems, which was studied a lot. The final result was a functional web-based application. All mandatory features of the application were not completed while writing the thesis, but most of the features specified by client were implemented anyway. The application has not been published for public use yet. Development will continue in the future.

KEYWORDS:

Microsoft SQL Server, ASP.NET, C#, WWW

SISÄLTÖ

KUVAT

KUVIOT

1 JOHDANTO

Tämän dokumentin yhteydessä tullaan käymään läpi projektin aikana käytetyistä työkaluista, tekniikoista, suunnittelusta, toteutuksesta, haasteista joihin törmätty projektin aikana sekä näytetään runsaasti käytännön esimerkkejä.

Toimeksiantajalle luotiin selainpohjainen ja tietokantaa hyödyntävä valjakkoajojen kestävyyskokeen estealueiden tulospalveluohjelma. Projektin pääasiallisena tarkoituksena oli helpottaa tulosten ylös kirjaamista ja näyttää tulokset katsojille reaaliaikaisesti päivittyvässä tulosnäytöllä. Sovellus on toteutettu C# .NET -ohjelmointikielellä ASP.NET ympäristössä.

Tämän opinnäytetyön toimeksiantajana toimi Hepoteqin toimitusjohtaja, jonka kanssa on käyty läpi sovelluksen määrittelyt ja luotu visio yhdessä sovelluksen ulkoasusta, sekä toiminnallisuudesta. Hepoteq on vuonna 2014 perustettu ohjelmistopalvelujen sekä niiden suojaamiseen liittyvien ratkaisuiden käyttöönottoja tuottava yritys, joka tuottaa pääasiassa hevosalalle suunnattuja ratkaisuja.

Sovellusta ei saatu kokonaan valmiiksi tämän opinnäytetyön aikana, mutta sitä tullaan kehittämään tulevaisuudessa. Sivustolla oleva reaaliaikaisesti päivittyvä tulospalveluohjelma, sekä tulosten syöttö palvelu tullaan eriyttämään muusta sivustosta ja toteutetaan myös mobiililaitteille. Tulospalveluohjelma kontrollin toteutuksessa on käytetty JavaScriptin Knockout-kirjastoa, jQueryn-kirjastoa ja ASP.NET MVC -mallia.

2 TYÖKALUT JA TEKNIIKAT

2.1 Visual Studio 2012

Visual Studio 2012 on Microsoftin kehittämä integroitu kehitysympäristö. Sitä käytetään tietokoneohjelmien, sivustojen, web-sovelluksien ja verkkopalvelujen kehityksessä. Visual Studio käyttää Microsoftin ohjelmistokehitysalustoja, kuten Windows API, Windows Forms ja Windows Presentation Foundation. Visual Studio tukee oletuksena Visual Basic, C# ja C++ ohjelmointikieliä, sekä XML/XSLT, HTML/XHTML, JavaScript ja CSS. Visual Studio tukee myös MUMPS, Python ja Ruby ohjelmointikieliä erikseen asennettuna. Tässä opinnäytetyössä on käytetty Visual Studion .NET -alustaa kehityksessä.

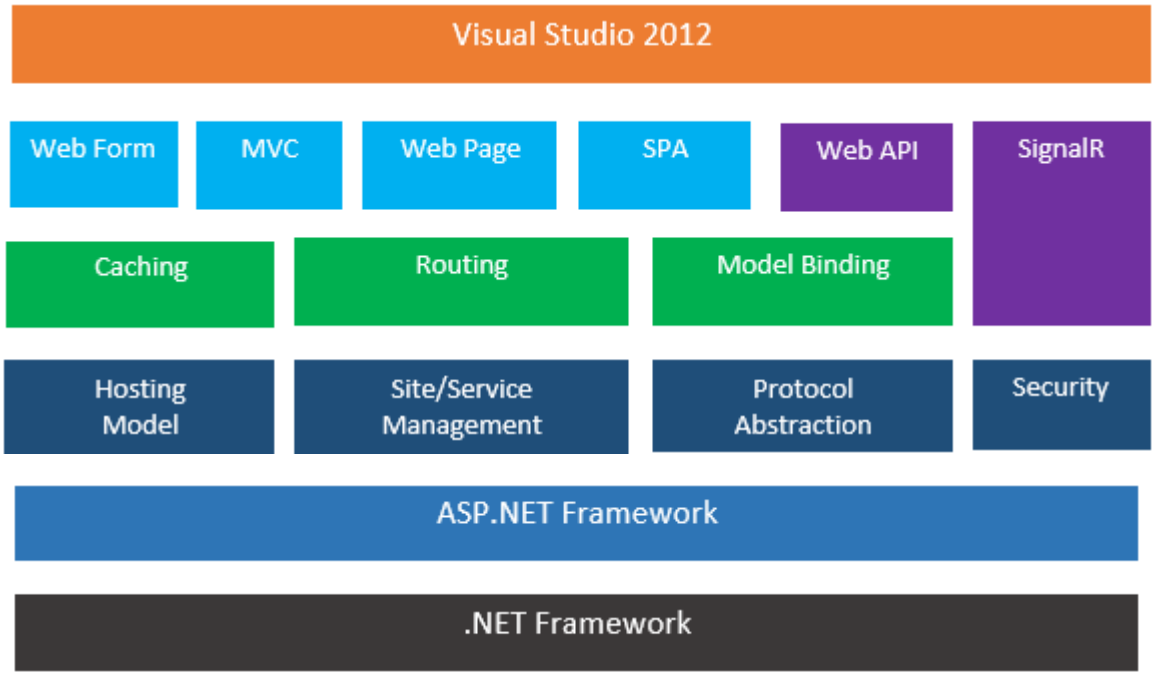
2.1.1 .NET -Alusta

.NET on Microsoftin luoma alusta mikä toimii pääasiallisesti Microsoft Windows -käyttöjärjestelmässä. .NET sisältää suuren määrän luokkakirjastoja, mikä tunnetaan nimellä Framework Class Library (FCL). Ensimmäinen versio .NET -alustasta julkaistiin helmikuussa 2002. Tässä projetissa on käytetty Microsoftin elokuussa 2012 julkaisemaa .NET 4.5 -alustaa. .NET-alustan malli tarjoaa mahdollisuuden kehittää ja toteuttaa erilaisia sovelluksia Visual Studion .NET-ympäristössä.

2.1.2 ASP.NET

ASP.NET on avointa lähdekoodia hyödyntävä palvelinpuolen web-sovelluksen alusta mikä on suunniteltu tuottamaan dynaamisia verkkosivuja. ASP.NET on Microsoftin kehittämä teknologia mikä on osa .NET-alustaa, luotu helpottamaan

kehittäjiä luomaan dynaamisia verkkosivuja, sovelluksia ja verkkopalveluita. ASP.NET-projektin vaatimukset ovat verkkosuunnittelun kieli(HTML/HTML5), asiakaspuolen koodi (JavaScript), Ohjelmointikieli(C#.NET/VB.NET) ja verkkopalvelin(IIS/ASP.NET-kehitysympäristö). Tässä projektissa on käytetty Microsoftin elokuussa 2012 julkaisemaa ASP.NET 4.5 -alustaa. (ASP.NET – Introduction)



Kuva 1. ASP.NET 4.5 –arkkitehtuuri (Chauhan 2013)

2.1.3 Ohjelmointikieli C# (Sharp)

C# (lausutaan C sharp) on Microsoftin kehittämä ohjelmointikieli mikä on luotu rakentamaan sovelluksia. .NET-alustaa hyödyntäen. C# on moderni

yleiskäyttöön tarkoitettu olioperustainen ohjelmointikieli. Ensimmäisen kerran C#-ohjelmointikieltä tavattiin vuonna 2000, kun .Net-alustaa kehitettiin ja siihen luotiin ohjelmointikieltä. Alun perin ohjelmointikieli tunnettiin nimeltä Cool (C-like Object Oriented language) joka muutettiin myöhemmin C#-nimiseksi. (Hejslsberg ym. 2004, 3)

2.1.4 Model-view-controller (MVC) -arkkitehtuuri

MVC-arkkitehtuuri on ohjelmistoarkkitehtuurityyli, jonka tarkoituksena on jakaa koodi kolmeen osaan: näkymiin, kontrolleihin ja malleihin. Malleihin sisällytetään kaikki tietokannassa olevat tiedot ja siihen liittyvät toiminnot. Käyttöliittymän ulkoasu ja tietojen näyttö määritetään näkymässä. Kontrolli vastaanottaa aineistopyynnöt käyttäjältä, jonka avulla saadaan välitettyä tieto mallin ja näkymän välillä. (Atwood, J. 2008)

2.1.5 HTML5

Uusin versio yleisesti käytetystä HTML-merkintäkielestä verkkosivujen kehittämiseen on HTML5-kieli. HTML5 tuo mukanaan paljon uudistuksia HTML-merkintäkieleen. mm. uudet multimedia-elementit, sekä selainohjelmoinnin rajapinnat, mikä mahdollistaa käyttäjän luomaan animoitua grafiikkaa tai offline-sovelluksia. Uuden version myötä merkintäkielessä voi käyttää uutta CSS3-tyyliohjejärjestelmää, jossa on mukana paljon uudistuksia. (HTML5)

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>HTML5-merkintäkieli</title>
</head>

<body>
    <div>
        <asp:Label ID="LBL_Numero" runat="server" Text="Valjakon numero"></asp:Label>:<br />
        <asp:TextBox ID="ValjakkoNro" runat="server" />
    </div>
</body>

</html>
```

Kuva 2. Esimerkki HTML5-merkintäkielestä.

2.1.6 Cascading Style Sheets (CSS)

CSS on tyyliohjeiden laji, joka on kehitetty pääsääntöisesti verkkosivujen ulkoasun tyylittelyyn HTML-merkintäkielessä. CSS on yksi tehokkaimmista ja tärkeimmistä työkaluista verkkosivujen suunnittelussa. CSS on kehitetty sallimaan tyylien erottelun muusta dokumentaatiosta.

Erottelu parantaa tyylien käytön käytettävyyttä, selkeyttää koodia sekä tyylejä voi käyttää useammalla sivulla ilman toistamista kun tyylit ovat erillisessä tiedostossa.

2.2 JavaScript

JavaScript on asiakaspuolen olipohjainen komentosarjakieli, jonka syntaksia voi verrata jollain tasolla C-ohjelmointikieleen. JavaScriptin käyttö yleistynyt huomattavasti ja sen käyttöä on helpotettu erilaisilla kirjastoilla. JavaScript komentosarjakieli kehitettiin alunperin kymmenessä päivässä ja sen esitteli ensimmäisenä Netscapen Brendan Eich toukokuussa 1995.

2.2.1 Knockout

Knockout on JavaScript-kirjasto, joka mahdollistaa HTML-elementin ja tietomallin yhdistämisen, sekä sillä pystyy parsimaan AJAX-kutsulla vastaanotettua dataa. Knockout tarjoaa kaksisuuntaisen yhteyden tietomallin ja HTML-elementin välillä, muutos kumpaan tahansa puoleen vaikuttaa myös toiseen. Ensimmäinen versio julkaistiin 5. heinäkuuta 2010 Steve Sandersonin toimesta, joka on kehittänyt ja ylläpitää projektia. (Knockout Introduction)

2.2.2 jQuery

JQuery on ilmainen avointa lähdekoodia hyödyntävä JavaScript-kirjasto, joka on suunniteltu yksinkertaistamaan asiakaspuolen ohjelmakoodia HTML-merkintäkielessä ja on tarkoitettu käytettäväksi kaikilla selaimilla. JQuery:n ensimmäinen versio julkaistiin vuonna 2006 ja sitä kehitetään jatkuvasti. JQuery on käytetyin JavaScript-kirjasto helposti ymmärrettävän syntaksin vuoksi. (Kuva 3.)

```
<link href="/styles/smoothness/jquery-ui.css" rel="stylesheet" media="screen" type="text/css">

<script type="text/javascript" src="/scripts/jquery-1.11.2.min.js"></script>
<script type="text/javascript" src="/jquery-ui-1.11.2.custom/jquery-ui.min.js"></script>
<script type="text/javascript" src="/jquery-ui-1.11.2.custom/datepicker-fi.js"></script>

<script type="text/javascript">
    $(function () {
        $("#<%= AlkuPvm.ClientID %>").datepicker({
            showOn: 'button',
            buttonImage: '../images/calendar.gif',
            buttonImageOnly: true,
            firstDay: 1
        });

        $("#<%= LoppuPvm.ClientID %>").datepicker({
            showOn: 'button',
            buttonImage: '../images/calendar.gif',
            buttonImageOnly: true,
            firstDay: 1
        });

    });

</script>
```

Kuva 3. Esimerkki jQuery-koodista joka mahdollistaa kalenterinäkymän.

Erikseen ladattu jQuery-moduuli mahdollistaa suomenkielisen kalenterinäkymän selaimella. (Kuva 4.)

Kuva 4. JQuery:lla toteutettu kalenterinäkymä selaimella.

2.3 Microsoft SQL Server 2008 R2

Microsoft SQL Server on Microsoftin kehittämä relaatiokannan hallintajärjestelmä, jolla voi tehdä erilaisia hakuja, muutoksia ja lisäyksiä. Alun perin tietokannanhallintajärjestelmää oli kehittämässä kolme yritystä, Microsoft, Sybase ja Ashton-Tate. Ensimmäinen versio OS/2 julkaistiin vuonna 1989. Windows NT-käyttöjärjestelmän julkaisun jälkeen Microsoftin ja Sybase kehittäjien tiet erkanivat. Microsoft siirsi SQL-serverin Windows NT -käyttöjärjestelmään ja keskittyi kehittämään sekä edistämään tuotetta. Tällä hetkellä markkinoiden suurimpia tietokannanhallintajärjestelmiä ovat MySql, Access, Microsoft SQL Server ja Oracle. SQL Server ja Oracle ovat johtavassa asemassa markkinoilla mainituista hallintajärjestelmistä. Microsoftin SQL-serveri on tehokas ja helppokäyttöinen back-end tietokannanhallintajärjestelmä, joka näkyy myös tuotteen suosiossa. Tässä

opinnäytetyössä on käytetty SQL Server 2008 R2 -versiota, mikä julkaistiin 21. päivä huhtikuuta 2010 ja SQL Server 2005 -version mukana julkaistua SQL Server Management Studio -työkalua. SQL Server Management Studio (SSMS) on integroitu ympäristö, jolla voi konfiguroida käsitellä, kehittää ja hallita kaikkia SQL-serverin tarjoamia komponentteja. (Microsoft SQL Server)

Microsoft SQL serverissä voi hyödyntää proseduureja. Proseduurit ovat SQL-skriptejä, tallennettu SQL-serverin tietokantaan. Yksi proseduurien hyödyistä on suorituskyvyn parantaminen. Proseduurien avulla kyselyjen suorittaminen on huomattavasti nopeampaa kuin sovelluksen päässä tehtävät kyselyt, koska tietokanta pystyy luomaan, optimoimaan ja tallentamaan kyselyt välimuistiin etukäteen. Proseduureja on helppo ylläpitää, koska ne on tallennettu erikseen muusta sovelluksen koodista. Tietoturva on myös parempi proseduureja luodessa, asiakkaalle voi myöntää käyttöoikeudet suorittaa, lisätä tai muokata tietoja ilman oikeuksia hallinnoida tauluja. (Marufuzzaman 2010)

3 SOVELLUKSEN TOTEUTUS

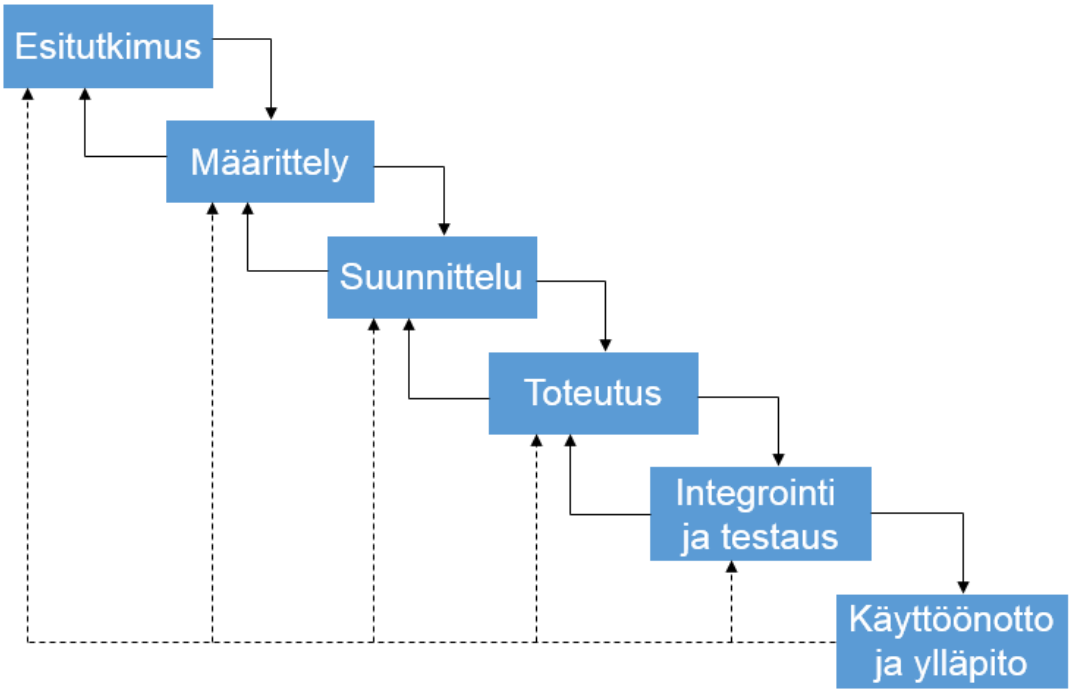
3.1 Määrittely

Ohjelmiston määrittelyprosessi alkaa vaatimusten kartoituksella. Ennen sovelluskehitystä on tarkoitus selvittää asiakkaan haluamia toimintoja ohjelmistoon, näiden perimmäiset syyt, tärkeyden arviointi, selvittää ja sovittaa mahdolliset ristiriitaiset vaatimukset yhteen. Ohjelmiston tärkein määrittelyyn liittyvä dokumentti on toiminnallinen määrittely, jonka luontiin on hyvä käyttää riittävästi aikaa ja luoda erittäin huolellinen sekä tarkka toteutus. Jälkeenpäin toimintojen lisääminen ja muokkaaminen ohjelmistoon on huomattavasti suuritöisempää ja kalliimpaa, kuin alkuvaiheessa huomioon otettuna. (Haikala & Märijärvi 2006, 36)

Arkkitehtisuunnittelu tehdään toiminnallisen määrittelyn jälkeen. Arkkitehtisuunnitteluvaiheessa järjestelmän toiminnot jaetaan erillisiin moduuleihin, jotta ne voidaan tehdä erikseen muusta projektista riippumatta useammalle suunnittelijalle.

Toimeksiantajalta sain projektiin erittäin hyvät ja tarkat määrittelyt, joiden perusteella oli helppo hahmottaa ja lähteä rakentamaan vaatimuksen puitteissa olevaa selainpohjaista, tietokantaa hyödyntävää sovellusta. Projektiin valittiin yksinkertaisuuden vuoksi elinkaarimalliksi vesiputousmalli, jossa vaiheet etenevät vain yhteen suuntaan. Vesiputousmallissa prosessi jaetaan lineaarisiin vaiheisiin, edellisen vaiheen tulos on seuraavan vaiheen syöte. Kuviossa 1 on esitetty vesiputousmallin eri vaiheet, johon kuuluu esitutkimus, määrittely, suunnittelu, toteutus, integrointi ja testaus sekä käyttöönotto ja ylläpito.

Tekninen määrittely syntyy arkkitehtisuunnittelun tuloksena, jossa kaikki ohjelmistoon liittyvät moduulit on kuvattu teknisellä tasolla. (Haikala & Märijärvi 2006, 36)



Kuvio 1. Vesiputousmalli (Haikala & Märijärvi, 2006).

Vesiputousmallissa on myös haittapuolensa, vaatimukset eivät saa muuttua prosessin aikana, asiakas näkee lopullisen tuloksen vasta projektin päättyessä ja myöhässä olevaa projektia on vaikea saada takaisin aikatauluun.

Toimeksiantajan vaatimat pakolliset ominaisuudet:

- Uuden kilpailun luonti
- Luokkien perustaminen
- Uuden valjakon lisääminen
- Valjakoiden liittäminen luokkiin
- Esteen lisäys
- Tulosten syöttö

    - Esteellä valitaan esteen numero listasta kisan mukaan, numero pysyy samana koko tulostensyötön ajan. Syötetään valjakon numero, aika ja virhepisteet.
- Tulosten listaus(tulospalvelu)
    - Lista kaikista tulleista tuloksista tulojärjestyksessä. Mahdollisuus muokata aikaa ja virhepisteitä, jos tiedetään niissä olevan virhe.
- Tulosten näyttö
    - Katsojille tarkoitettu reaaliaikaisesti päivittyvä näyttö, jossa tulokset näytetään tulojärjestyksessä.

Vaihtoehtoiset ominaisuudet:

- Mahdollisuus kertoa, mitkä esteet ajetaan missäkin luokassa.
- Mahdollisuus ilmoittaa missä järjestyksessä valjakot lähtevät.
- Mahdollisuus tarjota esteellä automaattisesti seuraavan valjakon numero.
- Tulospalvelussa mahdollisuus lajitella tuloksia valjakon tai esteen mukaan.
- Tulosten näyttöön erilaisia mahdollisuuksia tarkastella tuloksia valjakoittain, esteittäin ja luokittain.

3.2 Suunnittelu

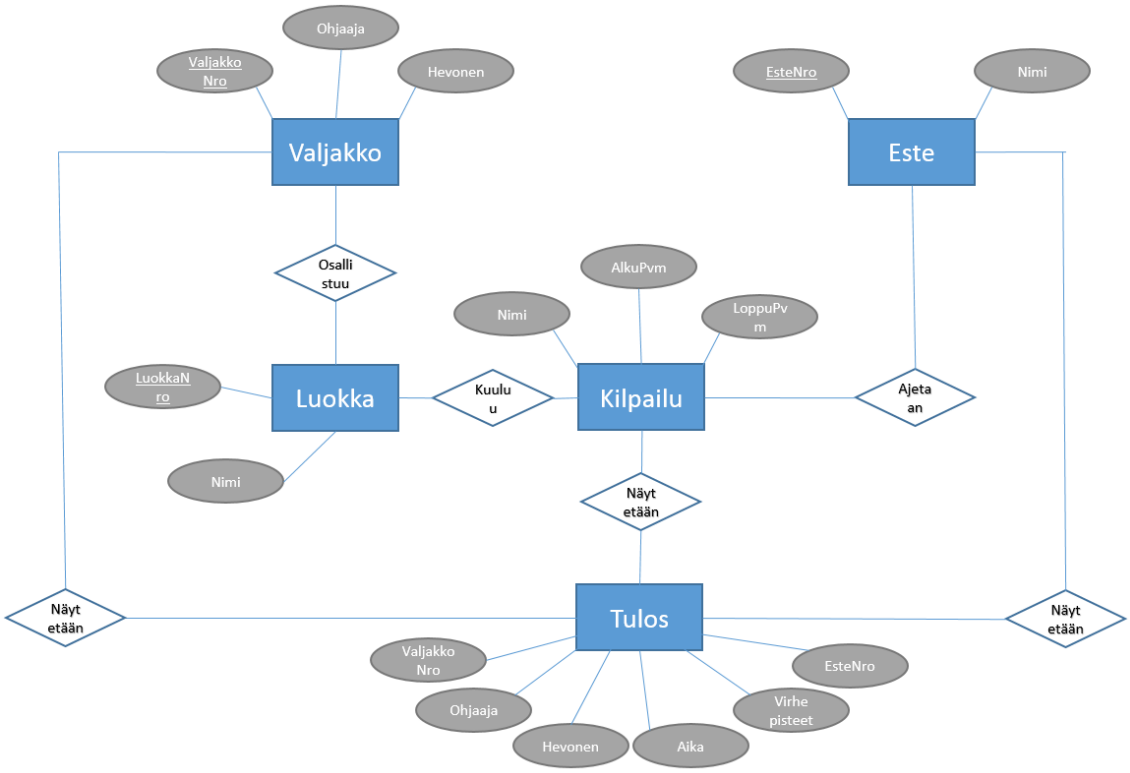
Aivan ensimmäinen vaihe suunnittelussa oli käyttöliittymäsovelluksen rakenne ja ulkoasu. Päädyimme toimeksiantajan kanssa käyttämään heille jo entuudestaan tuttuja sävyjä sivustolla, joita esiintyy heidän omilla sivuillaan. Sivuston rakenteesta haluttiin mahdollisimman yksinkertaisen näköinen myöhempää mobiilikäyttöä ajatellen. Kuviossa 2 on esitetty ensimmäinen suunniteltu rakenne sivustosta.

Kuvio 2. Www-sivuston rakenne.

Seuraavaksi käytiin toimeksiantajan kanssa yksityiskohtaisesti läpi kaikki pakolliseksi vaaditut määrittelyissä mainitut ominaisuudet jokaisen moduulin toiminnon osalta. Tässä vaiheessa tiedettiin jo, että reaaliaikaisesti päivittyvän tulosnäytön toteutus tulee olemaan haasteellista. Ohjelman tarvitsisi hakea tietyn aikavälin muutokset tietokannasta ja näyttää juuri tulleet muutokset tulosnäytöllä. Toimeksiantajan kanssa saatiin kuitenkin suunnitteluvaiheessa hyvä visio, kuinka tulosnäyttö tullaan toteuttamaan.

Hyvä tietokantasuunnittelu on jokaisen toimivan järjestelmän lähtökohta. Hyvällä tietokantasuunnittelulla helpotetaan järjestelmän kehitystä ja vältetään tiedon toistoa. Jatkuva tiedon toistuminen vie liikaa tilaa ja ylläpito on hankalaa.

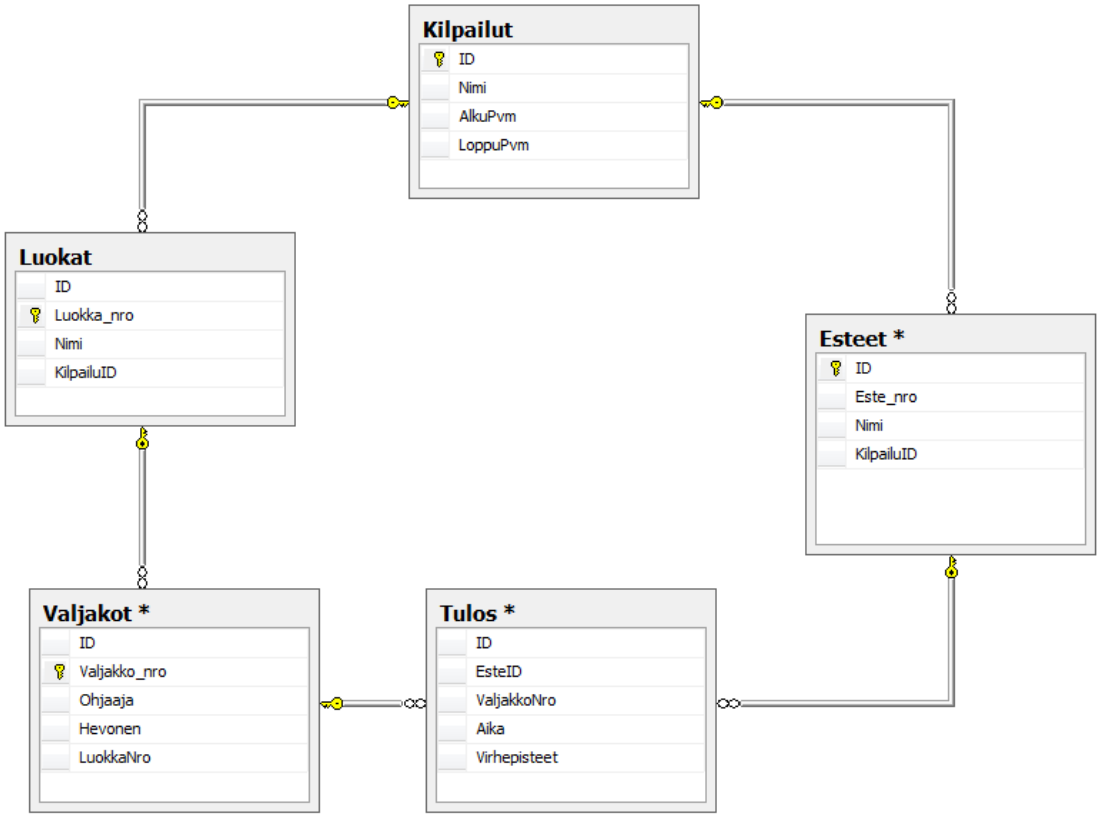
Tietokannan suunnitteluvaiheessa oli jo selvää, että proseduureja tullaan käyttämään, nopean tiedon haun vuoksi sekä helpottamaan ylläpitoa jatkokehitystä ajatellen. Jotta tietokannan rakenteesta ja sen sisällöstä saadaan hyvä käsitys, laadittiin ER-tietomalli. Suunnittelussa käytettiin ER-mallia yksinkertaisuuden vuoksi, se on helposti ymmärrettävä, helposti muunnettavissa toisenlaiseen tietomalliin ja tietokannan kehitysvaiheessa on toteutus suoraviivaista. (Quassnoi 2009.) Kuviossa 3. on esitettynä ER-malli.



Kuvio 3. ER-malli tietokannasta.

3.3 Tietokanta

Sovelluksen alustava tietokantasuunnittelu tehtiin määrittelyjen yhteydessä. Tietokannan perusrakenne, käytettävät taulut ja proseduurit hahmottuivat tässä vaiheessa, sekä ER-malli antoi hyvän pohjan tietokannan toteutukselle. Sovelluksen tietokantarakenne on esitetty kuviossa 4.

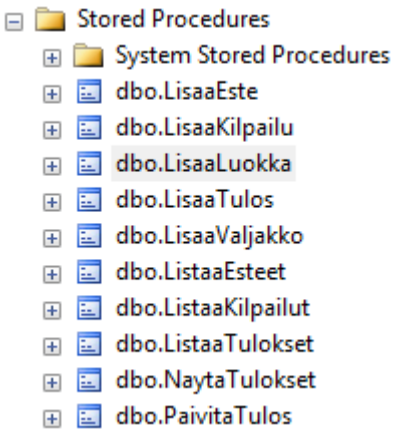


Kuvio 4. Järjestelmän tietokantarakenne.

Tietokantaan luotiin yhteensä viisi taulua, valjakot, luokat, esteet, kilpailut ja tulos taulut. Kaikki käyttöliittymässä tapahtuvat lisäykset päivitykset ja haut tapahtuvat proseduureja käyttämällä.

Jokainen tauluista on jollain tavalla riippuvainen toisistaan, sovelluksen oleellisin taulu on tulos-taulu, johon tallennetaan ja haetaan tuloksia ”NaytaTulokset”-proseduuria hyväksi käyttäen. Taulujen ja proseduurien luomiseen käytettiin Microsoft SQL Server Management Studio -työkalua.

Proseduureja luotiin tietokantaan yhteensä kymmenen kappaletta. (Kuva 5.)



Kuva 5. Proseduurit tietokannassa

Tällä hetkellä proseduurit ”ListaaTulokset” ja ”NaytaTulokset” ovat aivan samanlaiset proseduurit. ”ListaaTulokset”-proseduurilla listataan kaikki Tulos-taulun rivit tulosten syötön ja tuloksen listauksen yhteydessä, kun taas ”NaytaTulokset”-proseduurilla haetaan tulosnäyttöön Tulos-taulun kaikki rivit. Tässä vaiheessa on huomioitu tulevaisuudessa tapahtuvaa kehitystä, jolloin voidaan lisätä ehtoja kumpaan tahansa proseduuriin vaikuttamatta järjestelmän toiminnallisuuteen tai rikkomatta moduuleja.

3.4 Käyttöliittymä

Käyttöliittymä toteutettiin ASP.NET 4.5 -alustan tarjoamilla palveluilla. Aivan aluksi kehityksen yhteydessä tietokanta-asetukset määriteltiin Web.config-tiedostoon ja lisättiin projektiin ensimmäinen lisäyskontrolli. Kontrollin merkintäkielen päähän lisättiin tarvittavat tekstikentät ja Tallenna-nappi. Kontrollin koodi puolella kutsutaan Web.config-tiedostoon lisättyä tietokantakutsun avainta, näin saatiin ensimmäinen yhteys tietokantaan ja tallennettua dataa tauluun. Sovellukseen luotiin oma MasterPage-pohja, jonka

avulla voidaan luoda sovellukselle helposti haluttu mallipohja, jota voidaan käyttää sovelluksen kaikissa kontrolleissa. Sovelluksen MasterPage-pohja on esitetty kuvassa 6.

```
<head runat="server">
    <title>Tulospalvelu</title>

    <asp:ContentPlaceHolder ID="Stylesheets" runat="server">
        <link rel="stylesheet" href="/styles/valjakko.css" type="text/css" />
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
        <div>

            <img src="/images/Logo.png" align="left" />

            <table width="100%">
                <tr valign="top">
                    <td class="navigation">
                        <asp:TreeView ID="TreeView1" runat="server" DataSourceID="SiteMapDataSource1">
                        </asp:TreeView>
                        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
                    </td>

                    <td class="content">
                        <asp:ContentPlaceHolder ID="MainContent" runat="server">
                        </asp:ContentPlaceHolder>

                    </td>

                </tr>

            </table>
        </div>
    </form>
</body>
</html>
```

Kuva 6. Sovelluksen MasterPage-pohja.

MasterPage-pohjaan määritellään halutut tyylit, navigointi, logot jne. Masterpage-pohjan tärkeimpänä osana on ContentPlaceHolder-komponentti, johon lisätään kaikki sovelluksessa esitettävä sisältö.

Uuden web-lomakkeen lisäyksen yhteydessä voi valita Visual Studion tarjoamasta kontrollilistasta MasterPage-pohjaa käyttävän web-lomakeen, josta kaikki komponentit periytyvät. Käyttöliittymän päävalikon rakentamiseen on käytetty ASP.NET-alustan omaa TreeView-komponenttia, joka on lisätty suoraan MasterPage-pohjaan. Kaikki valikon tasot tallennetaan web.sitemap-tiedostoon, jolla on tarkoitus tallentaa valikkojen sisällöt. XML-pohjainen web.sitemap-tiedosto on esitetty kuvassa 7.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="Etusivu.aspx" title="Etusivu"  description="">
      <siteMapNode url="default.aspx" title="Lisää" description="">
        <siteMapNode url="Este.aspx" title="Lisää este"  description="" />
        <siteMapNode url="Luokka.aspx" title="Lisää luokka"  description="" />
        <siteMapNode url="Kilpailu.aspx" title="Lisää kilpailu"  description="" />
        <siteMapNode url="Valjakko.aspx" title="Lisää Valjakko"  description="" />
    </siteMapNode>

  <siteMapNode url="TulostenSyotto.aspx" title="Tulosten syöttö" description="" >
  </siteMapNode>

  <siteMapNode url="TulosListaus.aspx" title="Tulosten listaus" description="" >
  </siteMapNode>

  <siteMapNode url="TulosNaytto.aspx" title="Tulosnäyttö" description="" >
  </siteMapNode>
</siteMapNode>

</siteMap>
```

Kuva 7. Sovelluksen web.sitemap-tiedosto.

Jokaista tietokannassa esiintyvässä taulua varten on luotu omat luokat sovelluksessa. Luokkien tarkoituksena on pitää koodi helppolukuisena muillekin kehittäjille. Kaikki tietokantakyselyt tapahtuvat luokan sisällä, joita sitten kutsutaan kontrollin päässä.

Kaikki toiminnalisuudet tapahtuvat kontrolleissa, kontrollit lisätään web-lomakkeen ContentPlaceHolder-komponentin sisälle, joka perii kaikki MasterPage-mallipohjan komponentit. Kuvassa 8 on esitetty lopullinen näkymä käyttöliittymästä.

Kuva 8. Käyttöliittymä selaimella.

3.4.1 Tulosten syöttö

Tulosten syötössä valitaan ensin alasvetovalikosta kilpailu, jonka jälkeen listataan esteet valitun kilpailun mukaisesti. Kussakin kilpailussa voi olla eri määrä esteitä. Kun on valittu kilpailu sekä este täytetään kaikki lomakkeen tiedot ja painetaan lopuksi Tallenna-painiketta.

Ohjelma näyttää tekstikentän vieressä tähden, jos tekstikenttä jätetään tyhjäksi. (Kuva 9.)

Kuva 9. Tulosten syöttö käyttöliittymässä.

Käyttäjä täyttää lomakkeen kaikki kohdat ja tallentaa lomakkeen tiedot tietokantaan. Listaus päivittyy tallennuksen jälkeen. Tallennuksen yhteydessä päivitetään lista syötetyistä tuloksista- Lista on toteutettu ASP.NET-alustan ListView-kontrollilla.

ListView-kontrolliin haetaan lista tuloksista Tulokset-luokasta, joka suorittaa tietokantakyselyn kutsumalla ”ListaaTulokset”-proseduuria. (Kuva 10.)

```
public class Tulokset
{
    public int ID { get; set; }
    public int KilpailuID { get; set; }
    public int EsteID { get; set; }
    public int ValjakkoNro { get; set; }
    public TimeSpan Aika { get; set; }
    public int Virhepisteet { get; set; }
    public string Ohjaaja { get; set; }

    public Tulokset(int ID, int kilpailuID, int esteID, int valjakkoNro, TimeSpan aika, int virhepisteet, string ohjaaja)
    {
        this.ID = ID;
        this.KilpailuID = kilpailuID;
        this.EsteID = esteID;
        this.ValjakkoNro = valjakkoNro;
        this.Aika = aika;
        this.Virhepisteet = virhepisteet;
        this.Ohjaaja = ohjaaja;
    }

    public static List<Tulokset> Listaatulokset()
    {
        List<Tulokset> tulokset = new List<Tulokset>();

        SqlConnection connection = new SqlConnection(ConfigurationManager.AppSettings["ConnectionString"]);
        SqlCommand command = new SqlCommand("ListaaTulokset", connection);
        command.CommandType = CommandType.StoredProcedure;
        connection.Open();

        SqlDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            Tulokset tulos = new Tulokset(reader.GetInt32(0), reader.GetInt32(1), reader.GetInt32(2), reader.GetInt32(3),
            tulokset.Add(tulos);
        }
        connection.Close();
        return tulokset;

    }
}
```

Kuva 10. Tulokset luokka.

Luokka palauttaa listan tuloksista, jotka yhdistetään tietokannassa kyseisessä taulussa esiintyvän kolumnin mukaisesti ListView-kontrolliin.(Kuva 11.)

```
<td id="ValjakkoNroTD" runat="server">
    <asp:Label ID="lblVajalkkoNro" runat="server" Text='<%# DataBinder.Eval(Container.DataItem, "ValjakkoNro") %>'></asp:Label>
</td>
```

Kuva 11. Tiedon yhdistäminen Listview-kontrolliin.

3.4.2 Tuloslista

Tulosten listauksessa listataan kaikki tulleet tulokset tulosjärjestyksessä. Tuloslistan pääasiallisena tarkoituksena on aikojen ja virhepisteiden muokkaus, jos jollain rivillä huomataan virhe. Kuvassa 12 on esitetty tuloslista.

Tuloslista

Valjakko	Ohjaaja	Aika	Virhepisteet	
4	Ohjaaja4	01:15:00	70	Päivitä Peruuta
4	Ohjaaja4	01:15:00	1	Muokkaa
2	Ohjaaja2	02:00:00	100	Muokkaa
2	Ohjaaja2	02:00:00	-100	Muokkaa
1	Ohjaaja1	02:00:00	-90	Muokkaa
3	Ohjaaja3	02:15:00	7	Muokkaa
2	Ohjaaja2	12:00:00	-78	Muokkaa
2	Ohjaaja2	12:00:00	-2	Muokkaa
2	Ohjaaja2	12:00:00.0000023	-2	Muokkaa
4	Ohjaaja4	12:00:23	4	Muokkaa

Kuva 12. Tuloslista muokkaustilassa.

Tulosten listaukseen on käytetty GridView-kontrollia joka toimii periaatteessa aivan samalla tavalla kuin ListView-kontrolli muutamia ominaisuuksia lukuun ottamatta. Tulosten listaukseen valittiin tässä tapauksessa GridView- kontrolli monipuolisemman lajittelu ominaisuuksien takia. Tulokset listataan ja yhdistetään kontrolliin samalla tavalla kuin tulosten syötössä.

Sekä tulosten syötössä, että tulosten listauksessa käytetään samaa proseduuria. (Kuva 13.)

```
ALTER PROCEDURE [dbo].[ListaaTulokset]

AS
BEGIN

    SET NOCOUNT ON;

    SELECT t.*, v.Ohjaaja AS Ohjaaja FROM Tulos t
    LEFT JOIN Valjakot v on t.ValjakkoNro = v.Valjakko_nro

    ORDER BY t.Aika
END
```

Kuva 13. ”ListaaTulokset”-proseduuri tietokannassa.

3.4.3 Tulosnäyttö

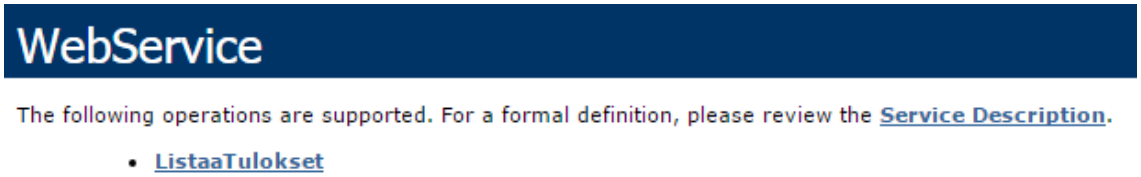
Tulosnäyttö on katsojille tarkoitettu näyttö, jossa tulokset voidaan näyttää reaaliaikaisesti. Tulosnäyttö on toteutettu ASP.NET-alustan MVC-mallilla, sekä JavaScriptin Knockout ja JQuery -kirjastoja hyödyntäen. Tulosnäytön toteutus oli ehdottomasti järjestelmän haasteellisin vaihe. Suurin ongelma oli tulosten haku tietokannasta ja päivittää haetut tulokset lomakkeelle sellaisessa muodossa, että tulosjoukkoa olisi helppo käsitellä. Reaaliaikaisen tulosnäytön toteuttamiseen käytin WebService-komponenttia, jolla kutsutaan tietokannan proseduuria. WebServicen otettua vastaan tietokannasta haetut tiedot, se käsittelee tiedot ja palauttaa listan tuloksista. Tulosnäyttö-kontrollissa kutsutaan WebService-komponenttia kolmenkymmenen sekunnin välein, riippumatta siitä onko uusia tuloksia syötetty palveluun. Kutsu tapahtuu jQuery-kirjaston AJAX-metodilla joka muuttaa vastaanotetun tiedon JSON-muotoon mikä helpottaa tulosjoukon käsittelyn. Seuraavaksi tullaan havainnollistamaan prosessin eri vaiheet kuvin.

Kuvassa 14 on esitetty WebService, jossa kutsutaan ”NaytaTulokset”-proseduuria tietokannasta. WebService käsittelee tiedot ja palauttaa lopuksi listan tuloksista.

```
public class WebService : System.Web.Services.WebService
{
    public class Tulokset
    {
        public int ValjakkoNro { get; set; }
        public string Ohjaaja { get; set; }
        public string Aika { get; set; }
        public int Virhepisteet { get; set; }
    }

    [WebMethod]
    public List<Tulokset> ListaaTulokset(List<string> aData)
    {
        SqlDataReader dr;
        List<Tulokset> TulosLista = new List<Tulokset>();

        using (SqlConnection con = new SqlConnection(ConfigurationManager.AppSettings["ConnectionString"]))
        {
            using (SqlCommand cmd = new SqlCommand())
            {
                cmd.CommandText = "NaytaTulokset";
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Connection = con;
                con.Open();
                dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
                if (dr.HasRows)
                {
                    while (dr.Read())
                    {
                        int valjakkoNro = Convert.ToInt32(dr["ValjakkoNro"]);
                        string ohjaaja = dr["Ohjaaja"].ToString();
                        string aika = dr["Aika"].ToString();
                        int virhepisteet = Convert.ToInt32(dr["Virhepisteet"]);

                        TulosLista.Add(new Tulokset
                        {
                            ValjakkoNro = valjakkoNro,
                            Ohjaaja = ohjaaja,
                            Aika = aika,
                            Virhepisteet = virhepisteet
                        });
                    }
                }
            }
        }
        return TulosLista;
    }
```

Kuva 14. WebService-komponentti.

Selaimella voidaan katsoa onko WebService toiminnassa. (Kuva 15.)



Kuva 15. WebService selaimella.

Tulosnäyttö-kontrollissa kutsutaan WebService-komponenttia kolmenkymmenen sekunnin välein. Jos WebService-komponentista on haettu tiedot onnistuneesti, käsiteellään tulosjoukko ja yhdistetään kukin tulosjoukon rivi Knouckout-kirjastoa hyödyntäen MVC-malliin.

```
var TuloksetVM = function () {

    var self = this;
    self.Tulokset = ko.observableArray();

    self.HaeData = function () {

        var aData = [];
        $("#contentHolder").empty();
        var jsonData = JSON.stringify({ aData: aData });
        $.ajax({
            type: "POST",
            url: "WebService.asmx/ListaaTulokset",
            data: jsonData,
            contentType: "application/json; charset=utf-8",
            dataType: "json",
            success: OnSuccess
        });

        function OnSuccess(response) {
            var items = response.d;
            $.each(items, function (index, val) {

                aData.push(new TulosVM(val));
            });
            self.Tulokset(aData);
        }
        setTimeout(function () {
            self.HaeData();
        }, 30000);
    }
    self.HaeData();
}

var TulosVM = function (params) {

    var self = this;
    self.ValjakkoNro = ko.observable(params.ValjakkoNro || '');
    self.Ohjaaja = ko.observable(params.Ohjaaja || '');
    self.Aika = ko.observable(params.Aika || '');
    self.Virhepisteet = ko.observable(params.Virhepisteet || '');
}

var tuloksetvm = new TuloksetVM();
ko.applyBindings(tuloksetvm);
```

Kuva 16. WebService-komponentista vastaanotetun tulosjoukon käsittely.

Kuvassa 17 on esitetty sovelluksessa käytetty MVC-malli, johon yhdistetään tulokset rivi kerrallaan.

```
<div class="tulos">
    <h1>
        <asp:Label ID="LBL_lomakeotsikko" runat="server" Text="Tulosnäyttö"></asp:Label>
    </h1>
    <table>
        <thead>
            <tr>
                <th>Valjakko</th>
                <th>Ohjaaja</th>
                <th>Aika</th>
                <th>Virhepisteet</th>
            </tr>
        </thead>

        <tbody data-bind="foreach: Tulokset">
            <tr>
                <td data-bind="text: ValjakkoNro"></td>
                <td data-bind="text: Ohjaaja"></td>
                <td data-bind="text: Aika"></td>
                <td data-bind="text: Virhepisteet"></td>
            </tr>
        </tbody>
    </table>
</div>
```

Kuva 17. MVC-malli sovelluksessa.

Selaimen Web Developer -työkalulla voidaan seurata asiakaspuolen päässä kolmenkymmenen sekunnin välein tapahtuvia WebService kutsuja. (Kuva 18.)

URL	Status	Domain
⊞ GET TulosNaytto.aspx	200 OK	localhost:49381
⊞ GET knockout-3.4.0.js	304 Not Modified	localhost:49381
⊞ POST ListaaTulokset	200 OK	localhost:49381
⊞ POST ListaaTulokset	200 OK	localhost:49381
⊞ POST ListaaTulokset	200 OK	localhost:49381
⊞ POST ListaaTulokset	200 OK	localhost:49381

Kuva 18. WebService-kutsut selaimella.

4 YHTEENVETO

Projektin tavoitteena oli toteuttaa toimiva selainpohjainen, tietokantaa hyödyntävä sovellus. Vaikka sovelluksen kaikkia pakollisia ominaisuuksia ei saatu valmiiksi tämän opinnäytetyön aikana, oleellisimmat ominaisuudet tulivat valmiiksi. Kehitystä tullaan jatkamaan pakollisten ja vaihtoehtoisten ominaisuuksien osalta.

Projektin alkuvaiheessa oli selvää, kuinka sovellus tullaan toteuttamaan ja kokonaisuus oli hahmotettuna ennen opinnäytetyön aloittamista. Toimeksiantajan hyvät määrittelyt, oma aikaisempi osaaminen tietojärjestelmien ja tietokantojen suunnittelusta sekä entuudestaan tutut Microsoftin työkalut helpottivat huomattavasti projektin etenemisprosessia.

Ongelmia ja haasteita tuli vastaan opinnäytetyön aikana, joista suurimpana haasteena oli toteuttaa reaaliaikaisesti päivittyvä tulosnäyttö, joka toteutettiin hyödyntäen MVC-mallia ja asiakaspuolen ohjelmointikieliä.

Tulevaisuudessa kehitystä ajatellen, tärkein ominaisuus on toteuttaa responsiiviset sivut mobiilikäyttöä varten. Kaikkia sivuston kontrolleja ei tarvitse tässä huomioida, ainoastaan tulosten syöttö ja tulosnäyttö optimoitava.

LÄHTEET

Anderson, R. 2014. Getting Started with ASP.NET MVC 5. Viitattu 8.12.2015 http://www.asp.net/mvc/overview/getting-started/introduction/getting-started

ASP.NET–Introduction. Viitattu 1.12.2015. https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm

Atwood, J. 2008. Understanding Model-Wiev_Controller. Viitattu 1.12.2015. https://blog.codinghorror.com/understanding-model-view-controller/

Chauhan, S. 2013. Understanding Detailed Architecture of ASP.NET 4.5. Viitattu 1.12.2015 http://www.dotnet-tricks.com/Tutorial/aspnet/SaJc221013-Understanding-Detailed-Architecture-of-ASP.NET-4.5.html

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. 11. painos. Talentum Oyj.

Hejslsberg, A. & Wiltamuth, S. & Golde, P. 2004. The C# Programming Language. Addison-Wesley.

HTML5. Wikipedia. Viitattu 1.12.2015 https://fi.wikipedia.org/wiki/HTML5

Knockout – MVVM Framework. Viitattu 6.12.2015 http://www.tutorialspoint.com/knockoutjs/knockoutjs_mvvm_framework.htm

Knockout Introduction, Observable Arrays. Viitattu 5.12.2015 http://knockoutjs.com/documentation/observableArrays.html

Knockout Introduction. Viitattu 1.12.2015. http://knockoutjs.com/documentation/introduction.html

Marshall, A. 2012. Introduction to using jQuery with Web Services. Viitattu 8.12.2015 http://www.codeproject.com/Articles/349598/Introduction-to-using-jQuery-with-Web-Services

Marufuzzaman, M. 2010. Overview of SQL Server Strored Procedure. Viitattu 2.12.2015. http://www.codeproject.com/Articles/38682/Overview-of-SQL-Server-Stored-Procedure#WhatIsSP

Microsoft SQL Server. Wikipedia. Viitattu 12.12.2015 https://en.wikipedia.org/wiki/Microsoft_SQL_Server

Microsoft SQL Server. Wikipedia. Viitattu 12.12.2015 https://en.wikipedia.org/wiki/Microsoft_SQL_Server

Quassnoi. 2009. What is entity-relationship model?. Viitattu 10.12.2015. https://explainextended.com/2009/10/18/what-is-entity-relationship-model/

Ragget, D. W3C, Client-side Scripting and HTML. Viitattu 5.12.2015 http://www.w3.org/TR/WD-script-970314

Ward, D. Using jQuery to Consume ASP.NET JSON Web Services. Viitattu 8.12.2015 http://encosia.com/using-jquery-to-consume-aspnet-json-web-services/