SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

AJANSEURANTAJÄRJESTELMÄ

Web-sovelluskehitys ASP.NET MVC -tekniikalla

TEKIJÄ/T: Joni Rautiainen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Joni Rautiainen	
Työn nimi Ajanseurantajärjestelmä	
Päiväys 18.2.2014	Sivumäärä/Liitteet 32
Ohjaaja(t) Lehtori Jussi Koisitnen	
Toimeksiantaja/Yhteistyökumppani(t) Data Prisma Oy	

Tiivistelmä

Opinnäytetyön aiheena oli tehdä Data Prisma Oy:lle työntekijöiden leimaussovellus. Tilaaja on kuopiolainen ohjelmistotalo.

Sovelluksen tarkoituksena oli tallentaa työntekijän sisäänkirjautumiset ja uloskirjautumiset asiakas-, vuoro-, projekti- ja toimintokohtaisesti. Sen avulla voidaan seurata projektiin kuluvia tunteja ja laskuttaa asiakkaita tuntien mukaan. Leimaussovellus tulee tilaavan yrityksen sekä yrityksen asiakkaiden käyttöön. Sovellus haluttiin kehittää web-ympäristöön käytettäväksi tietokoneilla sekä mobiililaitteilla.

Ohjelmistokehityksessä työvälineinä käytettiin Microsoft Visual Studio Professional 2012:ta. Ohjelmointikielenä käytettiin C#-kieltä ja ASP.NET:n tarjoamaa MVC-ohjelmistokehystä. Sovelluksen tietokantana käytettiin Microsoft SQL Serveriä. Käyttöliittymäsovelluksen lisäksi tehtiin myös palvelinpuolen sovellus, jonne tehtiin tarvittavat rajapinnat, jotta käyttöliittymäsovellus voi päivittää, hakea ja tallentaa tietoa.

Tuloksena valmistui toimiva versio, joka jäi yrityksen testattavaksi. Sovellusta ei ennätetty ottaa käyttöön testaamisen puutteen vuoksi. Sovelluskehitystä on tarkoitus jatkaa kesällä 2014, jotta uusi leimaussovellus saadaan yrityksen käyttöön ja myyntiin tilaajan asiakkaille.

Avainsanat

MVC, Kello, leimaus, Microsoft, C#, Web, WWW

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Joni Rautiainen			
Title of Thesis Employee login software			
Date	18 February 2014	Pages/Appendices	32
Supervisor(s) Mr Jussi Koistinen, Lecturer			
Client Organisation /Partners Data Prisma Oy			

Abstract

The purpose of this thesis was to create employee login software. The commissioner of the software was a company called Data Prisma Ltd.

The software can save an employee working hours. The application saves the employee's choices from user interfaces. The choices are customer, shift, line, product and function. The company can follow how many hours employee has worked on a project and the company can bill the customer accordingly. The application was developed as a www-application suitable for computers and mobile devices.

The application was made with Microsoft Visual Studio Professional 2012 software. The programming language was the C# language and the ASP.NET framework called MVC technology. The application uses the Microsoft SQL Server Database. The work included both the server and client side software.

As a result of this thesis working software was created. All functions and tasks were created but all of the tests were not done by the writing of this thesis.

Keywords

MVC, Clock, WWW, Web, Microsoft, C#

SISÄLTÖ

TERMIT JA LYHENTEET

ASP.NET MVC	Web-sovelluskehitykseen tarkoitettu ohjelmistokehys, joka käyttää MVC-mallia
C#	Microsoftin kehittämä ohjelmointikieli
SQL	Kyselykieli, jolla voi tehdä hakuja, lisäyksiä ja muutoksia relaatiotieto kantaan
.NET	Microsoftin ohjelmistokomponenttikirjasto
LINQ	SQL-kyselykielen tapainen ohjelmointisyntaksi
ENTITY FRAMEWORK	Mahdollistaa tietokannan käsittelyn oliomaisesti
JQUERY	JavasScript-kirjasto
JAVASCRIPT	Oliopohjainen ohjelmointikieli, jolla lisätään dynaamista tietoa www-sivuille.
RAZOR	Microsoftin kehittämä ohjelmointi syntaksi, jota käytetään dynaamisien web-sivujen luomiseen C# tai Visual Basic kielillä.
HTML	www sivunkuvaus kieli
CSS	www sivujen tyylit
IIS	Palvelinohjelmistokokonaisuus, jonka alla web-sovellus pyörii.

1 JOHDANTO

Opinnäytetyön tavoitteena on tehdä työntekijän työnajankirjausohjelma. Leimaussovelluksen tarkoitus ei ole ainoastaan työntekijöiden työaikojen seuranta vaan sovelluksen tarkoitus on myös seurata projekteja, tuotteita, tehtäviä ja asiakkaita. Sovelluksen avulla yritys voi esimerkiksi laskuttaa asiakkaita tai katsoa, miten paljon aikaa projektiin on kulunut. Se palvelee muun muassa yritysjohtoa, tuotannonseurantaa, hallintoa ja palkanlaskentaa. Leimausjärjestelmä on suunnattu pienille ja keskisuurille yrityksille.

Opinnäytetyön tilaaja on Data Prisma Oy. Yrityksellä on jo käytössä olemassa oleva Kellokas-leimaussovellus Windows-työpöytäversiona. Sovellus vaatii uudistusta ja lisää toivottuja ominaisuuksia. Näitä toivottuja ominaisuuksia ovat muun muassa ylityötuntien sekä lomapäivien seuraaminen, valikoiden latautuminen asiakaskohtaisesti ja tiedon löytäminen käyttöliittymästä.

Aihe tuli yrityksen tarpeesta, koska se haluaa uudistaa vanhan käyttöliittymän web-pohjaiseksi ja helposti saatavilla olevaksi. Vanhan kellokas sovelluksen tietokantaa voidaan käyttää hyödyksi uudessa Kellokas-sovelluksessa. Koska vanhan sovelluksen palvelinpuolen sovellusta ei voida käyttää, pitää tämäkin tehdä. Sovellukseen tulee neljäkerroksinen arkkitehtuurirakenne rajapintoineen ja metodeineen.

Opinnäytetyössä selvitetään sovelluksen kehittämisessä käytetyt tekniikat ja työvälineet ja kuinka sovellus suunniteltiin ja toteutettiin. Lisäksi esitellään sovelluksen päätoiminnot. Lopussa kerrotaan, kuinka työ testattiin yhdessä testaajien kanssa, ja pohditaan työn onnistumisesta.

2 DATA PRISMA OY

Data Prisma Oy on kuopiolainen yritys, jonka perusti kolme henkilöä vuonna 1987. Yritys on ohjelmistotalo, joka suunnittelee ja toteuttaa pääasiassa asiakkaan vaatimuksien mukaan tietokonejärjestelmiä. Heidän palveluita ovat muun muassa asiakaskohtainen räätälöinti, asiakaspalautejärjestelmä asiakastyytyväisyyden kartoittamiseen, laite- ja verkkoasennukset.

Data Prisman tuotteita ovat Vision, Kongressi, Jäsenrekisteri ja Kellokas. Yrityksen päätuote on palvelutelevision hallinta-, markkinointi- ja laskutusohjelmisto Vision. Sen tarkoitus on palvelutelevision päivittäisten rutiineiden hoitoon. Kongressi on erilaisten tapahtumien luennoitsijoiden, tilojen ja talouden hallintaan. Se sisältää monipuolisia lisämoduuleita tapahtumien suunnitteluun ja toteutukseen. Jäsenrekisteri on jäsentietojen hallintasovellus. Siihen voidaan tallentaa tietoa jäsenistä ja kertynyttä tietoa voidaan käyttää raporttien muodossa. Kellokas on ajanseurantajärjestelmä yrityksille ja sitä käytetään palkanlaskennan, urakoiden, projektien ja tuotteen tuntilaskennan pohjana. (Data Prisma Oy.)

3 KÄYTETYTY TEKNIIKAT JA TYÖKALUT

Sovelluksesta tehdään web-sovellus, joka on yhdellä asennuksella saatavilla yrityksen asiakkailla. Ohjelma toteutetaan Microsoft-tuoteperheen sovelluksilla. Ohjelmointikielenä käytetään C#-kieltä ja ASP.NET:n tarjoamaa Model-View-Controller-ohjelmistokehystä. Sovelluksessa otetaan huomioon myös Javascript-kieli ja sen tarjoamat kirjastot, kuten esimerkiksi jQuery. JQueryn käytössä otetaan huomioon sen tehokkuus ja yritetään välttää Javascriptin tuoma syntaksin virheellisyys. Sovellus tulee pyörimään IIS-palvelun alla yrityksen palvelimella.

Tilaaja on siirtynyt kokonaisuudessaan kaikissa tuotteissaan web-sovelluksiin, joten vanhasta kellokkaasta on tarpeen päivittää uusi versio. Sovelluksessa käytetään vanhan sovelluksen tietokantaa, mutta rajapinnat tehdään uusiksi yrityksen arkkitehtuurin mukaisesti. Tietokannasta tehdään palvelinsovelluksen puolelle Entity-malli, jota voidaan käyttää rajapinnassa oliomaisesti tiedon hakemiseen, poistamiseen ja tallentamiseen.

3.1 Microsoft Visual Studio 2012

Microsoft Visual Studio on Microsoftin sovelluksien kehitysympäristötyökalu. Sovelluksissa voi käyttää useita eri ohjelmointikieliä, kuten esimerkiksi C#:a, C:tä, C++:aa ja Visual Basiciä. Sillä voi tehdä konsolipohjaisia tai graafisia käyttöliittymäsovelluksia, kuten esimerkiksi Windows-työpöytäsovelluksia (Windows Forms tai WPF-sovelluksia), internetsivuja, Windows-kaupan sovelluksia ja mobiiliapplikaatioita. Jos sovelluksesta ei löydy jotain kaipaamaa ominaisuutta, saa siihen myös paljon lisäosia ja laajennuksia. (MSDN Microsoft.)

3.2 Microsoft SQL Server Management Studio 2012

SQL Server Management Studio (ssms) on sovellusohjelma, joka julkaistiin ensimmäisen kerran vuonna 2005 Microsoft SQL Serverin yhteydessä. Sitä käytetään konfigurointiin, hallintaan ja se sisältää kaikki järjestelmänvalvojan komponentit. Työkalulla otetaan yhteys tietokantoihin, ja sillä voidaan muun muassa hakea, poistaa, muokata ja lisätä tietokannan tauluihin tietoa, lisätä tietokantoja ja tietokannan tauluja. (Technet Microsoft.)

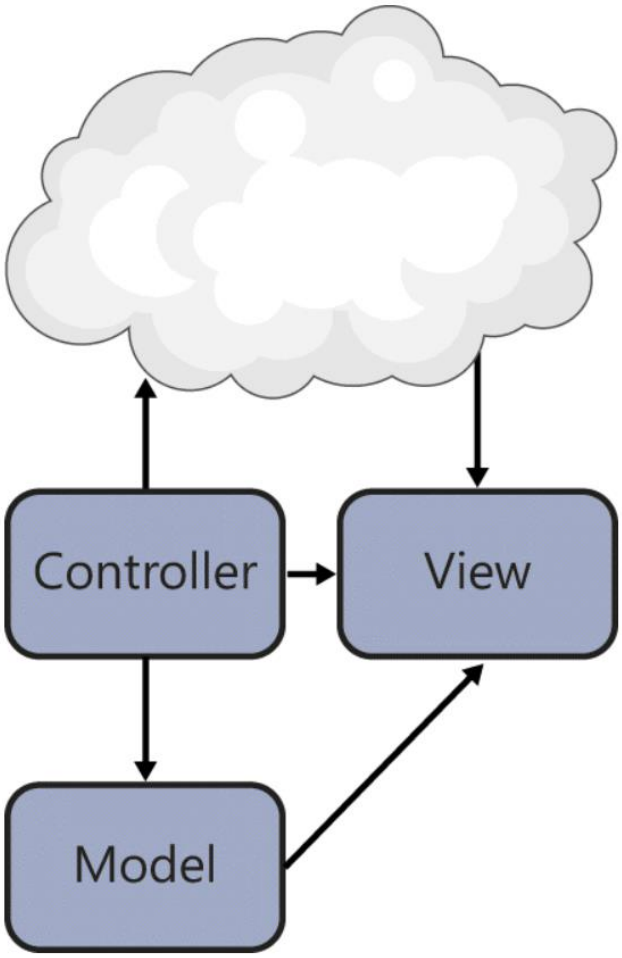
3.3 IIS – Internet Information Services

IIS on laajennus Microsoftin web-palvelimelle, jota käytetään Windows NT-tuoteperheessä. Laajennus tukee HTTP, HTTPS, FTP, FTPS, SMTP ja NNTP -protokollia. IIS on pois käytöstä oletuksena kaikissa Microsoft Windowseissa. Sovellus on kirjoitettu C++–ohjelmointikielellä. Uusin versio on IIS 8.5, joka on sisälletty Windows 8.1 ja Windows Server 2013 R2 -versioihin. (TechTarget.)

3.4 MVC

MVC-arkkitehtuuri tulee sanoista Model-View-Controller. Se on ohjelmistoarkkitehtuurityyli, jonka tarkoituksena on erottaa sovelluksen tieto käyttöliittymästä. Sitä käytetään graafisten käyttöliittymien kehityksessä.

MVC-arkkitehtuuriohjelma jaetaan kolmeen osaan, joita ovat malli, näkymä ja käsittelijä. Alla olevassa kuviossa (KUVIO 1) on esitetty nuolilla, kuinka nämä model-view-controller-kerrokset keskustelevat keskenään. (ASP.NET MVC 4.)



KUVIO 1 MVC-malli

3.4.1 Malli (Model)

Malli on ohjelmointikielessä oleva luokka, joka kuvaa järjestelmän tietojen ylläpidon, käsittelyn ja tallentamisen. Yleisesti luokka sisältää ominaisuuksia ja siihen voidaan kiinnittää tietoa. Malli on yksi osa ohjelmasta, mikä käsittelee bisneslogiikan ja suorittaa sen tiedolle käsittelijässä. Malli ei peri mitään eri rajapintoja tai johda mistään kantaluokasta. Mallit ovat ASP.NET MVC-tekniikassa Model-kansiossa. Tiedon säilyttäminen malleissa tekee mallin jakamisesta helpompaa, koska malleja voidaan käyttää useassa ohjelmassa. Malleissa voidaan hoitaa validointi, joka tarkoittaa merkkijonojen pituuksien rajoituksia, pakollisuustietoa ja kokonaislukujen arvovälejä. MVC-arkkitehtuuri tarjoaa yk-

sinkertaisen tavan malliin kytkemisen (Model Bind) lomakkeen lähetyksessä, joka menee parametrinä käsittelijälle. ”DefaultModelBinder” kytkee lomakkeiden syötteet automaattisesti mallien eli luokkien ominaisuuksiin, jos nimet vastaavat täysin lomakkeen syötteisiin ja luokkien ominaisuuksiin. MVC tarjoaa myös hyödyllisiä tapoja luoda kustomoituja malliin kiinnittämisiä, koska ”DefaultModelBinder” ei tue abstraktiluokkia tai rajapintoja. Alla olevassa kuviossa (KUVIO 2) on esimerkki Customer-luokasta eli mallista, jossa luokan ominaisuuksille on annettu eri validointisääntöjä, joilla rajoitetaan käyttäjän syötteitä ja näytetään kenttien näyttönimet kenttien vieressä käyttäjille. (ASP.NET MVC 4.)

```
public class Customer
{
    [Range(1, int.MaxValue, ErrorMessage = "Pituus menee yli")]
    public int CustomerId { get; set; }

    [Required(ErrorMessage = "Etunimi on pakollinen tieto")]
    [Display(Name="Etunimi")]
    public string FirsName { get; set; }

    [Required(ErrorMessage = "Sukunimi on pakollinen tieto")]
    [Display(Name = "Sukunimi")]
    public string LastName { get; set; }

    [Display(Name = "Osoite")]
    public string Address { get; set; }

    [Display(Name = "Kaupunki")]
    public string City { get; set; }

    [StringLength(5, ErrorMessage="Postinumeron pituus on 5 merkkiä")]
    [Display(Name = "Postinumero")]
    public string PostalCode { get; set; }
```

KUVIO 2 Esimerkki yksinkertaisesta model-luokasta.

3.4.2 Näkymä (View)

Näkymä on vastuussa sovelluksen tiedon näyttämisestä käyttäjälle. Se on ainut osa, jonka käyttäjä näkee. Näkymästä viitataan malliin, jolla tieto saadaan näytettyä. Mallien tiedot täytetään käsittelijässä, joka palautetaan näkymälle. Näkymät sisältävät normaaleja HTML-kielen tageja sekä myös MVC-arkkitehtuurin Razor-syntakseja, joilla voidaan hyödyntää edellä mainittua malliin kiinnittämistä.

Visual Studiossa voidaan luoda seuraavanlaisia malleja:

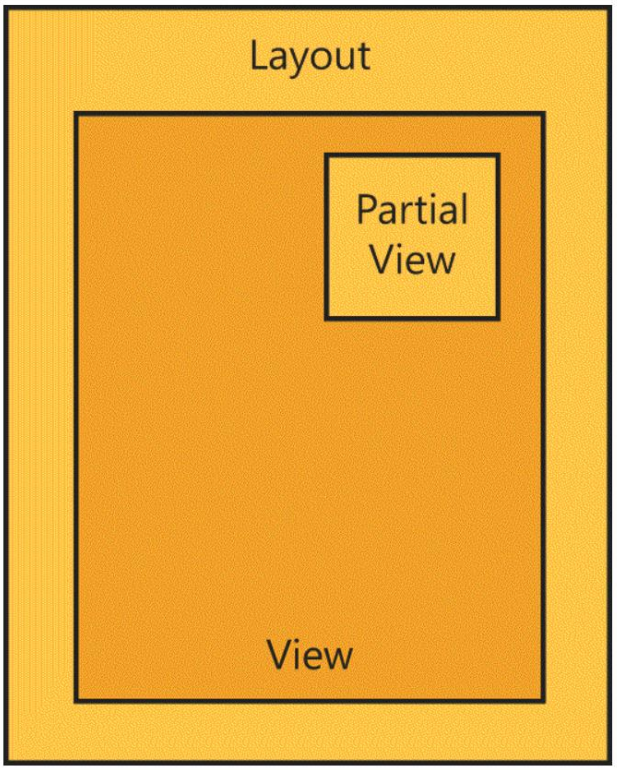
- Strongly-typed views asetuksen avulla saadaan valittua malli eli luokka näkymälle.

- Scaffold template asetuksen avulla saadaan kehittäjän valitseman luokan avulla nopeasti generoitua luokan ominaisuudet ja tiedot näkymälle tietolistana tai lomakkeen syötteinä käyttäjän tarpeiden mukaan.

- Partial View on sivun yksittäinen osa, joka voidaan kirjoittaa omaan tiedostoon ja näyttää se näkymällä, joka voi sisältää erilaisia toimintoja kuten lomakkeen tai muun listan tiedoista.

- Master sivu tai pohja on sivu, jonka avulla ulkonäkö voidaan toteuttaa kaikille sivuille kirjoittamalla ohjelmointikoodi yhteen tiedostoon.

Alla olevassa kuviossa (KUVIO 3) on havainnollistettu käyttöliittymäsivun rakenne.

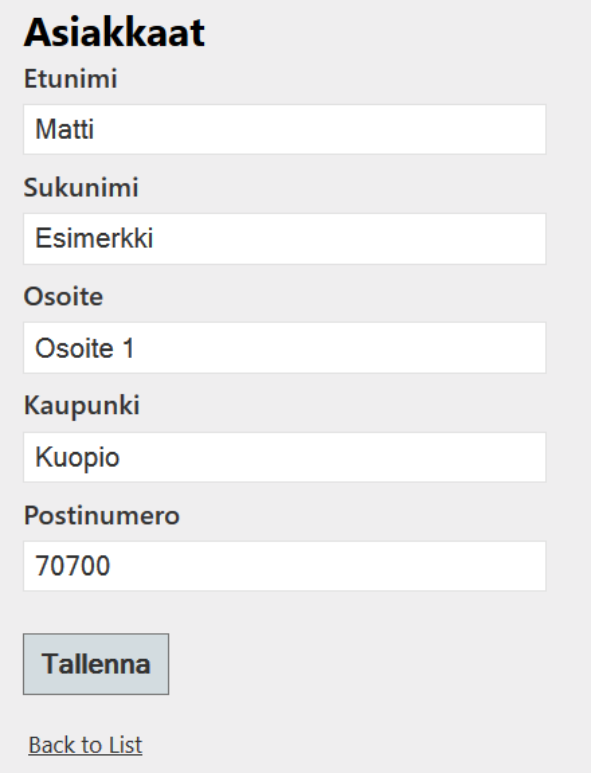


KUVIO 3 Käyttöliittymäsivun rakenne.

Alla olevassa kuviossa (KUVIO 4) on esimerkki HTML-sivusta, jonka käyttäjä näkee. Sivun ylhäällä otetaan mukaan luokka, joka täytetään käsittelijän puolella. Käyttäjälle näytetään sivun otsikko, minkä jälkeen tehdään lomake, johon kaikki model-luokan ominaisuudet tulostuvat tekstinä ja syöttökenttinä. Asiakkaan yksilöivä id-kenttä kulkee piilokentässä, jota käyttäjä ei nää käyttöliittymässä ilman lähdekoodin tarkastelua. Syöttökenttien alapuolelle syntyy validointia varten virhetekstit, jotka näytetään virheen syntyessä. (ASP.NET MVC 4.)

```
@model MvcApplication1.Models.Customer
@{
    ViewBag.Title = "Asiakkaat";
}
<h2>Asiakkaat</h2>
@using (Html.BeginForm()) {
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true)

    <fieldset>
        <legend>Asiakas</legend>

        @Html.HiddenFor(x => x.CustomerId)

        <div class="editor-label">
            @Html.LabelFor(model => model.FirsName)
        </div>
        <div class="editor-field">
            @Html.EditorFor(model => model.FirsName)
            @Html.ValidationMessageFor(model => model.FirsName)
        </div>

        <div class="editor-label">
            @Html.LabelFor(model => model.LastName)
        </div>
        <div class="editor-field">
```

KUVIO 4 Esimerkki HTML-sivusta.



KUVIO 5 Esimerkki käyttäjän käyttöliittymästä.

3.4.3 Käsittelijä (Controller)

Käsittelijä on luokka, joka hoitaa sovelluksen kaiken logiikan. Kun web-sovelluksen aukaisee, se menee ensimmäistä kertaa käyttäjän käsittelijäluokan metodiin eli niin sanottuun toimintometodiin (ActionResult). MVC-mallin Actionresult metodissa voidaan hakea tietoa tietokannasta tai kutsua omia metodeja. Näissä Actionresult metodeissa tieto kiinnitetään model-luokkiin, joka palautetaan edelli-

sessä otsikossa olevalle näkymälle. Käsittelijä (Controller) keskustelee näin ollen mallin ja näkymän kanssa. Alla olevassa kuviossa (KUVIO 6) on esimerkki käsittelijän Actionresult-metodista, jossa asetetaan MVC-tarjoamaan ViewBag:iin sivun otsikko, tehdään Customer-mallista olio, täytetään se asiakkaan tiedoilla ja palautetaan näkymälle eli HTML-sivulle. Actionresult-metodista voidaan palauttaa erityyppisiä objekteja, mutta yleensä palautetaan actionresult-luokasta periytyvä objekti. (ASP.NET MVC 4.)

```
public ActionResult Customers()
{
    ViewBag.Message = "Asiakkaat";

    Customer Person = new Customer();

    Person.FirsName = "Matti";
    Person.LastName = "Esimerkki";
    Person.City = "Kuopio";
    Person.Address = "Osoite 1";
    Person.PostalCode = "70700";

    return View(Person);
}
```

KUVIO 6 Actionresult Customers.

Kaikkiin Actionresult-metodeihin pääsee osoiteriviltä käsiksi. Jos käsittelijän nimi on Information-Controller ja Actionresultin nimi on Customers, osoiteriville pitää kirjoittaa testiosoite.fi/Information/Customers. Actionresultien nimi on oletuksena metodin nimi, mutta sen pystyy vaihtamaan laittamalla actionresult-kohdan yläpuolelle attribuutin [ActionName(”Asiakkaat”)]. Jos osoiteriviltä ei haluta päästä metodeihin käsiksi, pitää ne määrittää privateiksi tai laittaa metodin yläpuolelle attribuutti [NonAction].

3.5 JavaScript

JavaScript on olio-ohjelmointi komentosarjakieli syntaksiltaan lähelle C-ohjelmointikieltä. JavaScriptiä käytetään Web-ympäristöissä. Sen ideana ja valttina on, että web-sisältöä saadaan lisättyä sivuille dynaamisesti. Kielen etuna on myös, että se on alustariippumaton. Kaikki JavaScript-kieli kirjoitetaan <script></script>-tagien sisään. Alla olevassa kuviossa (KUVIO 7) on esitelty JavaScript-funktio, jossa etsitään html-elementti id:n perusteella ja lisätään elementtiin dynaamisesti sisältöä. kuviossa (KUVIO 7) näkyy myös Razor-syntaksi, jota käytetään käsittelijältä palauttaman tiedon näyttämiseen. (Ohjelmointiputka.)

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")

    <script>
        function AddHtmlTable()
        {
            var table = document.getElementById("table-content");

            table.innerHTML = "<table><tr><td>@Model.FirsName</td>"+
                "<td>@Model.LastName</td></tr></table>";
        }

    </script>

}
```

KUVIO 7 Esimerkki yksinkertaisesta javascript funktiosta.

3.6 JQuery

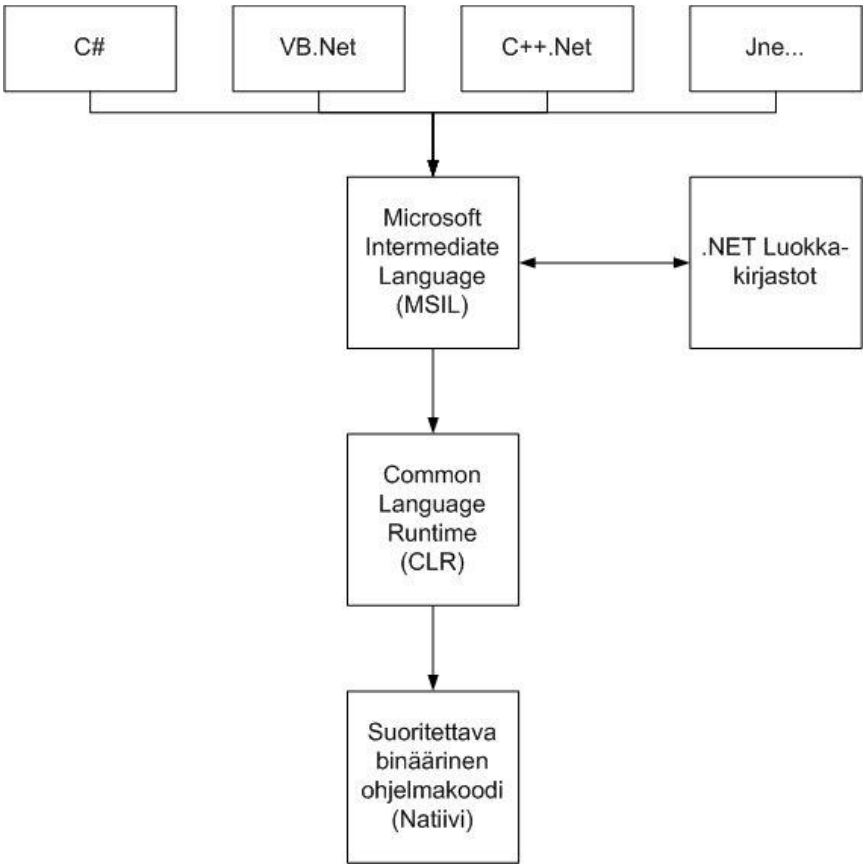
JQuery on erillinen JavaScript-tiedosto, joka sisältää paljon kaikkea toimintoja, jota voidaan käyttää nopeasti ja kätevästi web-sivuilla. Tiedosto sisältää paljon ulkoasuun liittyviä toimintoja, validointiin ja muihin toimintoihin. JQuerya voidaan yhtä hyvin sekoittaa javascriptin kanssa. (W3Schools.)

```
function ElementVisibility(id)
{

    var element = document.getElementById(id);

    if ($(element).hidden) {

        $(element).show(10);
    }
    else {
        $(element).hide(1);
    }
}
```

KUVIO 8 Id:n perusteella elementin piilottaminen

3.7 C#-kieli (C sharp)

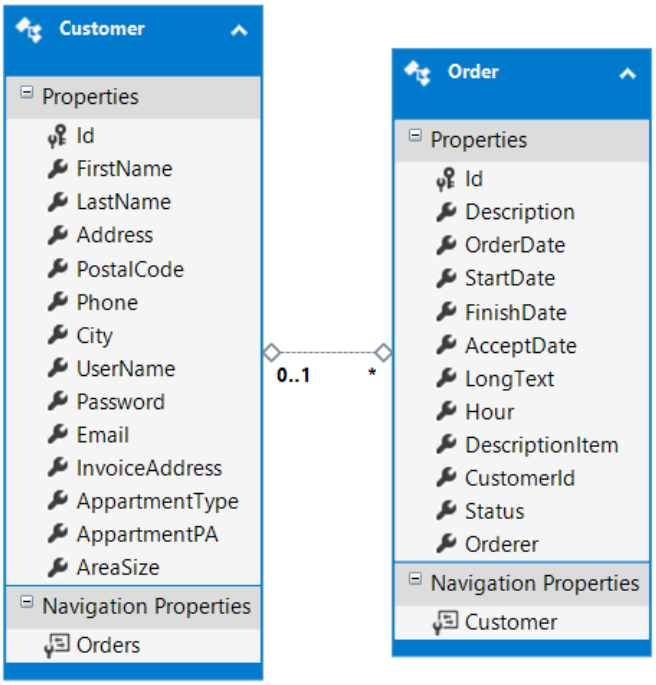
Microsoftin kehittämä C# on ohjelmointikieli, joka julkaistiin vuonna 2000. Se on kehitetty .NET-konseptia varten ja sen tarkoituksena oli yhdistää C++:n tehokkuus ja Javan-kielen helppokäyttöisyys. (MSDN Microsoft.)



KUVIO 9 .NET konsepti

3.8 Entity Framework

Entity Framework (EF) on tietokantojen ohjelmointitapa, joka on suunniteltu .NET Frameworkia varten, joka perustuu ORM-kehykseen (Object-relational mapping). Muita tunnettuja ORM:eja ovat muun muassa LINQ to SQL ja NHibernate. ORM:n tarkoitus on, että se esittää tietokantaan tallennetun tiedon loogisen mallin käsitteellisenä mallina ohjelmalle. Entity Frameworkin vahvuutena on sen siirrettävyys ja hyvä laajennettavuus. Se yhdistää relaatiomallin ja XML-kielen. (Codeplex 2013-08-31.)



KUVIO 10 Entity-malli.

```
var query = from c in customer.Customer
            where c.Id == CustomerId
            select new Customer
            {
                Id = c.Id,
                FirstName = c.FirstName,
                LastName = c.LastName,
                Address = c.Address,
                PostalCode = c.PostalCode,
                Phone = c.Phone,
                City = c.City,
                UserName = c.UserName,
                Password = c.Password,
                Email = c.Email,
                InvoiceAddress = c.InvoiceAddress,
                ApparmentType = c.AppartmentType,
                ApparmentPA = c.AppartmentPA.HasValue ? c.AppartmentPA.Value : 0d,
                AreaSize = c.AreaSize.HasValue ? c.AreaSize.Value : 0d

            };
```

KUVIO 11 Esimerkki Linq to Sql-tiedon hakeminen asiakastaulusta asiakasid:n perusteella.

3.9 LINQ

LINQ on .NET-kehyksessä mukana tullut SQL-kyselykielen tapainen ohjelmointisyntaksi. Sitä voidaan käyttää muun muassa tauluissa, enumeerisissa luokissa, XML-dokumenteissa, tietokantatauluissa tai geneerisissä listoissa. (MSDN Microsoft.)

```
var name = customer.Customers.
    Select(x => new { Name = x.FirstName + " " + x.LastName })
    .FirstOrDefault();
```

KUVIO 12 Linq-anonyymityyppi haku.

4 PROJEKTIN MÄÄRITTELY, SUUNNITTELU JA TOTEUTUS

4.1 Määrittely

Yritykseltä oli olemassa jo vastaavanlainen sovellus, ja jota tekijä oli itsekin käyttänyt, joten oli helppo lähteä suunnittelemaan ja toteuttamaan uutta versiota. Käyttäjäkokemuksilla saadaan hyvin parannettua uutta sovellusta. Käyttäjäkokemuksien mukaan havaittiin muutamia käyttöongelmia ja häiritseviä tekijöitä päivittäisessä käytössä. Tässä työssä ei uudistettu sovelluksen järjestelmänvalvojan hallintatyökalua, vaan yritys aikoo käyttää vanhaa sovellusta asiakkaiden, tuotteiden, käyttäjien ja muiden tarvittavien tietojen lisäämiseen.

4.2 Suunnittelu

Suunnittelussa oli tärkeä selvittää, miten voidaan parantaa vanhaa hyvin toimivaa sovellusta, poistaa häiritsevät tekijät ja tehdä käyttöliittymästä helppokäyttöisempi. Koska sovelluksesta tehtiin web-sovellus, oli sovelluksen tärkeänä osana sen tietoturva, jottei kuka tahansa pääse tekemään tuhoja ja syöttämään väärää tietoa. Vanhassa sovelluksessa ei ollut minkäänlaista salasanalla kirjautumista, mutta web-sovellukseen sellainen oli pakko tehdä, koska ohjelma pyörii muualla kuin sisäverkossa. Web-sovelluksen hyöty on, että sovelluksen käyttö ei ole riippuvainen tietystä paikasta ja tietokoneesta.

Sovelluksen avulla työntekijä vaihtaa työtehtävää noin 4 – 7 kertaa päivässä. Työtehtävän vaihtamisen vuoksi käyttöliittymäsovelluksen vaatimuksia olivat hyvä käytettävyys, yksinkertaisuus ja nopea käyttö.

Sovellus toteutetaan mobiililaitteille yhteensopivaksi ja siihen käytetään CSS-ohjelmointikieltä, jotta sovellusta on helppo käyttää pienemmältäkin näytöltä. Mobiililaitteiden kirjautuminen aiotaan toteuttaa mobiililaitteiden yksilöivällä tunnuksella, jolle pitää löytyä tietokannasta vastaavuus. Mobiililaitteiden kirjautuminen jätetään viimeiseksi toteutettavaksi ominaisuudeksi.

Sovellukselle tehtiin palvelinpuolen sovellus, mikä tehdään yrityksen arkkitehtuurin mallin mukaisesti. Palvelinpuolen sovellus hoitaa tietokantatoimenpiteet, kuten esimerkiksi tiedon hakemisen ja lisäyksen. Palvelinpuolelle toteutetaan rajapinnat, joita käyttöliittymäsovelluksesta kutsutaan.

4.3 Sovelluksen yksilöidyt käyttötapaukset

Seuraavissa luvuissa on kuvattu sovelluksen käyttötapaukset.

4.3.1 Työntekijä kirjautuu sisään

Työntekijä tulee töihin ja kirjautuu töihin. Työntekijän täytyy syöttää työntekijännumero tai -nimi sekä valita toiminto, vuoro, asiakas, tuote ja linja pudotusvalikoista.

4.3.2 Käyttäjä vaihtaa kirjautumista

Työntekijän pitää pystyä vaihtamaan toimintoa, vuoroa, asiakasta, tuotetta tai linjaa työtehtävän vaihtuessa kirjautumulla sisään, ulos ja takaisin sisään.

4.3.3 Kirjautuminen sisään edellisten kirjautumisien perusteella

Edellisten kirjautumisien perusteella käyttäjä voi tehdä nopean kirjautumisen esimerkiksi edellisen työpäivän työvuorosta.

4.3.4 Käyttäjä kirjautuu ulos

Kun käyttäjä lopettaa työvuoronsa tai pitää ruokatauon, hänen pitää pystyä kirjautumaan ulos.

4.3.5 Käyttäjä katselee omia työaikatilastoja

Käyttäjä haluaa tietää, paljonko hänellä on ylivuorotunteja ja lomia pidettävänä, sekä katsoa edellisen kuukauden työtunteja raporttina.

4.4 Toteutus

Työtä tehtiin omalla ajalla ja yrityksen tiloissa töiden jälkeen keväällä 2014. Yrityksen työntekijöiltä saatiin sovelluksen toteutukseen hyviä vinkkejä ja apuja ongelmatilanteissa.
Työ aloitettiin keräämällä käyttäjäkokemuksia, etsimällä puutteita ja hyviä puolia vanhasta sovelluksesta, tutustumalla tietokantaan, tekemällä käyttötapauskuviot, minkä jälkeen pystyi suunnittelemaan käyttöliittymäkuvat. Tämän jälkeen aloitettiin itse sovelluksen teko. Sovellukselle tehtiin ensimmäiseksi rajapinnat palvelinpuolelle yrityksen arkkitehtuurin mukaisesti. Kaikki rajapinnan metodit testattiin väärällä ja oikealla tiedolla, jotta palvelinpuolesta saadaan mahdollisimman toimiva. Tämän jälkeen aloitettiin leimaussovelluksen tekeminen, jota tehtiin käyttötapaus kerrallaan. Kun yksi käyttötapaus oli valmis, se voitiin laittaa testaukseen ja aloittaa seuraava käyttötapaus. Näin saadaan varmasti kaikki tehtyä loogisesti ja virheitä tulee vähemmän.
Opinnäytetyön kirjallisen osuuden kirjoittaminen aloitettiin keväällä 2014 heti aloituspalaverin jälkeen. Sovellukselle jäi vielä hyviä jatkokehitysideoita.

5 TIETOKANTA

5.1 Tietokannan suunnitteleminen

Työssä käytettiin vanhan Kellokas-sovelluksen tietokantaa, johon lisätiin uusia kenttiä uusia ominaisuuksia varten. Yrityksen tietokanta on salaista tietoa, joten sitä ei lisätä opinnäytetyöhön.

5.2 Tietokannan toteutus

Koska sovellus tehtiin Microsoftin .NET-kehyksellä, käytettiin SQL Serveriä. Koska tietokanta oli olemassa, ei sitä kannata kokonaan hylätä, vaan jatkokehittää sitä ja miettiä, miten vanhaa taulurakennetta hajotetaan ja parannetaan yrityksen nykyisten tietokantojen taulurakenteiden näköiseksi. Palvelinpuolen sovellukselle tehtiin tietokannasta Entity Framework -malli, josta haettiin, tallennettiin ja päivitettiin tietoa Linq-kyselykielen avulla. Linq-kieli estää käyttöliittymä puolen SQL-injektiot, joten tietokantaan on vaikea tehdä sitä kautta hyökkäyksiä. Entity Frameworkista generoitiin POCO-luokkia, joita voidaan palauttaa käyttöliittymäpuolelle tieto tarpeen mukaan.

Alla olevassa kuviossa on palvelinpuolen ohjelmalta (KUVIO 13) 10 viimeisimmän kirjautumisen haku eri tauluista liitoskyselyn avulla. Kyselyssä tehdään liitoskysely moneen eri tauluun ja palautetaan valikoitua tietoa käyttöliittymälle.

```
public List<CustomRegistration> GetTop10(double empId)
{
    var query = (from k in context.Registrations
                 join l in context.Lines on new { LinjaId = k.LineId.Value } equals new { LinjaId = l.LineId } into l_join
                 from l in l_join.DefaultIfEmpty()
                 join v in context.Shifts on new { VuoroId = k.ShiftId.Value } equals new { VuoroId = v.ShiftId } into v_join
                 from v in v_join.DefaultIfEmpty()
                 join t in context.Products on new { TuoteId = k.ProductId.Value } equals new { TuoteId = t.ProductId } into t_join
                 from t in t_join.DefaultIfEmpty()
                 join kp in context.CostCentres on new { KpaikkaId = k.CostCentreId.Value } equals new { KpaikkaId = kp.CostCentreId } into kp_join
                 from kp in kp_join.DefaultIfEmpty()
                 join tm in context.Functions on new { ToimintoId = k.FunctionId.Value } equals new { ToimintoId = tm.FunctionId } into tm_join
                 from tm in tm_join.DefaultIfEmpty()
                 where
                   k.EmployeeId == empId
                 select new CustomRegistration
                 {
                     FunctionDesc = tm.Description,
                     FunctionId = tm.FunctionId,
                     LineDesc = l.Description,
                     LineId = l.LineId,
                     ShiftDesc = v.Description,
                     ShiftId = v.ShiftId,
                     ProductDesc = t.Description,
                     ProductId = t.ProductId,
                     CostCentreDesc = kp.Description,
                     CostCentreId = kp.CostCentreId,
                     Hour = k.Hour,
                     EndDate = k.EndDate,
                     RecordId = k.RecordId,
                     EmployeeId = k.EmployeeId,
                     FreeReference = k.FreeReference
                 });

    var result = query.Where(x => x.EndDate != default(DateTime)).ToList();

    result = result.OrderByDescending(x => x.RecordId).Take(10).ToList();

    return result;
}
```

KUVIO 13 Tietojen haku ja palautus.

Myös POCO-luokat, jotka generoidaan palvelinpuolelle, pitää samalla tavoin tehdä myös käyttöliittymäpuolelle, koska käyttöliittymä pitää saada toimimaan ilman palvelinpuolen luokkia, mikä helpottaa käyttöliittymän yksikkötestausta. Yleensä yksi luokka määrittelee yhden taulun, mutta tarpeen mukaan luokkaan ei tarvitse tehdä kaikkia kenttiä vaan siihen tehdään ne kentät, joita palvelinpuolen sovelluksella valitaan ja palautetaan. Yksi luokka voi sisältää monesta eri taulusta eri kenttiä.

Alla olevassa kuviossa (KUVIO 14) on käyttöliittymän luokka, johon palautetaan palvelinpuolen tietojoukko. Luokka sialtää yksittäisiä tietueita sekä listoja, joissa voi olla paljon tietoa esimerkiksi työntekijöistä ja kirjautumisista.

```
public class RegisterInputs : RequiredRegister
{
    public string EmpName { get; set; }
    public List<Function> Functions { get; set; }
    public List<Line> Lines { get; set; }
    public List<Shift> Shifts { get; set; }
    public List<Employee> Employees { get; set; }
    public List<CostCentre> CostCentres { get; set; }
    public List<Title> Titles { get; set; }
    public List<Product> Products { get; set; }

    public string FreeReference { get; set; }
    public double? Hour { get; set; }

    public double OverWorkHour { get; set; }
    public bool IsLogged { get; set; }
}

public class RequiredRegister
{
    public double EmpId { get; set; }
    public double FunctionId { get; set; }
    public double ShiftId { get; set; }
    public double LineId { get; set; }
    public double ProductId { get; set; }
    public double CostCentreId { get; set; }
    public double RegistrationId { get; set; }
    public double OldRegistrationId { get; set; }
    public string ProductDesc { get; set; }
    public string LineDesc { get; set; }
    public string ShiftDesc { get; set; }
    public string CostCentreDesc { get; set; }
    public string FunctionDesc { get; set; }


}
```

KUVIO 14 Esimerkki kirjautumisen kahdesta luokasta.

Alla olevassa kuviossa (KUVIO 15) on esimerkki siitä, miten ohjelma tallentaa työntekijän kirjautumisen. Parametrina tulee käyttöliittymästä täytetty Registration-luokka, joka on generoitu luokka Entity Frameworkista. Tämä saadaan suoraan lisättyä palvelinpuolella kulkevaan sisältöön (context), joka sitten tallennetaan kantaan asti. Tuloksena palautetaan rivien tallennusmäärän mukaan tosi tai epätosi.

```
public bool SaveRegistration(Registration request)
{
    request.RecordId = SeuraavaId();

    context.Registrations.Add(request);

    int rows = this.context.SaveChanges();

    return rows > 0;
}
```

KUVIO 15 Tiedon tallentaminen tietokantaan

Alla olevassa kuviossa (KUVIO 16) kutsutaan käyttöliittymäsovelluksen puolelta palvelinpuolen sovellusta ja sen rajapinta tallenna metodia. Kuviossa (KUVIO 17) näytetään, miten avataan palvelinpuolen yhteys, jota SaveRegistration-metodi aluksi kutsuu (EnsureOpen()). Yhteyden avaamisen jälkeen voidaan kutsua tallenna-metodia.

```
public bool SaveRegistration(Registration request)
{
    EnsureOpen();

    return Client.SaveRegistration(request);
}
```

KUVIO 16 Rajapinnasta toteuttava tallenna metodi

```
protected void EnsureOpen()
{
    ThrowIfDisposed();

    if (State == CommunicationState.Created)
    {
        try
        {
            client.Open();
        }
        catch (InvalidOperationException)
        {
            client.Abort();
            throw;
        }
    }
}
```

KUVIO 17 Yhteyden avaus ja sulkeminen poikkeuksen sattuessa

6 LEIMAUSSOVELLUS

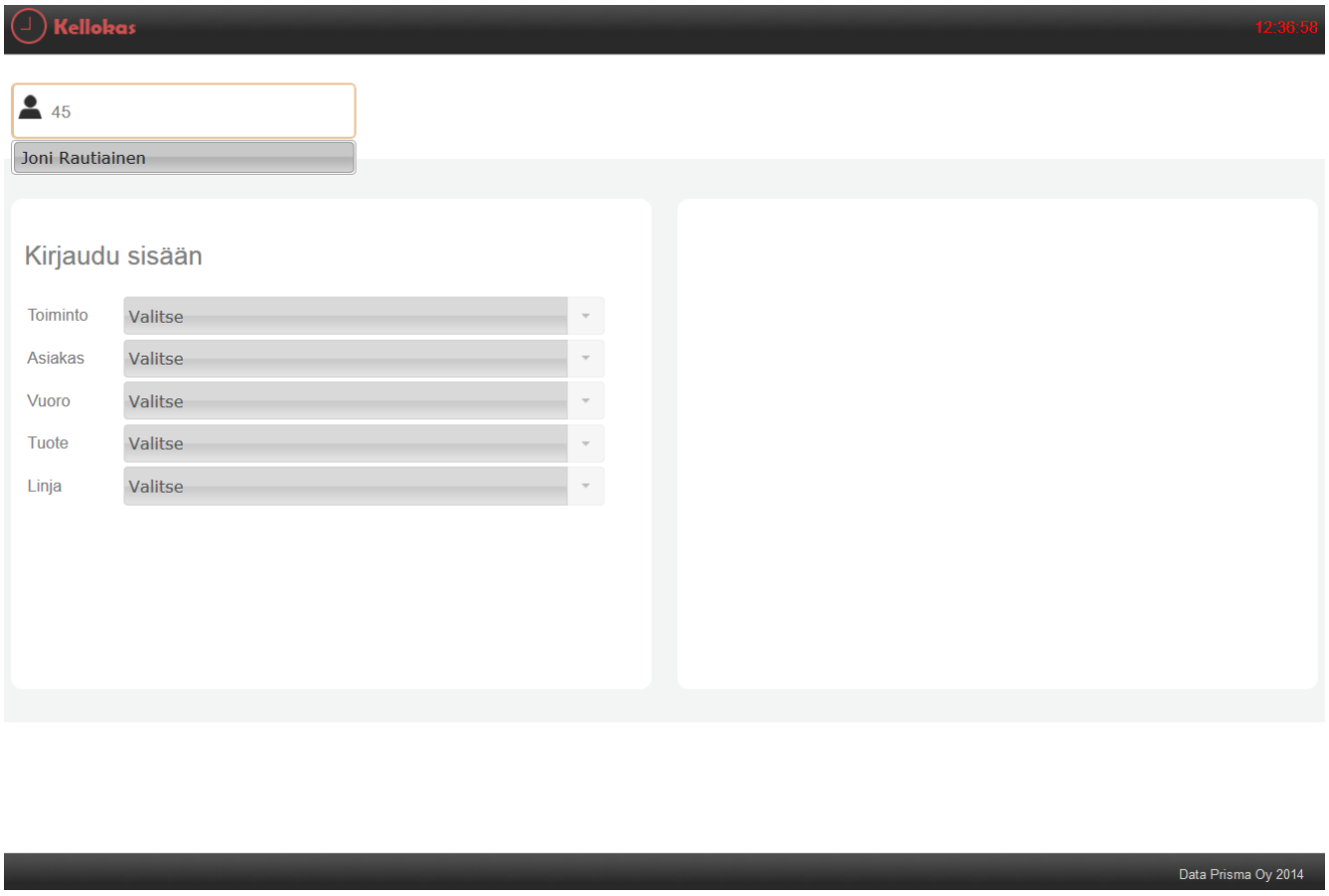
Sovellus sisältää yhden näytön, mutta tässä tapauksessa voidaan puhua laajuudeltaan ennemmin toiminnoista kuin näytöistä.

6.1 Sisäänkirjautuminen

Sovelluksen kirjautuminen hoidetaan paikallisessa verkossa pelkästään ennalta määritetyllä käyttäjänumerolla tai nimellä. Paikallisessa verkossa ei käytetä salasanaa, koska sovellus ei ole silloin julkisesti muiden saatavilla.

Kun sovelluksen avaa, sivulle avautuu alla olevan kuvion mukainen näyttö (KUVIO 18), jossa on kirjautuminen tekstikenttä ja kirjautuminen pudotusvalikoita, mitä ei voida valita ennen kuin on kirjauduttu sisään.

Kun ollaan ulkoverkossa, käyttäjän tekstikentän viereen tulee salasanakenttä, jotta ohjelmaa ei voi käyttää kuka tahansa.

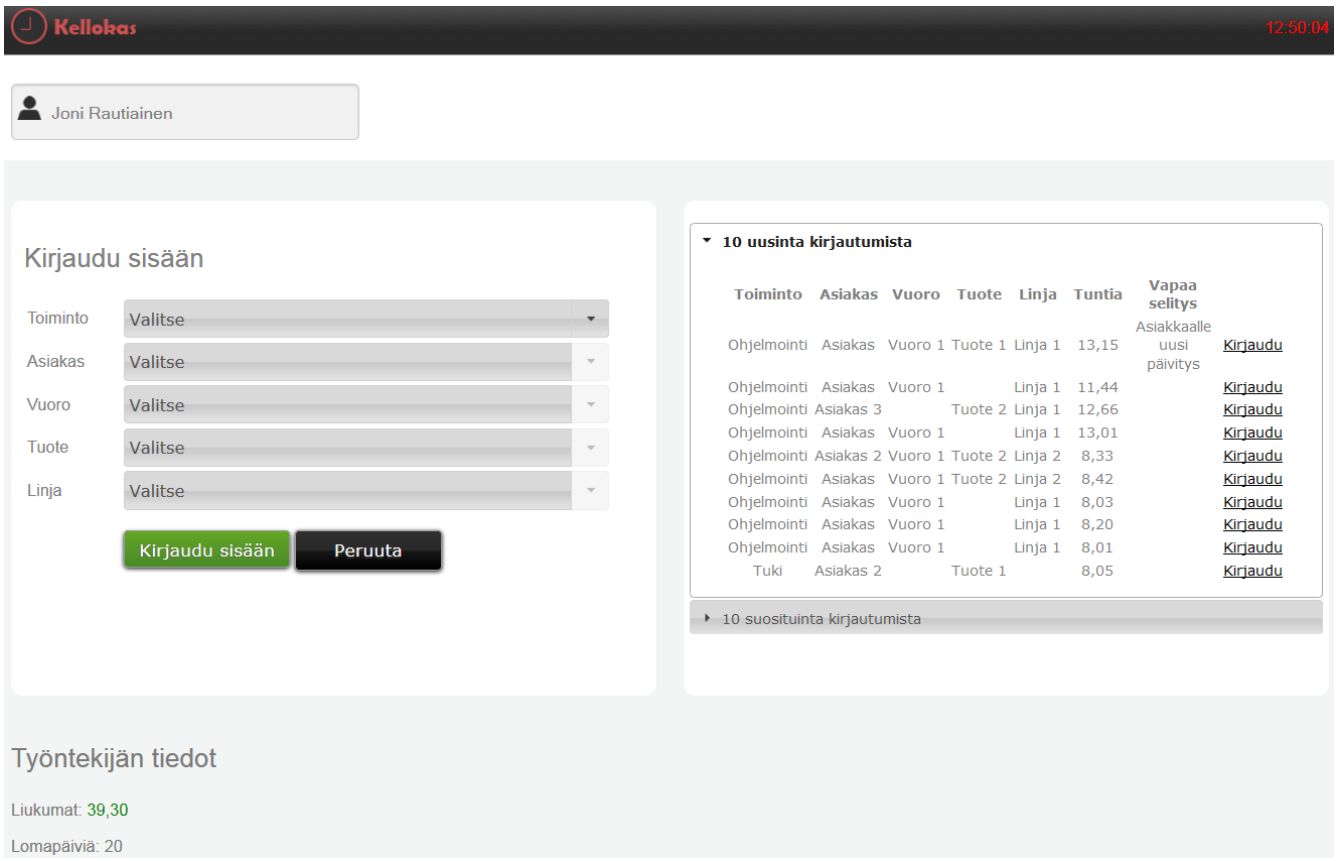


KUVIO 18 Kirjautuminen ohjelmaan. Tekstikenttään syötetään työntekijännumero tai nimi.

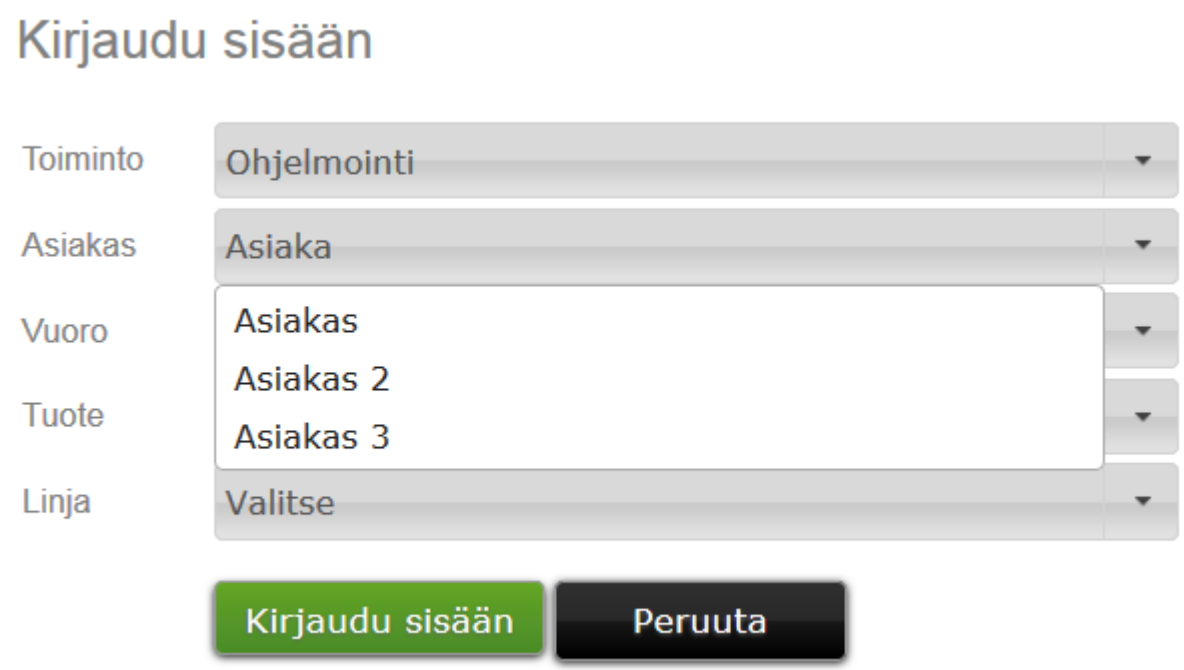
6.2 Käyttöliittymä

Kun käyttäjä on kirjautunut sovellukseen sisään, aukeaa mahdollisuus valita seuraava työtehtävä. Kirjautumisen jälkeen työntekijä näkee 10 viimeisintä kirjautumista, 10 suosituinta kirjautumista ja omat tietonsa.

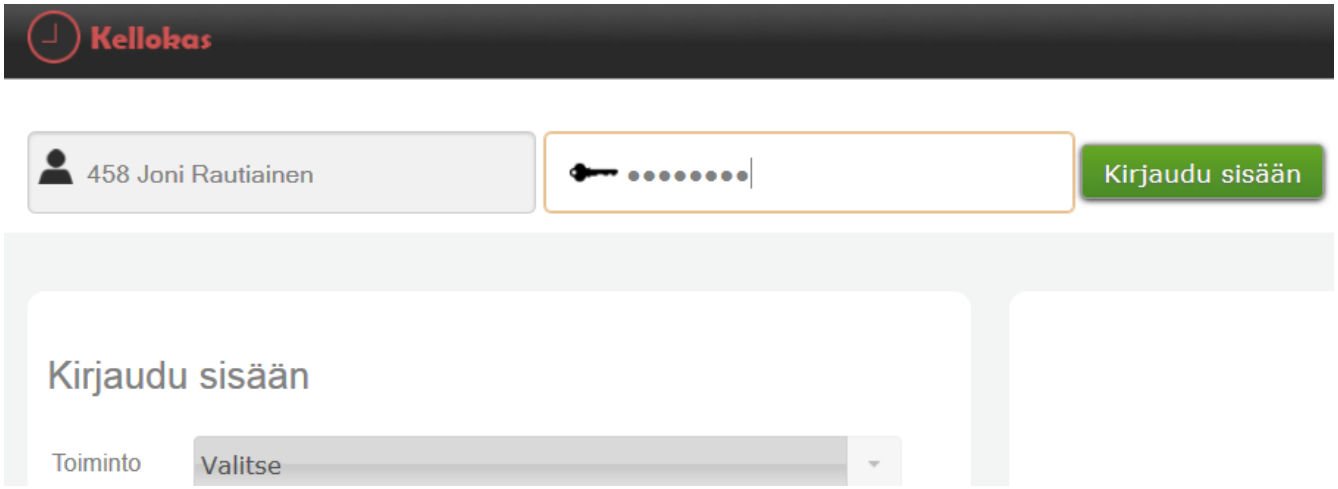
Alla olevassa kuviossa (KUVIO 19) on esitelty töihin kirjautuminen. Käyttäjä on itse valinnut pudotusvalikoista toiminnon, asiakkaan, vuoron, tuotteen ja linjan mahdollisista valinnoista. Pudotusvalikoiden sisällöt täytetään asiakaslähtöisesti, jotta käyttäjät eivät voi valita pudotusvalikoista mitä tahansa vaan ainoastaan esimerkiksi mitä tuotteita tai vuoroja asiakkaalle kuuluu. Jos työntekijä haluaa kirjautua edellisen päivän kirjautumisen mukaan, voi hän valita kohdista 10 uusinta tai suosituinta kirjautumista painamalla kirjaudu painiketta, jolloin pudotusvalikot täyttyvät edellisen kirjautumisen mukaan ja käyttäjä voi hyväksyä kirjautumisen painamalla kirjaudu sisään -painiketta. Käyttäjä näkee kirjautumisen aikana lomapäivänsä ja liukumatuntinsa.



KUVIO 19 Kirjautuminen töihin.



KUVIO 20 Pudotusvalikoissa on nimen ehdotus syötteen perusteella.

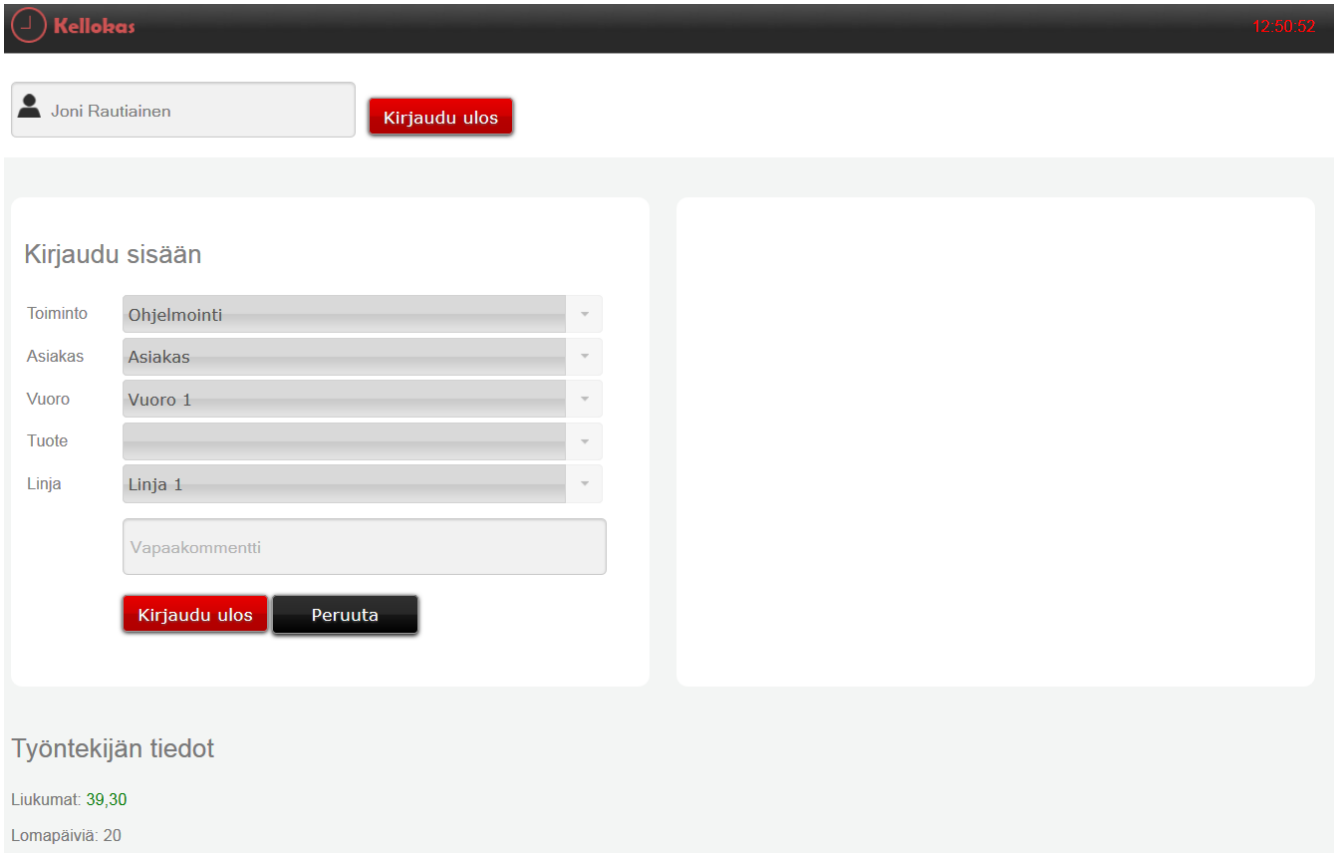


KUVIO 21 Kirjautuminen ohjelmaan ulkoverkossa.

6.3 Uloskirjautuminen

Uloskirjatuminen työvuorosta tapahtuu käyttäjän kirjautumisella sovellukseen sisään, minkä jälkeen hän näkee kirjaudu ulos-painikkeet punaisella käyttäjän tekstikentän vieressä sekä pudotusvalikoiden alapuolella. Käyttäjän pitää syöttää vapaaseen tekstikenttään kuvaus tehdystä työstä.

Alla olevassa kuviossa (KUVIO 22) on käyttäjän uloskirjautuminen työvuorosta.



KUVIO 22 Uloskirjautuminen.

6.4 Kirjautumisen vaihtaminen

Kirjautumisen vaihtaminen toimii samalla tavoin kuin kirjautuminen sisään ja ulos. Käyttäjä kirjautuu ensin olemassa olevasta projektista ulos, minkä jälkeen hän kirjautuu seuraavaan työtehtävään sisään.

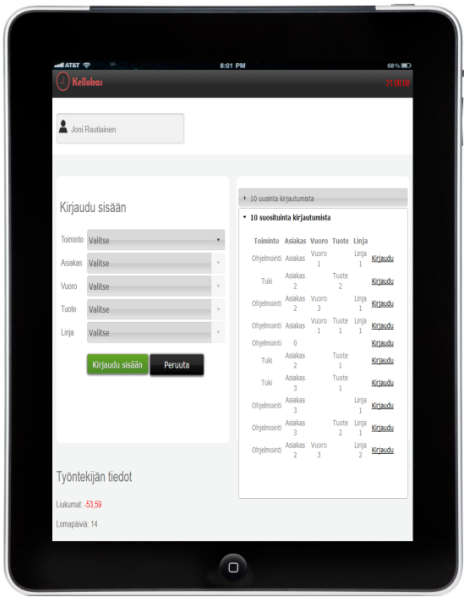
6.5 Käytettävyys

Sivun käytettävyydessä on otettu huomioon sen yksinkertaisuus ja tehty väreillä eri toimintoja, minkä vuoksi käyttäjän huomion herättäminen ja nopeampi käyttäminen sujuu hyvin. Sivun yksinkertaisuudesssa on huomioitu aikaisempia käyttäjäkokemuksia.

6.6 Mobiili

Nykyäivänä web-sivuja pitää pystyä käyttämään mobiililaitteilla ja pienemmillä näytöillä. Sivu skaalautuu alla olevan kuvioiden (KUVIO 23) ja (KUVIO 24) mukaisesti myös mobiililaitteille säilyttäen sen käytettävyyden.

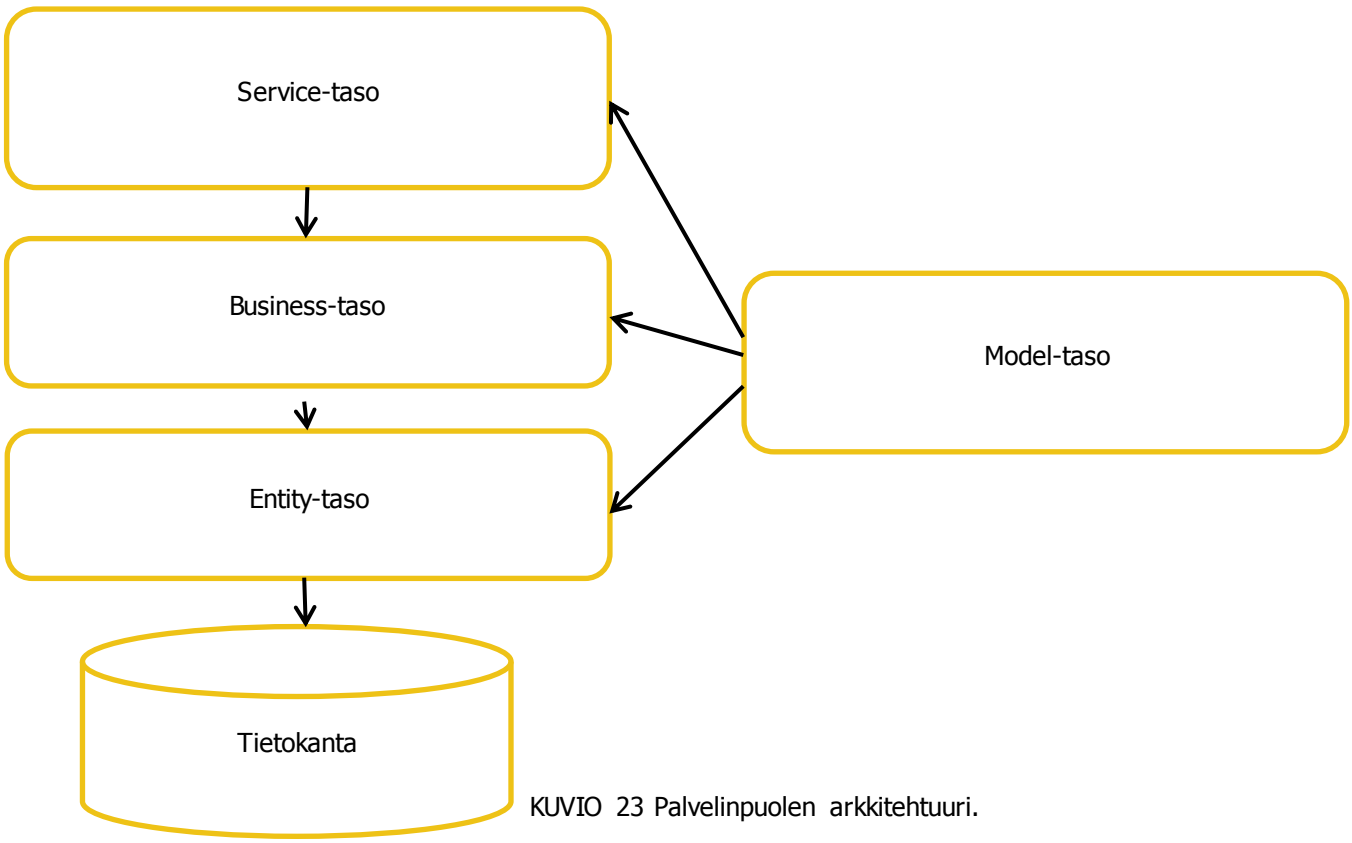
KUVIO 23 4-tuuman näyttö.



KUVIO 24 9,7-tuuman näyttö.

6.7 Arkkitehtuuri

Sovellukselle toteutettiin kaksi eri ohjelmaa, käyttöliittymä ja palvelinpuolen ohjelma. Tässä luvussa kuvataan palvelinpuolen arkkitehtuurirakenne kuviona (Kuvio 25), koska arkkitehtuuri on yrityksen salaista tietoa, jota ei julkaista opinnäytetyössä. Jokainen taso perii jonkin rajapinnan ja toteuttaa sen metodit.



KUVIO 23 Palvelinpuolen arkkitehtuuri.

7 TESTAUS

Projektia varten ei laadittu erillistä testaussuunnitelmaa vaan yhden käyttötapauksen valmistuttua sitä testattiin tekijän toimesta. Aikataulun kiirellisyyden vuoksi käyttöliittymän metodeja ei ehditty testa erillisillä yksikkötestauksilla, mutta tärkeämpänä pidettiin palvelinpuolen sovelluksen metodien yksikkötestausta. Joissain tapauksissa joidenkin ominaisuuksien lisääminen saattoi aiheuttaa vian toiseen koodilohkoon. Tämän välttämiseksi olisi pitänyt tehdä erillinen tehtävälista, joka olisi huolehtinut jokaisen erillisen toiminnon testaamisen koodin muuttuessa.

Virheiden etsimiseen käytettiin debugger-työkalua, joka on tietokoneohjelma, jota käytetään ohjelmointivirheiden jäljittämiseen ja korjaamiseen.

Koko sovelluksen testaus jää opinnäytetyön jälkeiseksi projektiksi, joka suoritetaan yhdessä ammattitestaajien kanssa. Ammattitestaajat merkitsevät testauspöytäkirjoihin kaikki virheet.

Sovelluksesta on tarkoitus tehdä kaupallinen sovellus, koska kyseinen ohjelma on yrityksen tuotevalikoimassa. Jatkokehityksenä voidaan miettiä lisää ominaisuuksia käyttäjäkokemuksien perusteella tulevia asioita sekä asiakkaiden toivomuksien mukaan saadaan sovellukseen lisää toiminnallisuutta.

8 TYÖN ARVIOINTI

Työn aihe oli kiinnostava ja sitä oli mukava tehdä. Työn aloittaminen ei ollut ollenkaan vaikeaa aikaisen sovelluksen käytön, tutun tekniikan käytön ja käyttökokemuksien takia. Projektin suunnitteluun käytettiin alussa jonkin verran aikaa, jotta ohjelmasta saadaan varmasti oikeanlainen ja sen idea sopii myös kaupalliseen käyttöön. Suunnittelun pohjatyö helpotti myös työn nopeaa valmistumista. Koulussa MVC-tekniikkaa ei ollut käsitelty aikaisemmin, mutta aikaisempi työkokemus auttoi tekemään sovellusta kyseisellä tekniikalla. Kehitysympäristö ja tietokantatyökalut ovat tulleet tutuksi koulu- sekä työympäristössä.

Aikaisempien käyttökokemuksien pohjalta saatiin karsittua vanhoja häiritseviä tekijöitä sekä lisättyä sovellukseen muutama kauan toivottu ominaisuus. Näitä ominaisuuksia olivat muun muassa lomapäivät, liukumat, pudotusvalikoista hakeminen tekstillä ja edellisen tehtävän valitseminen uudelleen.

Projektin kiinnostavuus vastasi hyvin odotuksiani. Tiesin tekiessäni, että sovellusta on mukava tehdä ja odotin sen haasteita. Sovelluksen teko ja opinnäytetyön kirjoittaminen sujui hyvin varatun ajan ja kotona tehdyn työn takia.

Olen tyytyväinen projektin lopputulokseen ja tuotteeseen. Sovelluksesta tuli selkeä ja yksinkertainen käyttää. Projektin aikana opin paljon syvällisemmin MVC-tekniikasta.

9 POHDINTA

Työn tekeminen oli haastavaa, koska opinnäytetyön ohelle joutui tekemään töissä muita töitä kuin opinnäytetyötä eikä aina ollut aikaa keskittyä työn tekemiseen. Tämän seurauksena opinnäytetyön tekeminen venyi kotona iltatyöksi.

Olen tyytyväinen, että aloitin opinnäytetyön kirjoittamisen ajoissa ja aloin hankkimaan lähdetietoja kirjoista ja internetistä.
Ohjelmistokehityksessä haluaisin vielä oppia koodin monikäyttöisyyden, mutta uskon tämän tulevan vuosien varrella. Omalle koodille ei kannata vielä tässä vaiheessa olla liian tiukka.
Projekti oli ajoittain henkisesti raskas, mutta onneksi kovalla työmäärällä kaikki onnistui hyvin.

LÄHTEET JA TUOTETUT AINEISTOT

Codeplex. What is EF? [Viitattu 2014-04-15] Saatavissa: http://entityframework.codeplex.com/

Microsoft 2014. LINQ (Language-Integrated Query). [Viitattu 2014-04-01] Saatavissa: http://msdn.microsoft.com/en-us/library/bb397926.aspx

MSDN Microsoft 2014. Visual Studio 2013. [Viitattu 2014-04-09] Saatavissa: http://msdn.microsoft.com/en-us/vstudio/cc136611.aspx

MSDN Microsoft 2014. What is Entity Framework? [Viitattu 2014-04-06] Saatavissa: http://msdn.microsoft.com/en-us/data/ef.aspx

Ohjelmointiputka 2007. JavaScript-perusopas. [Viitattu 2014-03-25] Saatavissa: http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=js_01

Technet Microsoft. Use SQL Server Management Studio. [Viitattu 2014-03-03] Saatavissa: http://technet.microsoft.com/en-us/library/ms174173.aspx

TechTarget. Internet Information Services.[Viitattu 2014-04-30] Saatavissa: http://searchwindowsserver.techtarget.com/definition/IIS

William Penberthy, Microsoft Press, O'Reilly Media, 2013. Exam Ref 70-486: Developing ASP.NET MVC 4 Web Applications.

W3-Schools. JQuery.[Viitattu 2014-05-04] Saatavissa: http://www.w3schools.com/jquery/jquery_intro.asp