Työajanhallintajärjestelmä

Ammattikorkeakoulututkinnon opinnäytetyö

Tietotekniikan koulutusohjelma

Riihimäki, kevät 2017

Teemu Heinonen

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

RIIHIMÄKI
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

Tekijä Teemu Heinonen **Vuosi** 2017

Työn nimi Työajanhallintajärjestelmä

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli luoda asiakkaalle selainpohjainen työajanhallintajärjestelmä, joka toimisi niin tietokoneessa kuin mobiililaitteessa. Tämän lisäksi sovelluksella tulisi voida ladata CSV-raportteja sekä siirtää työntekijän tunnit suoraan laskutusohjelmaan annettuja rajapintoja hyödyntäen.

Työajanhallintajärjestelmä luotiin ASP.NET-ympäristöön Umbraco CMS-järjestelmän päälle MVC-mallin mukaisesti. Liitännäisinä hyödynnettiin Googlen Material Designiin perustuvaa sovelluskehystä, Materializea sekä JavaScript-kirjastoa, jQueryä. Sovelluksen palvelimeksi päädyimme asiakkaan omaan Windows-pohjaiseen webhotelliin.

Opinnäytetyö käsittelee työajanhallintajärjestelmän toimintoja, vaatimuksia, ympäristöä sekä sovelluskehystä. Lisäksi työssä käsitellään projektin eri työvaiheita. Opinnäytetyön kohdeyleisö on tietotekniikan koulutusalan opiskelijat.

Valmis järjestelmä oli onnistunut ja mieluinen asiakkaalle. Kaikki asiakkaan määrittämät vaatimukset saatiin toteutettua aikamääreiden puitteissa.

Avainsanat Työajanhallinta, ASP.NET, Umbraco, SQL CE, MVC

Sivut 27 sivua, joista liitteitä 8 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

RIIHIMÄKI
Degree Programme in Information Technology
Software technology

Author Teemu Heinonen **Year** 2017

Subject Time Tracking Solution

ABSTRACT

The objective of the thesis was to create a web-based time tracking solution which would work on a computer as well as on a mobile device. In addition, the application would provide the user an option to download CSV reports and to move employees' working hours directly on to the billing software using the provided interfaces.

The time tracking solution was created into the ASP.NET environment on top of Umbraco CMS using the MVC model. Materialize, a modern responsive framework based on Google's Material Design, together with jQuery, a JavaScript library, were utilized in the front-end development. The customer's own Windows-based webhosting was decided to be used as the server for the application.

This thesis covers the operations, requirements, environment and application framework of the time tracking solution. Furthermore, the different phases of this project are thoroughly described. The target audience of this thesis are other information technology students.

The finished product was successful and the customer was pleased with the outcome. All of the requirements made by the customer were met within the given timeline.

Keywords Time Tracking, ASP.NET, Umbraco, SQL CE, MVC

Pages 27 pages including appendices 8 pages

TERMIT JA LYHENTEET

Suomi	**English**	**Selitys**
CSS	CSS	WWW-dokumenteille suunniteltu tyylioh-jeiden laji.
CSV	Comma-separated values	Tiedostomuoto, jolla tallennetaan taulukkomuotoista tietoa.
Enkoodaus	Encoding	Kooderi, joka tiivistää datan.
FTP	File Transfer Protocol	TCP-protokollaa hyödyntävä tiedostonsiirtomenetelmä.
JavaScript	JavaScript	Dynaaminen komentosarjakieli.
jQuery	jQuery	Avoimeen lähdekoodiin perustuva JavaScript-kirjasto.
Käsittelijä	Controller	Käsittelee käyttäjältä tulleet käskyt ja muuttaa mallia ja näkymää vastaamaan käskyä.
LESS	LESS	CSS-kieli.
Malli	Model	Suorittaa järjestelmän tiedon tallentamisen, käsittelyn ja ylläpidon.
Materialize	Materialize	Avoimeen lähdekoodiin perustuva sovelluskehys.
MVC	MVC	Ohjelmistoarkkitehtuurityyli.
Näkymä	View	Määrittää sovelluksen ulkoasun ja datan esityksen sovelluksessa.
Rajapinta	Application Programming Interface	Määritelmä, jonka mukaan ohjelmat tekevät pyyntöjä ja vaihtavat tietoja keskenään.
Responsiivinen	Responsive	Skaalautuva, skaalaa web-sovelluksen käyttäjän laitteen mukaan.
SSL	Secure Sockets Layer	Salausprotokolla.
SQL CE	SQL Server Compact	Tietokantahallintajärjestelmä.
Umbraco	Umbraco	Sisällönhallintajärjestelmä.

XML	Extensible Markup Language	Standardi, jolla tiedon merkitys on kuvattavissa tiedon seassa.
.NET	.NET	Ohjelmistokomponenttikirjasto.

SISÄLLYS

Liitteet

1 JOHDANTO

Opinnäytetyön tavoitteena on luoda asiakkaalle responsiivinen web-sovellus, jonka avulla hän voi seurata omia sekä työntekijöidensä työpäivien pituuksia projektikohtaisesti. Responsiivisuudella tarkoitetaan sitä, että sovellus mukautuu automaattisesti käyttäjän laitteen mukaan. Sovelluksessa tulee myös olla mahdollisuus merkitä matka- ja kulukorvaukset vaivattomasti järjestelmään sekä siirtää tehdyt tunnit laskutusohjelmaan hyödyntäen annettuja rajapintoja.

Sovellus toteutetaan Umbraco-sisällönhallintajärjestelmän päälle, jotta asiakas pystyisi muokkaamaan sisältöä sekä lisäämään käyttäjiä mahdollisimman helposti. Koska käytössä on .NET-pohjainen sisällönhallintajärjestelmä, tulee sovelluksen pyöriä Windows-pohjaisessa palvelimessa sekä ASP.NET web -ohjelmistokehyksessä. Ohjelmistoarkkitehtuurityylinä käytän MVC-arkkitehtuuria, jota hyödynnetään paljon varsinkin web-sovellusohjelmoinnissa.

Pääsääntöisesti sovellus ohjelmoidaan käyttäen C#-kieltä, mutta mukana on myös muita ohjelmistokieliä kuten LESS, JavaScript sekä jQuery. Tietokantahallintajärjestelmäksi päädyin hyödyntämään Microsoftin SQL CE-kantaa, koska sovelluksen kanssa ei ole tarvetta tehokkaammalle tietokantahallintajärjestelmälle. Sovelluksessa käytän myös Materializea, joka on Googlen Material Designiin perustuva sovelluskehys.

Opinnäytetyötä tehdessäni minulla on tarkoitus oppia enemmän .NET-sovelluskehitysteknologioista sekä rajapintojen hyödyntämisestä. Tämän lisäksi saan kokemusta selainpohjaisten ratkaisujen toteuttamisesta sekä oman projektin vetämisestä. Asiakkaalle toteutetaan valmis sovellus, johon hän saa täydet oikeudet.

2 TEKNOLOGIAT

Opinnäytetyö toteutetaan Visual Studiolla, joka pystyy pyörittämään ASP.NET-pohjaista web-sovellusta paikallisessa ympäristössä hyödyntäen IIS Expressiä. Lopullinen sovellus asennetaan asiakkaan omaan Windows-pohjaiseen webhotelliin.

Muita teknologioita opinnäytetyössäni on Umbracon sisällönhallintajärjestelmä, joka hyödyntää tässä työssä Microsoftin SQL CE -kantaa. Sovellus toteutetaan käyttämällä ASP.NET MVC -web sovelluskehystä, sekä Googlen Material Designiin perustuvaa Materializea.

2.1 **Microsoft Visual Studio**

ASP.NET-sovelluksia tehdessäni suosituin ohjelmankehitysympäristö on Microsoftin Visual Studio, jolla voi ohjelmoida usealla eri ohjelmointikielellä kuten C#, C++ ja Visual Basicillä. Visual Studiolla tehdään paljon Windows-, web- sekä mobiilisovelluksia hyödyntäen Visual Studiosta löytyviä laajennuksia. Mobiilisovelluksista löytyy esimerkiksi Xamarin laajennus, jolla voidaan luoda mobiilisovellus käyttäen pelkästään C#-kieltä. (Microsoft, n.d.)

2.1.1 NuGet

NuGet on Microsoftin luoma pakettienhallintaohjelma Visual Studioon. NuGetin avulla ohjelmistokehittäjä voi käyttää ja luoda ohjelmapaketteja esimerkiksi omiin projekteihinsa. NuGetin hyvinä puolina voidaan pitää pakettien helppoa päivittämistä. Mikäli vastaan tulee tilanne, että ohjelmistokehittäjällä on vanha versio paketista, hän voi päivittää sen automaattisesti NuGetin avulla. (Microsoft, n.d.)

Itse hyödynnän NuGetia opinnäytetyössäni asentaessani Umbraco CMS-järjestelmää sovellukseeni. Mikäli en hyödyntäisi NuGetia projektissa, minun pitäisi ladata Umbraco manuaalisesti verkosta.

2.2 **Umbraco**

Umbraco on avoimeen lähdekoodiin perustuva sisällönhallintajärjestelmä, jota hyödynnetään .NET-pohjaisissa web-sovelluksissa. Se on kirjoitettu pääsääntöisesti C#-kielellä ja toimii Microsoft IIS-ympäristössä. (Umbraco, 2017.) Tulevaisuudessa Umbraco voi pyöriä jopa Linux-ympäristössä käyttämällä tarvittavia sovelluskehyksiä kuten Monoa. Tähän saattaa kuitenkin kulua vielä tovi, mutta uskon, että kun Umbraco on saatu yhteensopivaksi, voi johtavan CMS-järjestelmän, WordPressin paikka olla vaakalaudalla.

Päädyin Umbraco-sisällönhallintajärjestelmään opinnäytetyön kanssa, koska olen työni puolesta tehnyt kyseisellä järjestelmällä aikaisemminkin töitä. Umbracoa on ollut helppo oppia, koska dokumentaatiota löytyy verkosta paljon. Tämän lisäksi minun on mahdollista saada tarvittaessa apua työkavereiltani.

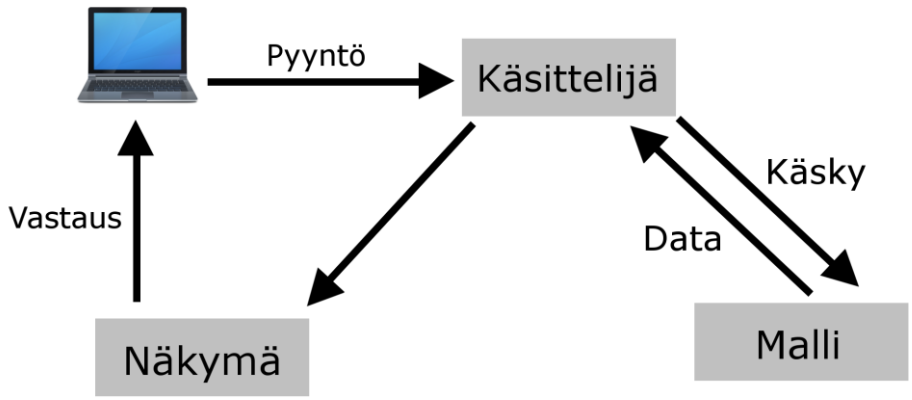
2.3 **Microsoft SQL CE**

Microsoft SQL Server Compact on Microsoftin kehittämä kompakti relaatiotietokanta sovelluksille, jotka pyörivät mobiililaitteilla sekä tietokoneilla. Toisin kuin muut Microsoftin SQL-serverit, SQL CE pyörii yhdessä applikaation kanssa ja yksittäinen tietokanta voi olla kooltaan jopa 4GB, mikä on enemmän kuin tarpeeksi opinnäytetyötä ajatellen. (Tiffany, 2010.)

SQL CE on ilmainen tietokanta, minkä vuoksi päädyin hyödyntämään tätä. Tämän lisäksi asiakkaan käyttömäärällä ei ole tarvetta tehokkaampaan tai monipuolisempaan tietokantatyyppiin.

2.4 **ASP.NET MVC**

ASP.NET MVC on Microsoftin kehittämä websovelluskehys, joka implementoi MVC-mallia. MVC on lyhenne sanoista malli, näkymä sekä käsittelijä. Malli, englanniksi model, kuvaa järjestelmän ylläpidon, käsittelyn sekä tallentamisen. Näkymän, englanniksi view, avulla määritellään sovelluksen ulkoasu sekä miten tieto näytetään käyttöliittymässä. Viimeisenä on käsittelijä, englanniksi controller, joka vastaanottaa datan ja jonka tehtävänä on muuttaa näkymä ja malli vastaamaan datan tulosta (Kuva 1). (Galloway, Wilson, Allen & Matson, 2014, 2.)

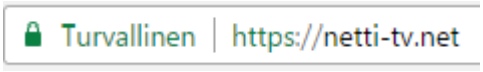


Kuva 1. MVC-mallin toiminta havainnollistettuna (Teemu Heinonen, 2017)

MVC:n etuina voidaan pitää sitä, että malli ei ole riippuvainen näkymästä eikä ohjaimesta ja niiden riippuvuus toisistaan on minimaalinen. Haittoina taas voidaan pitää suorituskykyongelmia, mikäli sovelluskehittäjä ei optimoi tietokantahakua käyttöliittymän tarpeiden näkökulmasta.

2.5 **SSL-varmenne**

SSL on lyhenne sanoista Secure Sockets Layer. Sitä hyödynnetään verkkoliikenteen suojaamisessa niin, että kahden tietokoneen välinen yhteys on turvallinen. SSL varmistaa sen, että lähtevä data, kuten esimerkiksi salasanat, pysyvät turvassa hakkereilta siirron aikana. (Aitoa, n.d.) SSL-varmenteen voi huomata sivuissa siitä, että osoitteen alussa lukee https (Kuva 2.).



Kuva 2. SSL suojattu sivu (Teemu Heinonen, 2017)

Koska työajanhallintajärjestelmä sisältää henkilötietoja sekä asiakastietoja, on SSL-varmenteen käyttöönotto todella tärkeätä. Tämän lisäksi Google on muuttanut hakutuloskäytäntöjään vuodelle 2017 niin, että sivut, joissa on SSL-varmenne näkyvät korkeammalla hakutuloksissa. (Hyer, 2016.) SSL-varmenne siis mahdollistaa asiakkaalle myös sen, että hänen yrityksensä sivut löytyvät helpommin Googlesta.

2.6 **Materialize**

Materialize on Googlen Material Designiin perustuva sovelluskehys. Google julkaisi Material Designin vuonna 2014 ja se perustuu ruudukkojärjestelmään, responsiivisiin animaatioihin sekä syvyysefekteihin kuten varjoihin. (Materialize, n.d.)

Materialize tarjoaa ohjelmistokehittäjälle valmiita CSS-tyylimäärittelyjä sekä JavaScript- ja jQuery-funktioita. Näistä CSS-tyylimäärityksinä on käytössä muun muassa ruudukkojärjestelmä, taulukkorakenne sekä varjostus. Funktiopuolelta löytyvät taas karuselli, navigaatio sekä ponnahdusikkuna.

3 **SOVELLUKSEN SUUNNITTELU**

Projektin lähtökohtana on luoda asiakkaalle sovellus, jonka avulla hän pystyy seuraamaan omia sekä työntekijöidensä työtunteja. Lisäksi sovelluksella tulee pystyä selvittämään suoraan, paljonko aikaa on mennyt minkäkin yrityksen työtehtäviin sekä oikea summa laskuttamista varten. Sovelluksella pitää pystyä myös siirtämään tiedot laskutusohjelmaan mahdollisimman vaivattomasti.

Asiakkaalla ei ole aikaisemmin ollut vastaavanlaista järjestelmää käytössään ja on usein nähnyt tarvetta omalle työajanhallintajärjestelmälle. Olemassa olevia työajanhallintajärjestelmiä löytyy kyllä monia erilaisia, mutta nämä on tarkoitettu lähes poikkeuksetta isommille yrityksille. Koska asiakkaalla on myös tarvetta siirtää tiedot omaan laskutusohjelmaan, jää ainoaksi vaihtoehdoksi kehittää oma järjestelmä.

3.1 **Vaatimusmäärittely**

Sovellusta lähdetään kehittämään asiakkaan kanssa kahdenkeskisissä palavereissa. Tällöin saan myös oppia, miltä tuntuu viedä omaa projektia eteenpäin. Palaverin pohjalta sovellukselle tuli seuraavia vaatimuksia:

- Työntekijän kirjautumissivu sekä omien tietojen muuttaminen
  - Työntekijöiden lisäämisestä sovimme, että tämä on helpointa toteuttaa Umbracon järjestelmässä.
  - Käyttäjiä on kahta eri tyyppiä: ylläpitäjä, jonka on mahdollista ottaa tarkempia raportteja ja siirtää tietoja laskutusohjelmaan, sekä normaali käyttäjä.
  - Käyttäjällä tulee olla mahdollisuus resetoida salasana, mikäli hän sen unohtaa.
  - Käyttäjällä tulee olla mahdollisuus kirjautua ulos järjestelmästä.
- Työtuntien merkitseminen
  - Tarvittavat kentät:
    - Työpäivä ja käytetty aika
    - Selite työlle
    - Mille projektille työtä on tehty
    - Laskutetaanko asiakasta
    - Onko työtunnin merkintä valmis
  - Mahdollisuus muokata sekä poistaa työtunteja.
  - Mahdollisuus nähdä edellisen kuukauden työtunnit.
  - Mikäli työntekijä on esimerkiksi sairauslomalla hän voi tallentaa pidemmän aikavälin työtunnit yhdellä sivulla.
- Matka- ja kulukorvausten merkitseminen
  - Matka- ja kulukorvaukset tulee olla jaoteltuina eri sivuille.
  - Kulukorvauksessa tulee olla alla olevat kentät:
    - Aloituspäivä ja -aika
    - Selite ja korvauksen määrä
    - Mikäli korvaus syntyy jollekin tietylle projektille, tulee tämä valita
    - Mahdollisuus lisätä liitetiedosto esimerkiksi kuittia varten
    - Laskutetaanko korvaus asiakkaalta
  - Matkakorvauksessa tulee olla samat kentät kuin kulukorvauksessa, mutta myös:
    - Lopetuspäivä ja -aika
    - Matkalla kertyneet kilometrit ja matkareitti
    - Matkustajien määrä ja nimet
    - Päivärahan tyyppi ja summa
  - Vanhojen korvausten muokkaus- ja poistotoiminto.
  - Mahdollisuus nähdä edellisen vuoden korvaukset.

- Raportit
    - Raportit voi ladata CSV-tiedostona.
    - Kuinka paljon työtunteja annetulla aikavälillä on tehty.
    - Työntekijällä on mahdollisuus nähdä vain omat raportit.
    - Ylläpitäjällä on mahdollisuus nähdä kaikkien työntekijöiden työtunnit sekä mahdollisuus ladata xml-tiedosto laskutusohjelmaa varten.
    - Työtuntien siirto laskutusohjelmaan hyödyntäen annettuja rajapintoja.
- Ulkoasu
    - Sovelluksen tulee olla responsiivinen.
    - Ulkoasun tulee olla samantyylinen kuin nykyisessä laskutusohjelmassa.
    - Helppokäyttöisyys.
- Sovelluksen tulee toimia sekä puhelimella että tietokoneella
    - Web-pohjainen sovellus, jolloin työtunteja pystyy lisäämään mahdollisimman monelta eri laitteelta.
    - Sovelluksen tulee toimia yleisimmillä selaimilla.

Sain asiakkaalta vapaat kädet sovelluksen toteuttamiseen, mikä on minulle todella tärkeää. Tämä mahdollisuus nopeuttaa työntekoa sekä antaa minulle mahdollisuuden tuoda omia näkemyksiäni esille.

3.2 **Aikataulu**

Aikataulullisesti sovittiin asiakkaan kanssa, että saisin tehdä sovellusta rauhassa. Tämä tarkoitti sitä, että sovelluksen tuli olla valmis viimeistään 4/2017, jotta minulle jäisi aikaa kirjoittaa opinnäytetyöraporttia. Koska aikataulua oli näin väljä, en koskaan tehnyt varsinaista aikataulua itse sovellusta koskien. Alla olevasta taulukosta (Taulukko 1) voi kuitenkin saada suuntaa antavan käsityksen, milloin tein mitäkin. Sovelluksen teko aloitettiin 10/2016, jolloin aikaa projektin tekemiseen minulla oli yhteensä 7 kuukautta.

Taulukko 1. Suuntaa antava aikataulu.

Tehtävä	Arviolta valmis	Toteutunut aikataulu
Projekti määrittely	10/2016	10/2016
Kirjautumissivu	10/2016	10/2016
Ulkoasunsuunnittelu	10/2016	10/2016
Työtunnit -sivut	12/2016	12/2016
Matka- ja kulukorvaus –sivut	12/2016	12/2016
Raportit -sivut	1/2017	1/2017
Tietojen vienti laskutus-ohjelmaan	2/2017	1/2017
Ulkoasun kehittäminen mainostoimiston kanssa	2/2017	2/2017
Korjaukset	3/2017	2/2017
Sovelluksen esittäminen ja asiakkaan hyväksyntä	3/2017	2/2017
Sovelluksen toimitus	3/2017	2/2017
Palaute	4/2017	3/2017
Mahdolliset korjaukset	4/2017	3/2017
Valmiin sovelluksen toimitus	4/2017	3/2017

Aikataulullisesti sovellus saatiin valmiiksi nopeammin kuin olin itse uskonut. Sovelluksessa oli kuitenkin suhteellisen laajat vaatimusmäärittelyt eikä kyseessä ollut mikään konseptisovellus vaan lopullinen tuotos. Syitä nopeampaan valmistumiseen on monia. Näistä yhtenä voidaan pitää sitä, että olin asettanut itselleni aikataulun suhteellisen väljäksi, myös tietojen vienti laskutusohjelmaan oli varsin helposti toteutettavissa.

Varsinaisia projektipalavereja sovelluksen teon aikana ei pidetty, koska olin saanut niin vapaat kädet toteuttamisen kanssa. Mikäli minulle kuitenkin heräsi kysymyksiä, pystyin olemaan asiakkaaseen yhteydessä joko sähköpostitse tai puhelimitse.

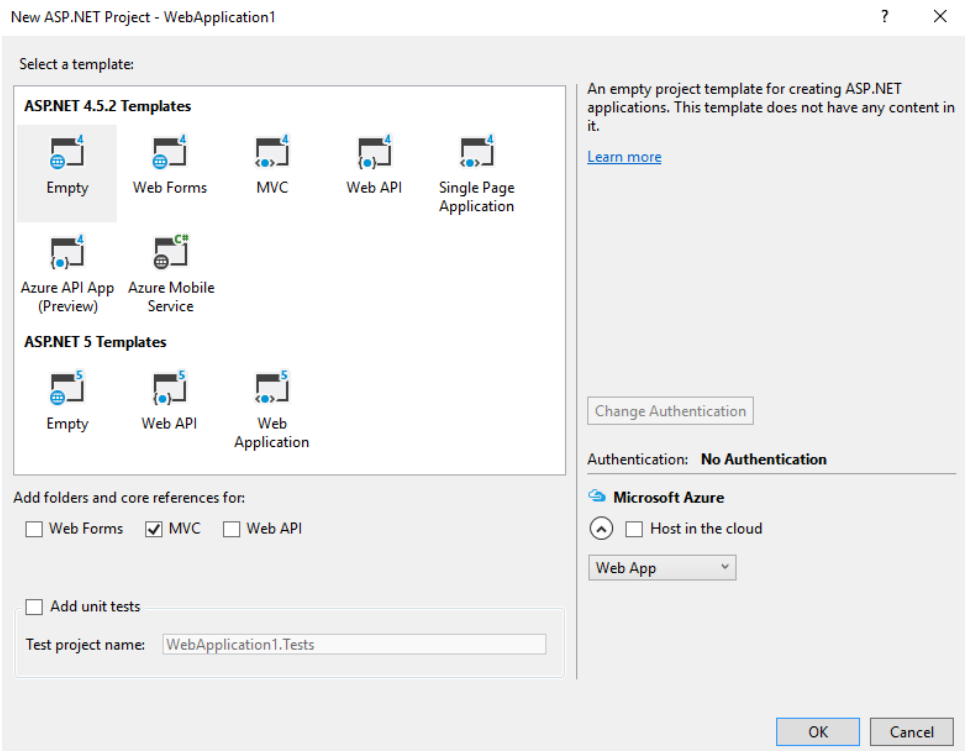
4 SOVELLUKSEN TOTEUTUS

4.1 Sovelluksen pohjan toteuttaminen

Sovelluksen suunnittelun sekä vaatimusmäärittelyjen jälkeen pääsin aloittamaan itse sovelluksen tekoa. Koska vaatimusmäärittely oli niin tarkasti kirjattu, itse sovellusta oli helppo lähteä toteuttamaan. Ensimmäisenä vaiheena oli luoda uusi projekti Visual Studiossa.

4.1.1 Uuden projektin luonti Visual Studiossa

Visual Studiota käytettäessä on hyvä käynnistää ohjelma aina järjestelmänvalvojana. Tämä vähentää huomattavasti päänvaivaa työkalujen toimimattomuuden kanssa, koska yleisin ongelma on oikeuksien puutteellisuus. Aloitin sovelluksen teon luomalla tyhjän ASP.NET 4.5 -web-sovelluksen MVC -kansioilla (Kuva 3).



Kuva 3. Tyhjän ASP.NET -web-sovelluksen luonti (Teemu Heinonen, 2017)

Visual Studio loi automaattisesti tarvittavat tiedostot sekä kansiot projektiani varten. MVC-laatikon rastittaminen loi automaattisesti controller-, model- sekä view-kansiot projektiini. Nämä olisi voinut myös itse lisätä manuaalisesti, mutta aikaa säästääkseni annoin Visual Studion luoda ne projektiini automaattisesti.

Koska kehitin sovellusta paikallisessa ympäristössä tuli minun pystyä myös pyörittämään sovellusta mahdollisimman vaivattomasti. Onneksi Visual Studion mukana tulee IIS Express, jonka avulla pystyin vaivattomasti pyörittämään sovellustani paikallisessa ympäristössä. Tämä mahdollisti myös sovelluksen virheiden testaamisen Visual Studion Debug -toiminnolla.

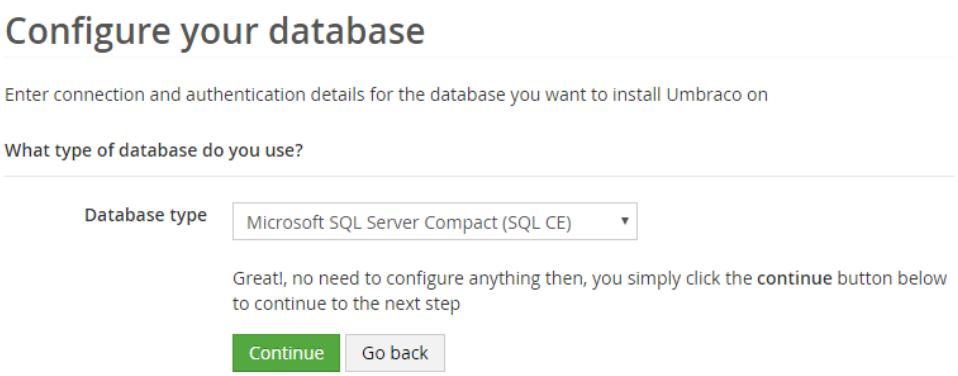
Asensin Visual Studioon myös työkaluja, jotka nopeuttavat sovelluksen luomista. Ensimmäisenä työkaluna asensin Web Compilerin, joka koostaa esimerkiksi LESS-tiedoston automaattisesti css-tiedostoksi. Toiseksi työkaluksi asensin Web Essentialssin, jonka avulla pystyin suoraan näkemään

tyylimuutokset ilman selaimen jatkuvaa päivittämistä. Molempia työkaluja olin ennenkin käyttänyt töissä, joten niiden hyödyntäminen opinnäytetyössä oli itsestäänselvyys.

4.1.2 Umbracon asennus

Projektin rungon oltua valmis oli seuraavaksi vuorossa Umbraco-sisällönhallintajärjestelmän asentaminen NuGetin avulla. Projektia luodessa uusin Umbraco-versio oli 7.5.3, joten luonnollisesti asensin edellä mainitun version tähän sovellukseen. Visual Studio tarjoaa NuGetista sekä konsoli- että graafista järjestelmää. Umbracoa asentaessa ei ollut tarvetta käyttää konsolijärjestelmää, koska graafisesta järjestelmästä löytyi kaikki tarvittavat ominaisuudet.

Itse Umbracon asentaminen on tehty todella vaivottomaksi NuGetin avulla. Ohjelmistokehittäjän täytyy vain valita haluttu versio Umbracosta ja tämän jälkeen painaa Asenna -nappia, minkä jälkeen NuGet hakee ja asentaa automaattisesti tarvittavat liitännäiset sisällönhallintajärjestelmää varten. Kun NuGet on hakenut ja asentanut tarvittavat liitännäiset on aika käynnistää web-sovellus, jossa pääsen määrittelemään Umbracossa käytettävän tietokannan (Kuva 4).

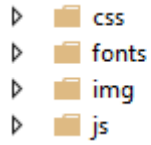


Kuva 4. Tietokannan valitseminen (Teemu Heinonen, 2017)

Päädyin hyödyntämään Microsoft SQL CE -tietokantaa sovelluksessa, koska asiakkaalla ei ollut tarvetta tehokkaammalle tai monipuolisemmalle tietokantajärjestelmälle. Kun haluttu tietokantajärjestelmä on valittu, Umbraco aloittaa automaattisesti tarvittavien taulujen luomisen. Näitä ovat muun muassa käyttäjät- sekä sisältötaulut. Taulujen luonnin jälkeen Umbraco-sisällönhallintajärjestelmä on onnistuneesti asennettu ja käyttäjän eteen aukeaa Umbracon ylläpitosivu.

4.1.3 Tarvittavien liitännäisten asennus

Liitännäisiä varten on hyvä luoda kansiot JavaScript-, LESS -tiedostoille, kuville sekä fonteille (Kuva 5). Kansiot nimetään lähes poikkeuksetta aina englanniksi, jolloin toinen ohjelmistokehittäjä osaa löytää tarvittavat tiedostot vaivattomasti.



Kuva 5. Kansiot liitännäisiä varten (Teemu Heinonen, 2017)

Seuraavaksi on aika ladata tarvittavat liitännäiset sovellusta varten. Aloitin lataamalla Materializen uusimman version Materializen omilta sivuilta:

http://materializecss.com/getting-started.html

Zip-tiedosto sisältää JavaScript-, CSS sekä fonttitiedostot. Näistä JavaScript- sekä CSS-tiedostot on jo valmiiksi kompressoitu pienemmiksi, mikä lyhentää sivujen latautumisaikaa. Siirsin zip-tiedostosta löytyvät tiedostot oikeisiin kansioihin, minkä jälkeen latasin Chosen liitännäisen GitHubista:

https://github.com/harvesthq/chosen

Chosen liitännäisen avulla pystyin tekemään moderneja *select* –laatikoita (Kuva 6). Päädyin hyödyntämään Chosenia siksi, että asiakkaalla saattoi olla monta erinimistä projektia työajanhallintajärjestelmässä. Kun käyttäjä kirjaa tunnin halutulle projektille on huomattavasti helpompi hakea hakusanalla haluttu projekti kuin käydä kaikki projektit läpi manuaalisesti oikean löytämiseksi.

Kuva 6. Vasemmalla normaali select–laatikko ja oikealla Chosen-laatikko (Chosen, n.d.)

Jotta edellä mainitut liitännäiset toimisivat oikein, vaadittiin myös jQuery-tiedosto, joka on avoimeen lähdekoodiin perustuva JavaScript-kirjasto (W3Schools, n.d.). Tässä projektissa päädyin käyttämään kompensoitua

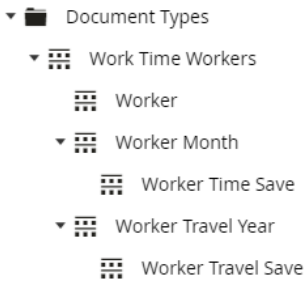
jQueryn versiota 2.1.1. Kyseisen version jQuerystä sai ladattua heidän omilta kotisivuiltaan osoitteesta:

https://code.jquery.com/jquery/

Tarvittavat liitännäiset sovellusta varten oli nyt ladattu sekä asetettu oikeisiin kansioihin. Luvussa 4.1.5 kerron tarkemmin, kuinka liitännäiset otettiin käyttöön web-sovelluksessa.

4.1.4 Dokumenttityyppien sekä näkymien luominen

Dokumenttityyppi on datan säilytyspaikka Umbracossa, jonka avulla käyttäjä voi muokata sivun sisällä esiintyviä arvoja, esimerkiksi otsikoita tai käyttäjän työtunteja. Sovelluksessa oli tarvetta kolmelle eri dokumenttityyppiluokalle; työntekijöiden tuntien, projektien tietojen sekä verkkosivujen datalle. Ensiksi loin dokumenttityypeille pääluokat, joiden sisälle asetin lapsiluokat.

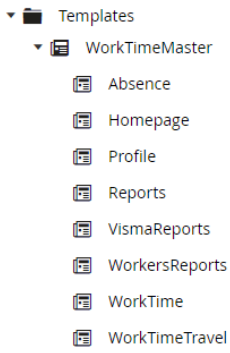


Kuva 7. Työntekijöiden tuntien dokumenttityypin runko (Teemu Heinonen, 2017)

Kuten kuvasta 7 voi havainnoida, pääluokan Work Time Workers alle on luotu kolme lapsiluokkaa; Worker, Worker Month, sekä Worker Travel Year. Näistä Worker sisältää työntekijän nimen sekä henkilökohtaisen ID:n. Worker Monthin alla olevassa Worker Time Save -luokassa säilytetään työtuntien dataa, kuten muun muassa työpäivä, tehdyt tunnit, asiakas ja selite projektille. Worker Travel Save sisältää datan matka- ja kulukorvauksille.

Loin samalla tavalla dokumenttityypit myös projektien tietojen sekä verkkosivujen datalle. Projektin tietojen sisälle käyttäjällä on mahdollista lisätä asiakkaita, projekteja sekä tuotteita. Verkkosivujen dokumenttityyppi sisälsi taas erilaisia sivuja, kuten profiili-, työtunti-, matka- ja kulukorvaus- sekä raportit sivun.

Verkkosivujen dokumenttityyppi vaati näkymän, jotta itse sisältö näkyisi verkkosivulla. Ensin luotiin pääluokka, jonka alle luotiin lapsiluokkia. Asetin lapsiluokat perimään pääluokan rungon, jotta tätä ei tarvitsisi tehdä useaan kertaan.



Kuva 8. Verkkosivun näkymien luonti (Teemu Heinonen, 2017)

4.1.5 Sovelluksen perusrunko

Aikaisemmassa luvussa luodulle näkymän pääluokalle tuli luoda runko, jota muut näkymät perivät. Aluksi määrittelin liitännäiset toimimaan verkkosivuilla kutsumalla niitä komennolla *Html.RequireJs* sekä *Html.RequireCss.* Tämä ei kuitenkaan vielä tulostanut liitännäisiä projektin runkoon. Komennoilla *html.RenderCssHere()* ja *Html.RederJsHere()* sain liitännäiset pakattua yhteen tiedostoon. Tämä nopeuttaa sivun latausaikaa sekä selkeyttää sivun runkoa.

Sivun otsikko haettiin aina käyttäjän tämän hetkisen sivun perusteella olevasta dokumenttityypistä ja mikäli tätä ei löytynyt hyödynnettiin pääluokan otsikkoa. Jotta verkkosivun lapsiluokat voisivat hyödyntää pääluokan runkoa, tuli tiedostolle antaa komento *RenderBody(),* joka hakee tarvittavan lapsiluokan sisällön ja tulostaa tämän näkymän käyttäjälle.

WorkTimeMaster.cshtml*

```
@inherits Umbraco.Web.Mvc.UmbracoTemplatePage
@using umbraco.NodeFactory
@using ClientDependency.Core.Mvc
@{
    Layout = null;

    var home = new Node(CurrentPage.AncestorOrSelf(1).Id);

    Html.RequiresJs("~/js/jquery-2.1.1.min.js", 1);
    Html.RequiresJs("~/js/materialize.min.js", 2);
    Html.RequiresJs("~/js/materialize.clockpicker.js", 3);
    Html.RequiresJs("~/js/chosen.jquery.min.js", 4);
    Html.RequiresJs("~/js/customize.js", 5);

    Html.RequiresCss("~/css/materialize.min.css", 1);
    Html.RequiresCss("~/css/materialize.clockpicker.css", 2);
    Html.RequiresCss("~/css/chosen.min.css", 3);
    Html.RequiresCss("~/css/style.min.css", 4);
}

<!DOCTYPE html>
<html lang="fi">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    @Html.RenderCssHere()

    <title>@((CurrentPage.pageTitle != "") ? (CurrentPage.pageTitle) : (home.GetProperty("pageTitle")))</title>

</head>
<body>

    @RenderBody()

    @Html.RenderJsHere()

</body>
</html>
```

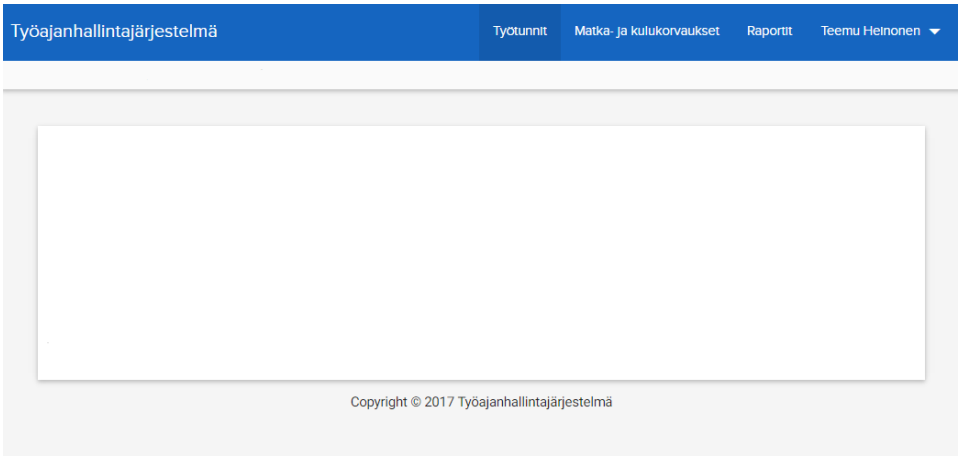
Kuva 9. Verkkosivun perusrunko (Teemu Heinonen, 2017)

Perusrungon valmistuttua pystyin tekemään lapsiluokkien näkymät. Lapsiluokat perivät pääluokan rungon, minkä johdosta ei liitännäisiä tarvinnut määritellä kuin kerran. Ohjelmistokehittäjän näkökulmasta tämä mahdollistaa selkolukuisemman koodin sekä helpottaa jatkokehitystä, koska suoraan runkoon voidaan lisätä koodia.

4.1.6 Ulkoasun suunnittelu ja toteutus

Kuten vaatimusmäärittelyissä olimme määritelleet, tuli sivujen ulkoasun olla tyyliltään samankaltainen kuin laskutusohjelman. Näin tekemällä asiakkaalle ei syntyisi tunnetta, että kyseessä olisi kaksi toisistaan täysin poikkeavaa järjestelmää. Toteutin ulkoasun pääsääntöisesti itse, mutta sain ohjenuoria opinnäytetyön kanssa yhteistyössä olleelta mainostoimistolta, jolloin lopputuloksena syntyi moderni ja responsiivinen järjestelmä.

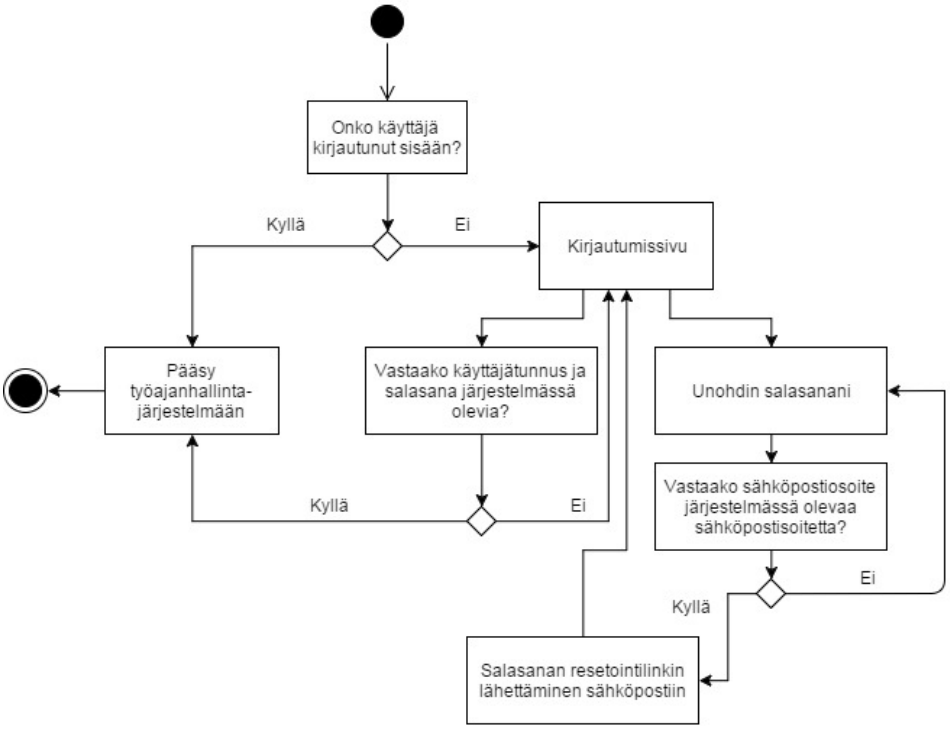
Toteutin ulkoasun suoraan ohjelmointikielellä, koska projektin laajuuden takia minulle ei jäänyt aikaa luonnosten tekemiseen. Ongelmia kyseinen ratkaisu ei aiheuttanut minulle eikä asiakkaalleni, koska olin saanut vapaat kädet ulkoasun toteutuksen suhteen. Tämä oli myös minulle luonnollinen tapa toteuttaa ulkoasu, koska näin olen tehnyt muidenkin projektien kanssa. Alla olevasta kuvasta näkyy sovelluksen ulkoasun suunnittelun jälkeinen toteutus.



Kuva 10. Työajanhallintajärjestelmän ulkoasun lopputulos (Teemu Heinonen, 2017)

4.2 **Kirjautumissivu**

Työajanhallintajärjestelmään pääsy haluttiin rajata niin, että ulkopuoliset henkilöt eivät pääsisi järjestelmään sisälle. Sovelluksella oli siis tarvetta käyttäjille. Onneksi tämä oli sisäänrakennettu ominaisuus Umbracon sisällönhallintajärjestelmässä. Kirjautumissivulla tuli olla myös mahdollista uusia unohtunut salasana. Kirjautumissivu toteutettiin kuvan 11 mukaisen kaavion pohjalta.

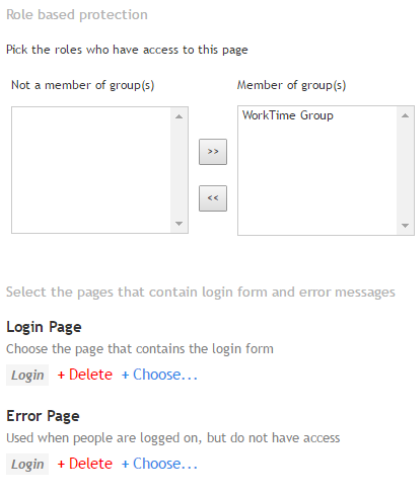


Kuva 11. Kirjautumissivun rakenne kaaviona (Teemu Heinonen, 2017)

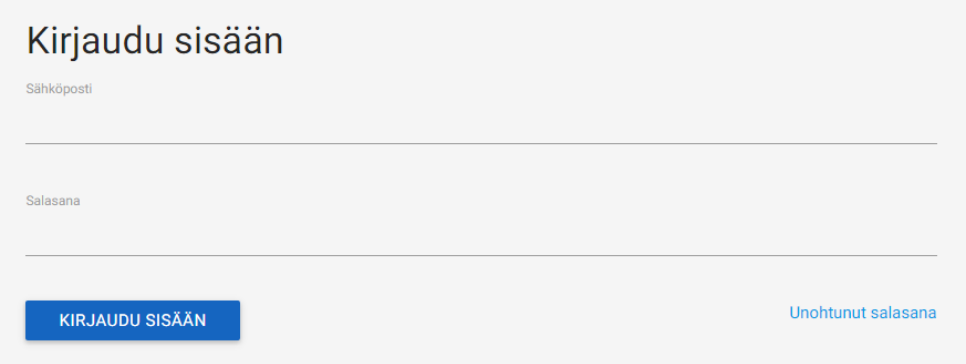
Itse toteutus tehtiin MVC-mallin mukaisesti lähettämällä käsittelijälle käyttäjän kirjautumistiedot ja palauttamalla näkymään joko työajanhallintajärjestelmä tai kirjautumissivu, mikäli tunnukset eivät täsmänneet.

Asiakas hankki itselleen myös SSL-sertifikaatin, joka parantaa kirjautumissivun turvallisuutta estämällä salasanojen kaappauksen käyttäjän ja palvelimen välissä.

Työajanhallintajärjestelmään pääsy tuli vielä estää Umbracon sisällönhallintajärjestelmästä (kuva 12). Valitsin käyttäjäryhmän, jolla on oikeus päästä järjestelmään sekä kirjautumissivun (kuva 13), johon kaikki käyttäjät ohjataan ennen järjestelmään pääsyä.



Kuva 12. Pääsyn rajoittaminen työajanhallintajärjestelmään (Teemu Heinonen, 2017)



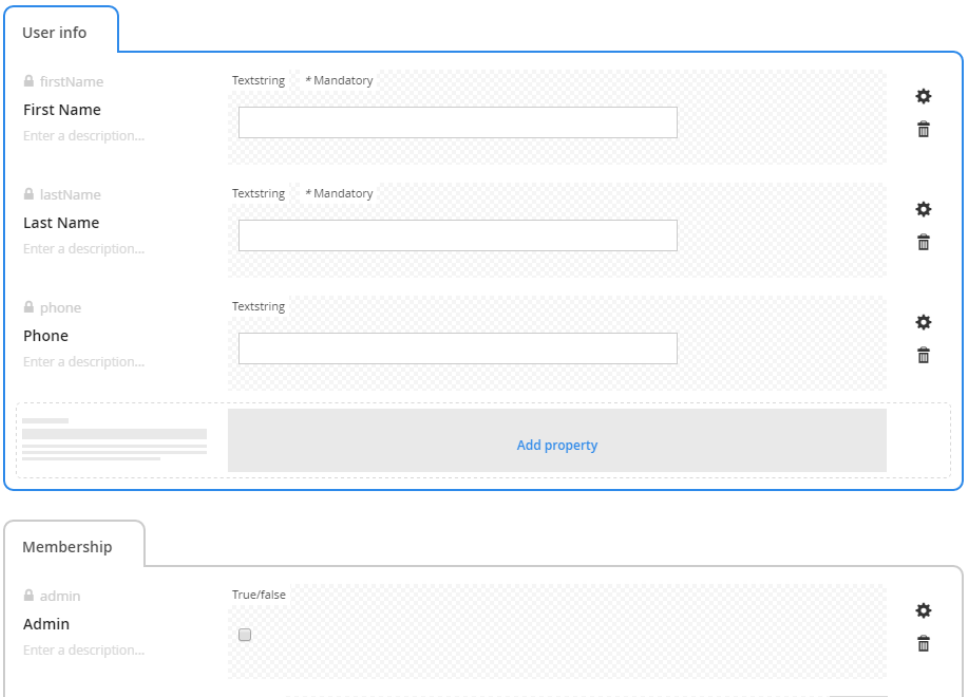
Kuva 13. Valmis kirjautumissivu (Teemu Heinonen, 2017)

4.2.1 Käyttäjäryhmän ja käyttäjän luominen

Käyttäjäryhmät ja käyttäjät luotiin Umbracon sisällönhallintajärjestelmässä. Käyttäjäryhmät luotiin, jotta vain tietyn ryhmän jäsenet pääsisivät työajanhallintajärjestelmään. Tällä ratkaisulla voidaan helposti säädellä, kenellä on oikeuksia millekin sivulle.

Loin uuden Käyttäjä-rungon käyttäjien luontia varten, koska Umbracon sisäinen ei ollut tarpeeksi kattava vaatimusmäärittelyä silmällä pitäen. Ylimääräisinä kenttinä lisäsin etunimen, sukunimen, puhelinnumeron se-

kä ylläpitäjä option. Mikäli käyttäjälle haluttiin antaa laajemmat oikeudet työajanhallintajärjestelmään, tuli Ylläpitäjä-optio ruksittaa. Tämä mahdollistaa muun muassa laajemman Raportti-näkymän.



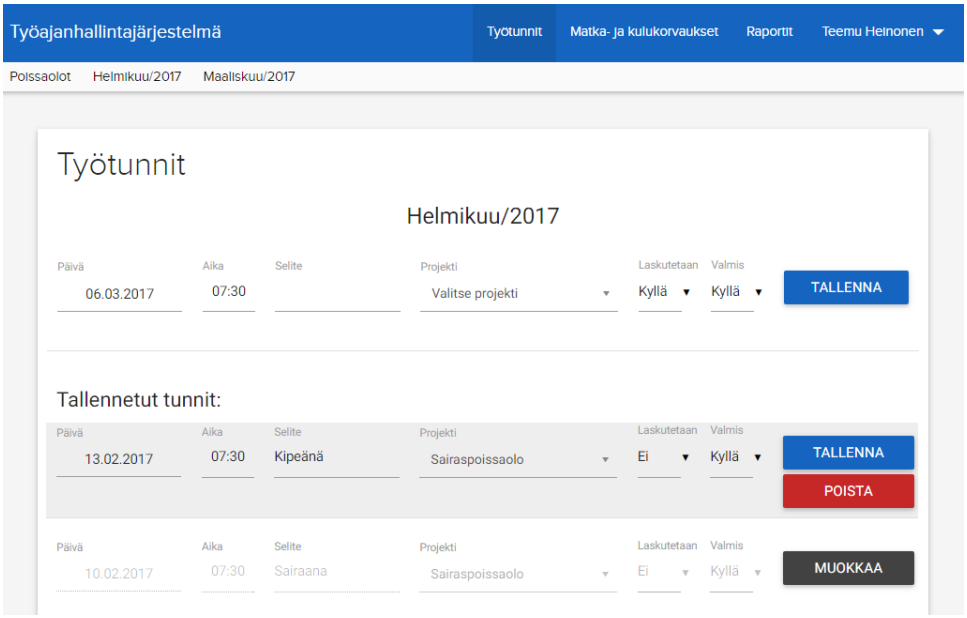
Kuva 14. Lisätyt kentät käyttäjien määrittämistä varten (Teemu Heinonen, 2017)

Kun Käyttäjä-runko oli valmis, voitiin järjestelmään lisätä käyttäjiä. Kirjautumistietoihin asetettiin sähköpostiosoite ja salasana sekä varmistettiin, että käyttäjä kuului aikaisemmin luotuun ryhmään. Mikäli käyttäjä ei pystyisi kirjautumaan sisään olisi todennäköisin syy siinä, että ryhmää ei ollut valittu käyttäjää luodessa.

4.3 Työtunnit-sivut

Varsinaista työajanhallintajärjestelmää lähdin toteuttamaan Työtunnit-sivusta. Vaatimusmäärittelmien mukaisesti sivu koostui seuraavista kentistä; päivämäärästä, työajasta, selitteestä, projektista, laskutuksesta ja valmiudesta. Asiakas tahtoi myös mahdollisuuden muokata ja poistaa työtunteja kahden kuukauden ajalta. Päädyin jakamaan Työtunnit-sivun kuukausien mukaan, jotta käyttäjällä olisi tarvittaessa helpompi löytää haluttu työpäivä.

Työtunnit-sivulla käyttäjälle aukeaa oletuksena sen hetkisen kuukauden näkymä. Kyseisen kuukauden tallennetut tunnit haetaan automaattisesti MVC-mallin mukaisesti (Kuva 15). Mikäli tunteja ei ole tallennettu järjestelmään, ei käyttäjälle näytetä mitään.



Kuva 15. Työtunnit-sivun toiminnallisuudet (Teemu Heinonen, 2017)

Työtuntia tallennettaessa tai muokattaessa tarkistetaan ensin JavaScriptillä, että lomakkeen kaikki kentät on täytetty. Mikäli työtunti täyttää tämän ehdon lähetetään data Post-metodin kautta käsittelijälle, joka varmistaa vielä kertaalleen, että kaikki kentät ovat täytettynä. Työtuntia muokattaessa käsittelijä varmistaa tämän lisäksi muun muassa sen, että käyttäjä muokkaa omia tuntejaan eikä kenenkään muun. Tarkistusten jälkeen data lähetetään mallille, joka tallettaa datan tietokantaan ja palauttaa uuden päivitetyn listan kyseisen kuukauden tunneista käsittelijälle. Tämä lista viedään näkymälle, joka tulostaa päivitetyn listan tallennetuista työtunneista sekä ilmoittaa käyttäjälle, että työtunnit tallennettiin järjestelmään onnistuneesti. Mikäli tallennus jostain syystä epäonnistuisi ilmoitettaisiin siitä myös käyttäjälle.

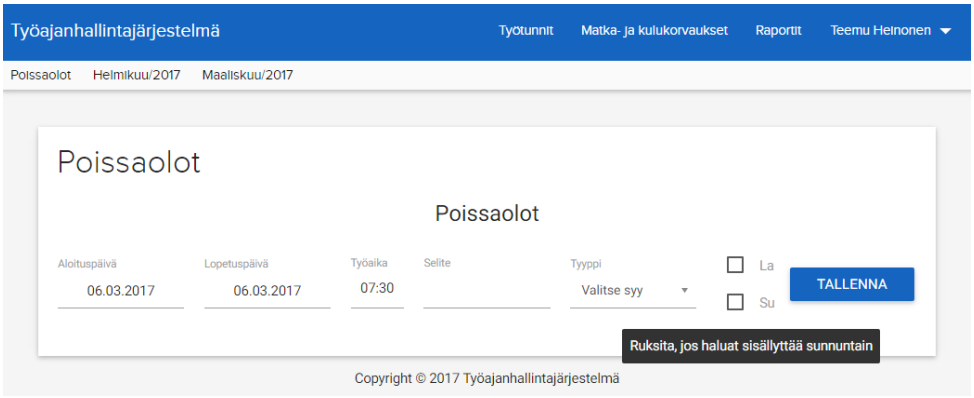
Työtunnin poistossa käsittelijä tarkistaa, että poistettava tunti on käyttäjän oma eikä kenenkään toisen. Mikäli tämä ehto pitää paikkansa, siirretään tallennettu tunti mallissa roskakoriin, jolloin ylläpitäjälle jää vielä tarvittaessa mahdollisuus palauttaa poistettu tunti.

4.3.1 Poissaolot

Asiakas oli toivonut vaatimusmäärittelyssä, että hän voisi määritellä pidempiä aikavälejä nopeasti ja vaivattomasti. Tästä syntyi idea Poissaolot-sivulle, jonka kautta käyttäjä pystyy määrittelemään työtunteja pidemmille aikaväleille. Esimerkiksi, jos käyttäjä on ollut sairaana viikon, hän voi merkitä koko viikon työtunnit helposti yhdellä kertaa.

Poissaolot-sivu (Kuva 16) koostui aloitus- ja lopetuspäivämääristä, työpäivän kestosta, selitteestä, poissaolon syystä sekä mahdollisuudesta sisällyttää lauantai tai sunnuntai aikavälille. Työtunnit tallentuvat automaatti-

sesti oikean kuukauden alle, eikä Poissaolot-sivun kautta pysty tallennettuja työtunteja muokkaamaan.



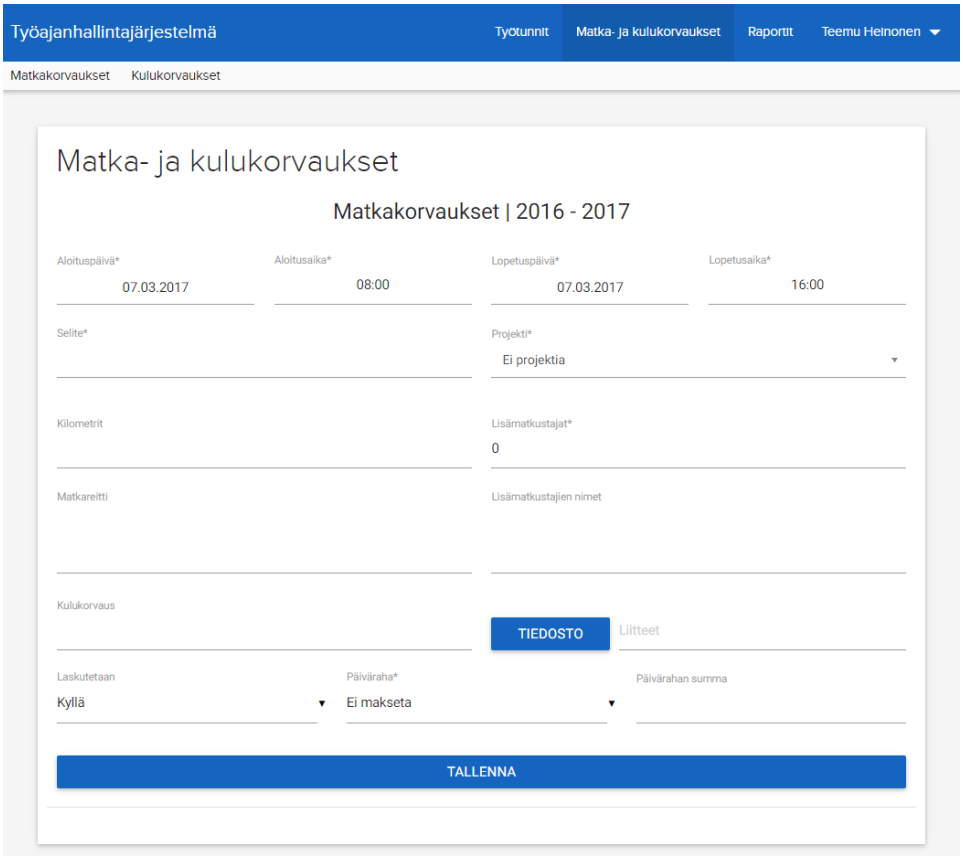
Kuva 16. Poissaolot -sivun näkymä (Teemu Heinonen, 2017)

Teknisesti sivu toimii samalla tavalla kuin Työtunnit-sivu. JavaScriptillä varmistetaan ensin, että kaikki kentät on täytetty. Tämän jälkeen data lähetetään Post-metodilla käsittelijälle, joka varmistaa datan paikkansa pitävyyden. Mikäli data läpäisee tarkastuksen, välittää käsittelijä datan mallille, joka tallentaa työtunnit yksitellen tietokantaan. Työtuntien tallennuksen jälkeen malli kuittaa käsittelijälle onnistuneen tallennuksen, joka vastaavasti välittää tämän tiedon näkymälle. Lopuksi käyttäjälle ilmoitetaan, että työtunnit on tallennettu järjestelmään ja lomake tyhjennetään.

4.4 Matka- ja kulukorvaukset -sivut

Asiakkaalla oli myös tarvetta matka- ja kulukorvauksien tallentamiselle. Päädyin vaatimusmäärittelyjen mukaisesti jaottelemaan matka- ja kulukorvaukset omiin sivuihinsa, jolloin korvausten tallentaminen olisi selkeämpää. Työajanhallintajärjestelmään on mahdollista lisätä ja muokata korvauksia kahden vuoden ajalta.

Matkakorvaus-sivulla (Kuva 17) olevassa lomakkeessa on vaatimusmäärittelmien mukaiset kentät. Näistä kentistä käyttäjän täytyy täyttää ainakin aloituspäivä, aloitusaika, lopetuspäivä, lopetusaika, selite, projekti sekä päiväraha. Muut kentät ovat valinnaisia eikä niitä tarvitse täyttää, mikäli niille ei ole tarvetta. Kulukorvaus-sivulla käyttäjän tarvitsee täyttää vain aloituspäivä, aloitusaika, selite sekä projekti. Vapaavalintaisia kenttiä Kulukorvaus-sivulla on laskutus, kulukorvaus sekä liitetiedostot.



Kuva 17. Matka- ja kulukorvaukset -sivun sisältö (Teemu Heinonen, 2017)

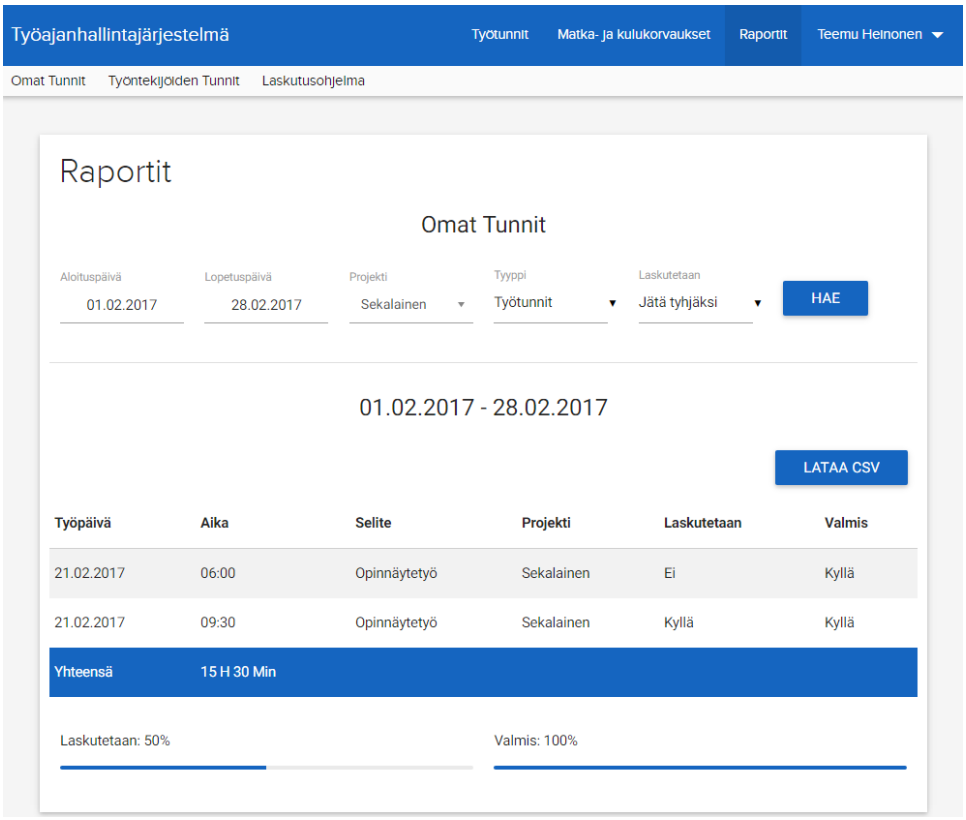
Tallennettaessa korvausta järjestelmään tarkistetaan ensin, että kaikki tarvittavat kentät on täytetty. Tämä tarkistus suoritetaan JavaScriptillä. Mikäli tarvittavat kentät on täytetty oikein, lähetetään lomakkeen data Post-metodilla käsittelijälle, jonka tehtävänä on tarkistaa data uudemman kerran. Kun datan paikkansapitävyys on tarkistettu, siirretään data malliin, jonka tehtävänä on tallentaa saatu data tietokantaan ja palauttaa käsittelijälle kaikki korvaukset kahden vuoden ajalta. Tämä data siirretään käsittelijältä näkymälle, jolloin käyttäjä näkee uuden korvauksen ilmestyvän sivulle sekä ilmoituksen siitä, että korvaus on tallennettu järjestelmään. Korvauksen poistaminen tapahtuu samalla tavalla kuin työtunnin poistaminen.

4.5 **Raportit-sivut**

Raportit-osiossa alanavigointinäkymä riippuu siitä, onko käyttäjä asetettu Umbraco-sisällönhallintajärjestelmästä ylläpitäjäksi vai ei. Mikäli käyttäjä on ylläpitäjä, voi hän katsoa kaikkien työntekijöiden työtunteja sekä siirtää työtunteja laskutusohjelmaan. Tavalliselta käyttäjältä on eristetty täysin laajempi Raportti-osio eikä hän näkisi mitään sisältöä itse sivulla, vaikka kirjoittaisi selaimen URL-kenttään oikean osoitteen.

Tavallisella käyttäjällä on mahdollisuus hakea joko omia työtunteja, matkakorvauksia tai kulukorvauksia tietyllä aikavälillä. Tämän lisäksi hän voi suodattaa hakutulosta projektin tai laskutuksen mukaan. Ylläpitäjällä on näiden toimintojen lisäksi mahdollisuus suodattaa raporttia tiettyjen työntekijöiden mukaan.

Kun hakuparametrit on annettu järjestelmään pyytää käsittelijä hakemaan mallilta oikeat tapahtumat näkymään (Kuva 18). Näkymästä on mahdollista ladata CSV-tiedosto sekä tarkistaa, kuinka monta tuntia työntekijä on tehnyt tietyllä aikavälillä. Lisäsin myös asiakkaan toiveesta mahdollisuuden nähdä, kuinka monta prosenttia töistä on laskutettavaa sekä missä tilassa työtunti on.

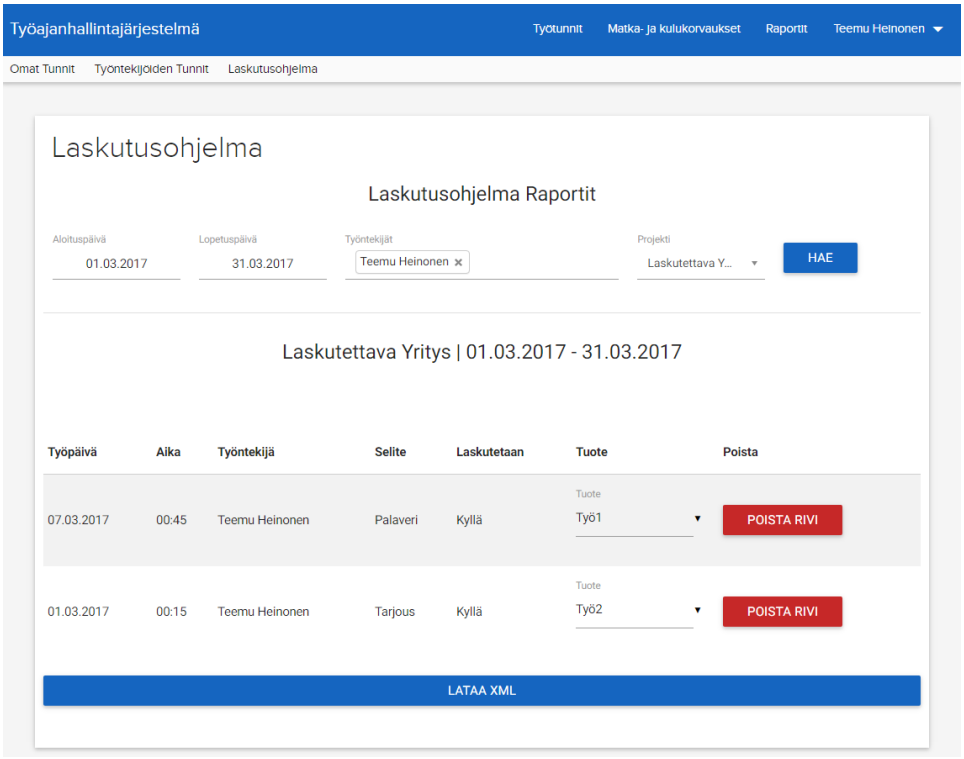


Kuva 18. Omien tuntien Raportti-näkymä (Teemu Heinonen, 2017)

4.5.1 Tietojen vienti laskutusohjelmaan

Asiakas halusi viedä työtunnit suoraan laskutusohjelmaan työajanhallintajärjestelmän kautta. Ennen kuin pääsin suunnittelemaan itse sivua, tuli minun varmistaa laskutusohjelman tekijöiltä, oliko heillä rajapintaa kyseiseen toimintoon. Selvisi, että heillä oli kahdenlaista rajapintatyyppiä; suora- tai manuaalista siirtoa. Sain laskutusohjelman tekijöiltä opinnäytetyötäni varten ilmaiseksi rajapinnat manuaalista siirtoa varten.

Kun rajapinnan olemassaolo oli saatu selvitettyä ja tarvittavat rajapinta-dokumentit oltiin toimitettu minulle, pystyin aloittamaan itse laskutusohjelman Rajapinta-sivun teon. Suunnittelin sivun ulkoasun niin, että käyttäjä valitsee ensin halutun aikavälin, minkä jälkeen hän valitsee työntekijät sekä projektin. Lomake palauttaa listan kaikista työtunneista, jotka vastaavat annettuja hakuparametreja (Kuva 19). Ennen XML-tiedoston lataamista voi käyttäjä vielä poistaa rivejä laskutuksesta sekä merkitä tiettyjä tunteja tietyn tuotekoodin alaiseksi. Näin laskutusohjelma osaa asettaa suoraan oikean hinnan työtunnille.



Kuva 19. XML-tiedoston lataussivu (Teemu Heinonen, 2017)

XML-tiedoston lataaminen lähettää käsittelijälle käskyn luoda XML-tiedosto käyttäjän syöttämillä asetuksilla. Käsittelijällä ei ole tarvetta käyttää mallia apunaan tässä tapauksessa, vaan lähettää valmiin XML-tiedoston suoraan käyttäjälle ladattavaksi. Kuva 20. on esimerkki validista XML-tiedostosta, jonka käyttäjä voi halutessaan lähettää laskutusohjelmaan.

```
<?xml version='1.0' encoding='ISO-8859-15'?>
<[redacted] XML xsi:noNamespaceSchemaLocation="https://asp.XXX.net/XXXX/XXXXX-XML.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <BusinessId>12345</BusinessId>
  <SalesOrder>
    <SalesOrderNo>1000</SalesOrderNo>
    <SalesOrderCompanyId>1000</SalesOrderCompanyId>
    <SalesOrderRow>
      <SalesOrderRowNo>1</SalesOrderRowNo>
      <SalesOrderRowProductId>200</SalesOrderRowProductId>
      <SalesOrderRowQuantity>3</SalesOrderRowQuantity>
    </SalesOrderRow>
    <SalesOrderRow>
      <SalesOrderRowNo>2</SalesOrderRowNo>
      <SalesOrderRowProductId>100</SalesOrderRowProductId>
      <SalesOrderRowQuantity>12.5</SalesOrderRowQuantity>
    </SalesOrderRow>
    <SalesOrderRow>
      <SalesOrderRowNo>3</SalesOrderRowNo>
      <SalesOrderRowProductId>500</SalesOrderRowProductId>
      <SalesOrderRowQuantity>6.5</SalesOrderRowQuantity>
    </SalesOrderRow>
  </SalesOrder>
</[redacted] XML>
```

Kuva 20. Validi XML-tiedosto (Teemu Heinonen, 2017)

XML-tiedostossa määritellään aluksi tarvittavat parametrit, kuten esimerkiksi enkoodaus. Taulukko 2. käsittelee kuvan 20 parametrit vielä tarkemmin.

Taulukko 2. Rajapinnan parametrit selitettynä

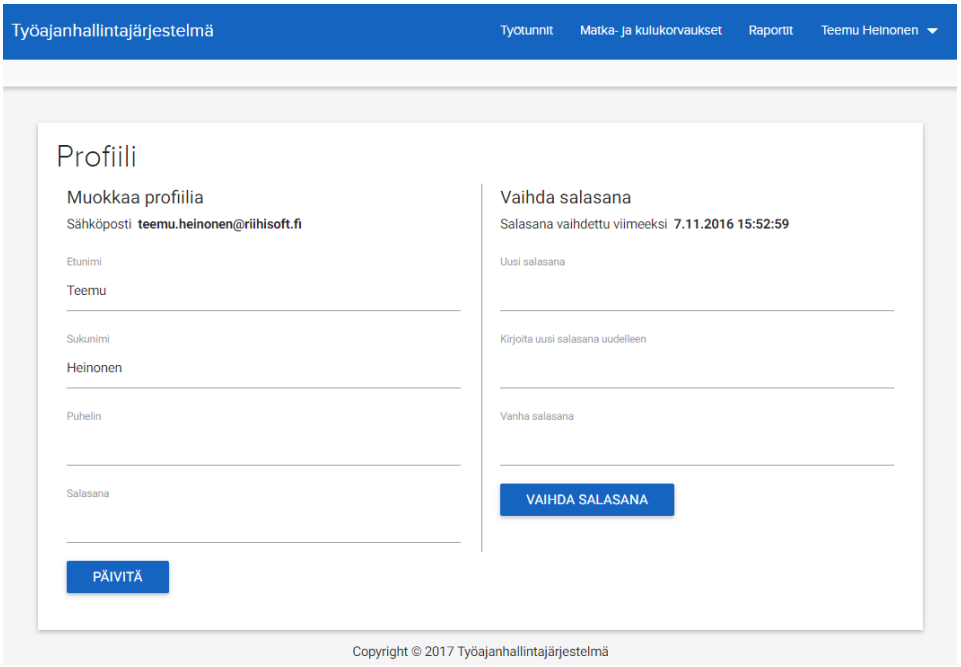
Parametrit	Selite
BusinessId	Vastaanottavan yrityksen yritystunnus laskutusohjelmassa.
SalesOrderNo	Rajapinnan luoma yksilöivä tieto.
SalesOrderCompanyId	Asiakkaan yritysnumero laskutusohjelmassa. Tämän avulla laskutusohjelma osaa hakea kaikki tiedot asiakasta koskien.
SalesOrderRowNo	Tilausrivin numero.
SalesOrderRowProductId	Tuotekoodi, joka vastaa laskutusohjelmassa olevaa.
SalesOrderRowQuantity	Tilauksen määrä.

Mikäli asiakas haluaa viedä XML-tiedoston laskutusohjelman rajapintaan manuaalisesti, pystyy hän tekemään sen laskutusohjelman omilla sivuilla. Laskutusohjelman rajapinta luo XML-tiedostosta automaattisesti uuden tilauksen annetun asiakkaan tiedoilla. Tämän lisäksi laskuun lisätään automaattisesti käytetyt tunnit, jolloin asiakkaalle jää tehtäväksi vain laskun hyväksyttäminen. Tämä nopeuttaa laskuttamista huomattavasti, kun tietoja ei tarvitse käydä syöttämässä käsin järjestelmään.

4.6 Profiilisivu

Käyttäjällä tuli olla mahdollisuus päivittää omia tietojaan kuten nimeä, puhelinnumeroa tai salasanaa. Omien tietojen päivittäminen tehtiin niin, että käyttäjältä vaaditaan aina omaa salasanaa. Näin tekemällä saadaan vähennettyä riskiä siitä, että ulkopuolinen henkilö päivittäisi toisen tietoja ilman hänen lupaansa. Päivitetyt tiedot tallennetaan automaattisesti Umbracon järjestelmään MVC-mallin mukaisesti. Umbracosta löytyi sisäänrakennettuna ominaisuutena tietojen päivittäminen, jolloin esimerkiksi salasanan kryptaamiselle ei tarvinnut tehdä omia toimintoja.

Päädyin erottamaan omien tietojen, kuten nimen ja puhelinnumeron, päivittämisen salasanan päivittämisestä alla olevan kuvan mukaisesti. Näin käyttäjä voi päivittää oman sukunimensä ilman, että hänen täytyy päivittää salasanaansa.



Kuva 21. Profiilisivun näkymä (Teemu Heinonen, 2017)

4.7 Sovelluksen viimeistelyt

Kun olin saanut sovelluksen siihen pisteeseen, että kaikki vaatimusmäärittelyn toiminnallisuudet oli saatu tehtyä, oli aika siirtyä sovelluksen viimeistelyyn. Viimeistelyyn kuului ulkoasun kehittäminen mainostoimiston kanssa, asiakkaalta tulleiden korjausten teko sekä valmiin sovelluksen toimittaminen asiakkaalle.

4.7.1 Ulkoasun kehittäminen mainostoimiston kanssa

Ensimmäinen vaihe oli olla yhteydessä opinnäytetyöprojektissa mukana olleeseen mainostoimistoon. Olin saanut heiltä jo aikaisemmin ohjeita ulkoasun suunnittelussa ja toteutuksessa, joten oli luonnollista olla heihin yhteydessä myös sovelluksen loppupuolella. Sain mainostoimistolta arvokasta palautetta liittyen toiminnollisuuksiin ja ulkoasuun. Esimerkiksi sovelluksen fontit oli hyvä vaihtaa Material Designiin pohjautuviin fontteihin.

Kävimme yhdessä läpi sovelluksen yhteensopivuutta eri selaimilla ja huomasimmekin, että Safarin verkkoselaimella Päivänvalinta-ikkuna ei toiminut kuten piti. Tämän sai onneksi korjattua helposti muuttamalla valinnan tyylimäärityksiä. Mainostoimisto ehdotti myös, että työtuntia tallennettaessa olisi hyvä, jos käyttäjälle ilmaantuisi lataus -ikoni siksi aikaa, kun tuntia tallennetaan järjestelmään. Toteutin mainostoimiston ehdotuksen JavaScriptiä hyödyntäen.

Mainostoimiston kanssa oli helppo käydä läpi ulkoasua, koska olin tehnyt heidän kanssaan yhteistyötä muissa projekteissa. Aikaisemmista projekteista olin myös oppinut, mihin asioihin kannattaa kiinnittää huomiota web-sovelluksia tehdessä. Tämä tieto auttoi minua myös paljon opinnäytetyön ulkoasua tehdessä.

4.7.2 Asiakkaalta tulleet korjauskehotukset

Asiakas pääsi käyttämään sovellusta ennen lopullisen version toimittamista. Tämä antoi minulle mahdollisuuden kehittää sivua edelleen sekä korjata tarvittaessa toimintoja, mikäli jokin ei jostain syystä toimisi kuten sen pitäisi.

Suurempia ongelmia sovelluksen testausvaiheessa ei esiintynyt, mutta mobiilikäyttöjärjestelmillä Projekti-valintaa ei ollut mahdollista valita. Ongelman syyksi ratkesi Chosen liitännäisen sisäänrakennettu ominaisuus, jossa laajennuksen käyttö oltiin estetty kaikilla mobiililaitteilla. Muutin Chosen liitännäisen JavaScript-koodia toimimaan niin, että myös mobiililaitteilla kyseinen laajennus olisi käytettävissä.

Asiakas toivoi myös, että käyttäjä pysyisi kirjautuneena sovelluksessa pidempään. Oletuksena käyttäjä pysyi kirjautuneena 20 minuuttia, mutta määrittelin asiakkaan toiveiden pohjalta aikarajoituksen 60 minuuttiin. Tämä muutos tehtiin web.config-tiedostoon, mikä löytyy jokaisesta ASP.NET web-sovelluksesta.

Raportit-sivulla oli määritelty aikaväli hakeutumaan kuukauden ensimmäisen ja viimeisen päivän mukaan automaattisesti. Asiakkaalla oli kuitenkin tarvetta nähdä, kuinka paljon hän oli tehnyt töitä kyseisenä päivä-

nä, joten päädyin vaihtamaan aikavälin näyttämään aina sen hetkistä päivää. Tämä muutos oli helposti toteutettavissa muokkaamalla näkymää.

4.7.3 Valmiin sovelluksen toimittaminen

Asiakkaalta tulleiden korjauskehotusten sekä kehittämisideoiden toteutusten jälkeen siirsin valmiin version työajanhallintajärjestelmästä asiakkaan omaan Windows-pohjaiseen palvelimeen FTP-yhteyden avulla. FTP-yhteys mahdollistaa tiedostojensiirron kahden tietokoneen välillä TCP-protokollaa hyödyntäen. Tarvittavat FTP-tunnukset sain asiakkaan palveluntarjoajalta. (Afterdawn, n.d.)

Kun tiedostot oli siirretty palvelimelle, tarkistin sovelluksen toimivuuden vielä uudelleen. Näin tekemällä varmistin, että kaikki tarvittavat tiedostot olivat todella päätyneet palvelimelle saakka eikä tiedostoja ollut hävinnyt FTP-siirron aikana bittiavaruuteen.

Asiakas sai sovelluksen käyttöönsä maksutta sellaisena, kuin se oltiin hänelle toimitettu. Sovelluksen toimittamisen jälkeen sovellusta ei enää ylläpidetä, korjata tai jatkokehitetä ja kaikki mahdolliset käyttökustannukset tulevat asiakkaalle. Asiakkaalle toimitettiin käyttöohjeet järjestelmän käyttöä varten (Liite 1). Ohjeiden avulla hän pystyy muun muassa tarkistamaan, miten uusia käyttäjiä, asiakkaita, projekteja tai tuotteita lisätään järjestelmään.

5 YHTEENVETO

Opinnäytetyössäni käsittelin työajanhallintajärjestelmän kehittämistä vaihe vaiheelta sekä siinä käytettyjä teknologioita. Tämän lisäksi kävin läpi asiakkaalta tulleita vaatimusmäärittelyjä sekä aikataulutusta. Projekti itsessään toteutui aikataulun puitteissa, vaikka tehtävää valmiin järjestelmän toteuttamisessa oli ollut todella paljon.

Sovelluksen toteuttaminen oli minulle suhteellisen suoraviivaista, koska olin käyttänyt teknologioita jo aikaisemmin töissä. Tämän lisäksi minulla oli hyvä tukiverkosto ja mahdollisissa ongelmatilanteissa sain apua työkavereiltani.

Sovelluksen teon aikana suurin kohtaamani ongelma oli tietokannan korruptio. Tämä tapahtui sovelluksen teon alkuvaiheessa. Onneksi minulla oli järjestelmästä varmuuskopioita, mitkä mahdollistivat pääsyni takaisin järjestelmään. Ratkaisuksi päädyin aloittamaan täysin uuden projektin, koska tietokannassa ollut korruptio oli syntynyt jo Umbracon asennusvaiheessa. Muita ongelmia sovelluksen kanssa ei esiintynyt. Osasyy tähän oli

se, että Umbracosta löytyi laajat dokumentaatiot, jolloin sisällönhallinta-järjestelmään liittyvät ongelmat löytyivät helposti.

Järjestelmään on tulevaisuudessa helppo lisätä uusia toimintoja, koska sovelluksen tekemisessä käytettiin uusinta tekniikkaa, kuten MVC:tä. Ohjelmistokehittäjän näkökulmasta on tärkeää, että sovellukset tehdään aina sen hetkisillä teknologioilla. Näin vältytään siltä, että sovellusta jouduttaisiin lähitulevaisuudessa muokkaamaan radikaalisti vastaamaan uusia standardeja.

Tavoitteenani oli luoda asiakkaalleni mahdollisimman yksinkertainen ja toimiva järjestelmä, josta myös asiakkaani pitäisi. Palautteen perusteella olin myös onnistunut toteuttamaan sovelluksen, joka vastasi asiakkaan toiveita ja tarpeita. Tämä oli minulle todella tärkeää, koska halusin tekemääni sovellusta käytettävän tulevaisuudessakin. Saatu lopputulos oli tavoitteeni mukainen ja siksi voinkin esittää tuotostani ylpeänä muille.

Kokemuksena opinnäytetyö oli todella opettavainen sekä vastuullinen. Sain johtaa ensimmäistä kertaa itse omaa projektia alusta loppuun, mitä toivoinkin opinnäytetyötä aloittaessa. Opin enemmän käytetyistä teknologioista sekä siitä, kuinka tärkeä SSL-sertifikaatti on tänä päivänä, mikäli dataa ei haluta kaapattavaksi oman tietokoneen ja palvelimen välillä. Työtä oli mukava toteuttaa ja se oli sopivan haasteellinen minulle.

LÄHTEET

Aitoa, (n.d.), Mikä on SSL-sertifikaatti, mikä https on ja miksi?, Viitattu 27.02.2017
http://www.aitoa.fi/ssl-sertifikaatti-mika-on-ja-miksi

Afterdawn, (n.d.), FTP, Viitattu 08.03.2017
http://www.download.fi/verkko/ftp/

Hyer G, 06.12.2016, 2017 is the year of SSL and https for websites, Viitattu 27.02.2017
https://www.theedesign.com/blog/2016/2017-year-ssl-https-websites

Allen K, Galloway J, Matson D & Wilson B (2014), Professional ASP.NET MVC 5
http://www.cs.unsyiah.ac.id/~frdaus/PenelusuranInformasi/File-Pdf/Professional%20ASP.NET%20MVC%205.pdf

Materialize, (n.d.), About, Viitattu 27.02.2017
http://materializecss.com/about.html

MacDonald M (2012), Beginning ASP.NET 4.5 in C#
https://books.google.fi/books?id=JalZGsmf9hwC&printsec=frontcover&hl=fi&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

Microsoft, (n.d.), NuGet Documentation, Viitattu 27.02.2017
https://docs.microsoft.com/fi-fi/nuget/

Microsoft, (n.d.), Welcome to Visual Studio 2015, Viitattu 27.02.2017
https://msdn.microsoft.com/en-us/library/dd831853.aspx

Tiffany R, 07.07.2010, Here Comes SQL Server Compact 4.0, Viitattu 27.02.2017
http://robtiffany.com/here-comes-sql-server-compact-4-0/

Umbraco, (n.d.), About us, Viitattu 27.02.2017
https://umbraco.com/about-us

W3Schools, (n.d.), jQuery Introduction, Viitattu 02.03.2017
https://www.w3schools.com/jquery/jquery_intro.asp

LIITTEET

Liite 1/1

Asiakkaalle toimitetut käyttöohjeet

ruhisoft

6.3.2017

Työajanhallintajärjestelmä

Versio:	0.1
Päivämäärä:	24.02.2017
Tekijä:	Teemu Heinonen

6.3.2017

SISÄLLYSLUETTELO

6.3.2017

1 MUUTOSHISTORIA

Henkilö	Päiväys	Versio	Kommentti
Teemu Heinonen	24.2.2017	0.1	Dokumentti luotu

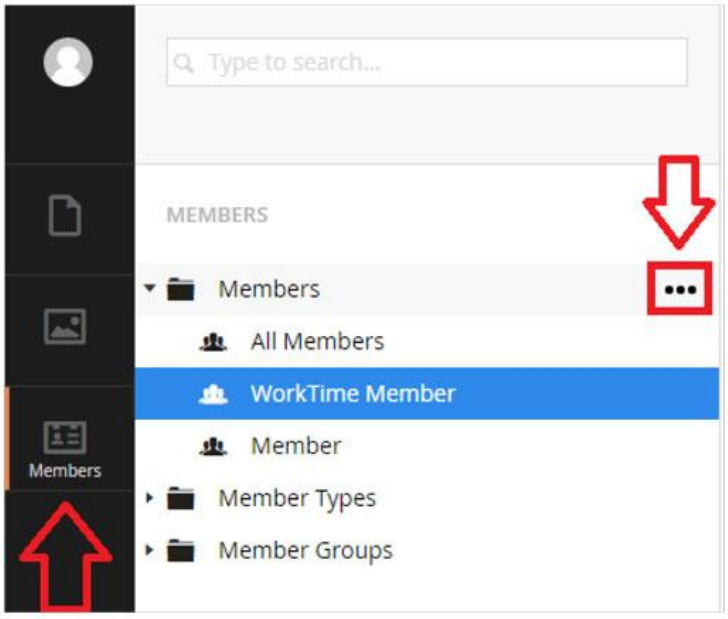
2 JOHDANTO

Tässä dokumentissa opastetaan, kuinka työajanhallintajärjestelmän ylläpitopuoli toimii. Umbracon ylläpitoon pääsee kirjautumaan osoitteessa: https://omasivu.fi/umbraco/ omilla henkilökohtaisilla tunnuksilla.

3 SISÄLTÖ

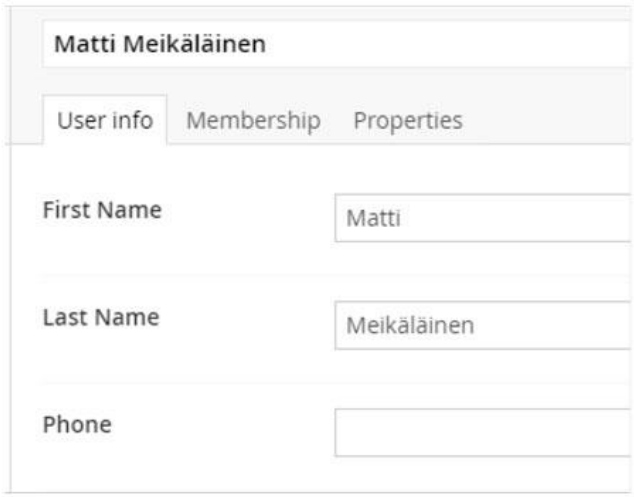
3.1 Käyttäjän lisääminen

Käyttäjä lisätään valitsemalla vasemmalta ”Members” –välilehti sekä painamalla kolmea pistettä ”...” Members ikkunan kohdalla.



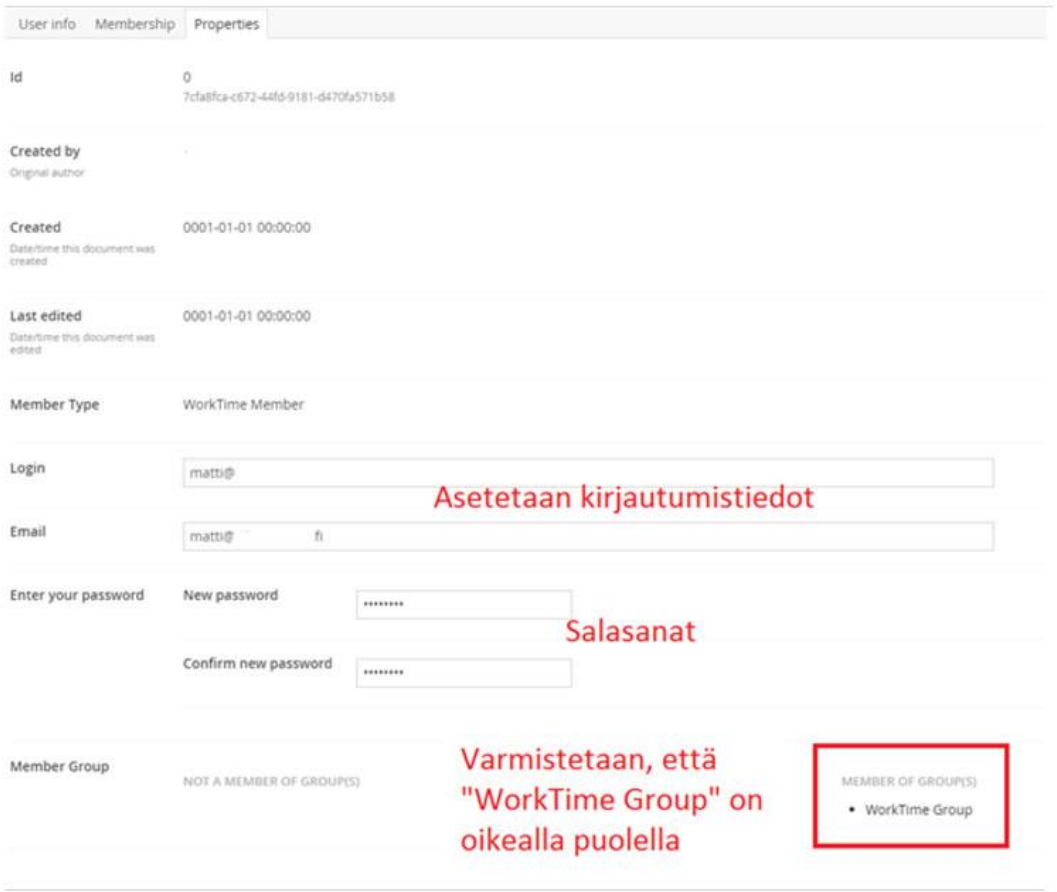
6.3.2017

Oikealle aukeaa ikkuna, josta valitaan ”WorkTime Member”. Täytetään kentät:



Liite 1/5

6.3.2017

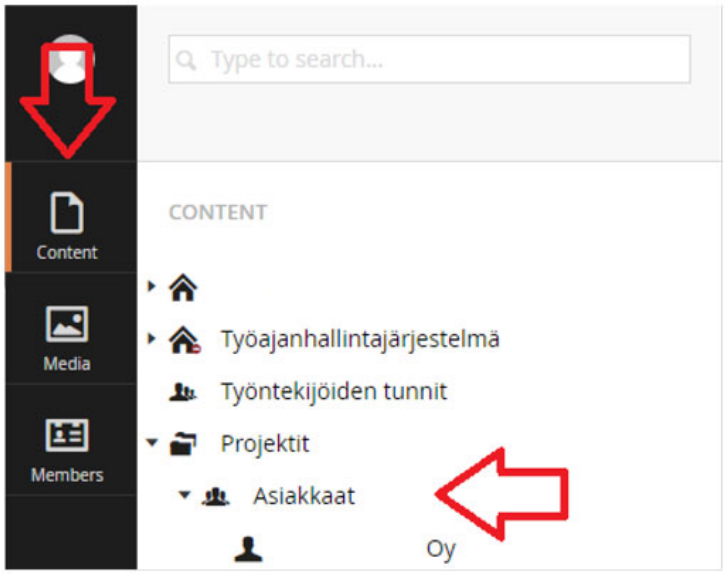


Tämän jälkeen painetaan ”Save” -nappia oikeasta alakulmasta, jonka jälkeen käyttäjä on luotu työajanhallintajärjestelmään.

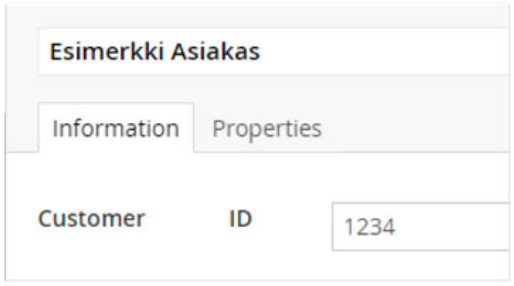
6.3.2017

3.2 Asiakkaan luominen

Asiakkaita ei tarvitse enempää luoda, mutta mikäli jostain syystä tälle tulee tarvetta niin ”Content” – välilehdeltä valitaan Projektit -> Asiakkaat:



Painetaan kolmea pistettä ”...” asiakkaat laatikon oikeassa reunassa ja valitaan oikealle aukeavasta ikkunasta ”Work Time Customer”. Tämän jälkeen täytetään tarvittavat kentät ja tallennetaan oikeassa alakulmassa olevasta ”Save and publish” -napista.



Customer ID vastaa laskutusohjelmassa olevaa yritystunnusta

3.3 Projektin luominen

Projekteja on kahti erityyppiä; laskutettavaa ja ei laskutettavaa. Valitaan listalta haluttu projekti tyyppi ja painetaan kolmea pistettä ”...”. Oikealle aukeaa ikkuna, josta valitaan ”Work Time Project”. Täytetään kentät alla olevan ohjeen mukaan:

ruhtisoft

6.3.2017

Project Sales Order Company ID = Yritystunnus

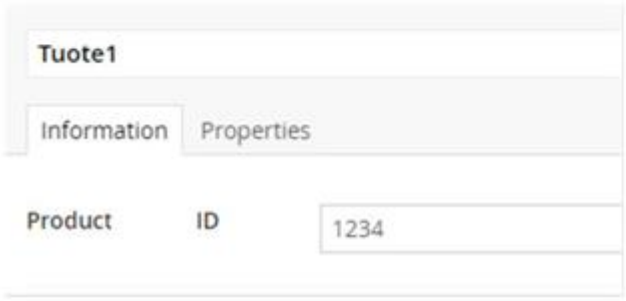
Lopulta painetaan oikeasta alakulmasta “Save and publish” -nappia.

ruhtisoft 6.3.2017

3.4 Tuotteen lisääminen

Tuoteen lisääminen tapahtuu valitsemalla "Content" -välilehdeltä Projektit -> Tuotteet. Painetaan kolmea pistettä "..." ja valitaan aukeutuvata ikkunasta "Work Time Product".

Tämän jälkeen täytetään kentät alla olevan kuvan mukaisesti:



Product ID = Tuotekoodi

Kun kentät on täytetty, painetaan "Save and publish" –nappia oikeasta alakulmasta.