SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

ASIAKKUUKSIEN HALLINTAJÄRJESTELMÄN TOTEUTUS

TEKIJÄ: Kimmo Turunen

SAVONIA-AMMATTIKORKEAKOULU
OPINNÄYTETYÖ

Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Kimmo Turunen	
Työn nimi Asiakkuuksien hallintajärjestelmän toteutus	
12.03.2019	Sivumäärä/Liitteet 21
Ohjaaja(t) Jukka Kinnunen ja Pasi Liimatainen	
Toimeksiantaja/Yhteistyökumppani(t) Data Group Kuopio / Savon Tietokeskus Oy	

Tiivistelmä

Opinnäytetyön tilasi Savon Tietokeskus Oy, joka tarjoaa tietotekniikan ylläpitopalveluja yrityksille. Opinnäytetyön tavoitteena oli luoda järjestelmä, johon saadaan tallennettua ylläpitoasiakkaat, heidän palvelut sekä järjestelmädokumentaatiot. Asiakkuuksien hallintajärjestelmään kuuluu myös tiketöintijärjestelmä ja työajankirjaus. Järjestelmän tavoite on yksinkertaistaa asiakkaiden hallintaa sekä helpottaa ympäristöjä ylläpitävien asentajien työntekoa.

Työ toteutettiin intranetissä ylläpidettävänä ASP.NET Core web-sovelluksena. C#-pohjaisen intranettiin kuuluu virtuaalinen WWW- sekä tietokantapalvelin, joka ylläpitää ohjelmiston web-sovellusta ja tietokantaa.

Projekti jaettiin asiakkuuksien hallintaan, tiketöintijärjestelmään ja ympäristöjen dokumentoinnin suunnitteluun. Ensimmäiseksi suunniteltiin millä tavalla asiakkaat listataan, mitä dataa heistä halutaan tallentaa sekä raportoida ja kuinka em. data näytetään loppukäyttäjälle. Tämän päälle rakennettiin tiketöintijärjestelmä ylläpidon työkaluksi. Asiakkaat pystyvät lähettämään järjestelmään tukipyyntöjä, jotka työnjohto pystyy järjestelmän avulla jakamaan Savon Tietokeskus Oy:n asentajille. Asentajien avuksi järjestelmään tallennetaan myös asiakkaiden tietojärjestelmien dokumentaatio. Ylläpitoasiakkaiden ympäristöjen dokumentaatiotapa suunniteltiin uudestaan. Ympäristöjen dokumentaatiot muutettiin taulukkomuotoon, jotta niitä pystytään käsittelemään relationaalisessa tietokannassa. Tietokannan sisältöä voidaan tarkastella ja muokata web-sovelluksen kautta.

Opinnäytetyön tuloksena Savon Tietokeskus Oy sai täyden asiakkuuksien hallintajärjestelmän, johon kuuluu myös tiketöintijärjestelmä. Järjestelmä kehitettiin helpottamaan asentajien työtehtäviä sekä parantamaan tukiasiakkaista saatavaa tuottoa.

Avainsanat
Asiakkuuksienhallinta, web-sovellus, MVC-malli

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Computer Science			
Author(s) Kimmo Turunen			
Title of Thesis Implementation of a Customer Relationship Management System			
Date	12 March 2019	Pages/Appendices	21
Supervisor(s) Jukka Kinnunen, Pasi Liimatainen			
Client Organisation /Partners Data Group Kuopio / Savon Tietokeskus Oy			

Abstract

Thesis was commissioned by Savon Tietokeskus Oy, which offers support and IT services for businesses. The goal of this thesis was to design and implement a system for saving the data of support service customers, their services and documentations of their IT infrastructure. The customer management system includes ticketing system and working hour recording system. The goal of this system was to simplify customer management and to ease the jobs of the system administrators who maintain the IT environments.

The project was designed and implemented as an ASP.NET Core based intranet web service. The web service and its database were hosted on a virtual server in the data center of Savon Tietokeskus.

The project was split into three sections: the customer management system, the ticketing system and the IT infrastructure documentation. The required data for the system to be productive was planned. After planning and collecting the required data, the customer management system was designed based on the previous data. The system contains customers' data, their contracts and the services Savon Tietokeskus offers them. The ticketing system was designed and implemented on top of the CRM. Support service customers can send tickets to management. The received tickets will be redirected to technicians by management.

The documentation method of the support service customers' environments of Savon Tietokeskus was redesigned. The environments were documented in spreadsheet-like forms for easier insertion to a relational database. The contents of the database can be viewed and updated via the web application.

As a result of this thesis Savon Tietokeskus received a full customer relationship management system to ease their workload and improve their revenue from support service customers.

Keywords
Customer relationship management, ASP.NET Core, MVC web-application

SISÄLTÖ

LYHENTEET

LYHENTEET

ASP.NET	Active Server Pages .NET on avoimen lähdekoodin web-palvelin ohjelmistokehys, jonka päälle järjestelmä rakennettiin
CSV	Comma-separated values eli pilkuin erotetut arvot on tiedostomuoto, jolla voidaan siirtää taulukkomuotoista tietoa.
PDF	Portable Document Format on avoin sähköisten dokumenttien standarditiedostomuoto. PDF-tiedostoja voi avata esimerkiksi Foxit Reader-ohjelmalla.
O/RM	Object-relational mapping on rajapinta objektien ominaisuuksien sitomiseen suoraan tietokannan tauluihin.
MVC	Model-view-controller -ohjelmistoarkkitehtuurityyli.
DI	Dependency injection on tekniikka, jossa tarvittavat riippuvuudet objektiin syötetään ulkoisesti, esimerkiksi konstruktorin kautta.
HTML	Hypertext Markup Language. Avoimesti standardoitu merkintäkieli.
AJAX	JavaScript pohjainen rajapinta asynkroonisesti suoritettavien pyyntöjen lähettämiseen ja vastaanottamiseen.
API	Application programming interface eli ohjelmointirajapinta.
MSSQL	Microsoft SQL Server. Järjestelmässä käytetty relationaalinen tietokannan hallintajärjestelmä.
LINQ	Language Integrated Query on rajapinta listaan tehtävien kyselyjen ohjelmointiin.
IIS	Internet Information Services. Microsoftin palvelinohjelmisto kokonaisuus web-sovellusten hostaamiseen.
SSL	Secure Sockets Layer on järjestelmässä käytetty salausprotokolla, jolla suojataan järjestelmän tietoliikenne.
VPN	Virtual Private Network mahdollistaa kahden aliverkon yhdistämisen julkisen verkon yli.

1 JOHDANTO

Opinnäytetyön tilasi Savon Tietokeskus Oy, joka tarjoaa tietotekniikan ylläpitopalveluja yrityksille. Opinnäytetyön tavoitteena oli kehittää järjestelmä, johon tallennetaan yrityksen tarjoamat palvelut tukiasiakkaille ja näiden tietoteknisten ympäristöjen dokumentaatiot. Järjestelmän tavoite on yksinkertaistaa asiakkaiden hallintaa ja helpottaa ympäristöjä ylläpitävien asentajien työntekoa.

Asiakkuuksien hallintajärjestelmän rinnalle suunniteltiin ja toteutettiin tiketöintijärjestelmä ja työkalu työajankirjaukseen. Tukiasiakkaiden alle voidaan kirjata tikettejä, joille yrityksen järjestelmäasiantuntijat pystyvät kirjaamaan töitä. Järjestelmän tavoitteena oli yhtenäistää tuen työkaluja ja tuoda ne saman ohjelmiston alle. Tämä mahdollistaa laajemman tietomäärän keräyksen, jonka pohjalta yrityksen hallinnolle voidaan luoda raportteja esimerkiksi tikettien vasteajoista, tehdyistä töistä sekä tarjotuista palveluista.

Projekti jaettiin asiakkuuksien hallintaan, tiketöintijärjestelmään ja ympäristöjen dokumentoinnin suunnitteluun. Projektin ensimmäisessä vaiheessa suunniteltiin miten asiakkaat listataan ja mitä dataa heistä halutaan tallentaa ja raportoida. Tämän päälle rakennettiin tiketöintijärjestelmä ylläpidon työkaluksi. Asiakkaat pystyvät lähettämään järjestelmään tukipyyntöjä, jotka työnjohto voi jakaa Savon Tietokeskus Oy:n järjestelmäasiantuntijoille. Asiantuntijoiden avuksi järjestelmään tallennetaan myös asiakkaiden tietojärjestelmien dokumentaatio.

Työ toteutettiin intranetissä ylläpidettävänä ASP.NET Core 2.1 web-sovelluksena. C#-pohjaisen intranettiin kuuluu virtuaalinen WWW- sekä tietokantapalvelin, jolla web-sovellus ja tietokanta sijaitsevat. Palvelin on Savon Tietokeskus Oy:n omassa konesalissa VMWare-virtualisointialustalla.

2 .NET CORE -FRAMEWORK

ASP.NET Core on uudelleensuunniteltu ASP.NET 4.x -sovelluskehys, joka tarjoaa modulaarisen ja järjestelmäriippumattoman ohjelmistoalustan web-sovelluksille. C#-pohjaisia ASP.NET Core -sovelluksia voidaan kehittää Windows, macOS ja Linux -käyttöjärjestelmillä esimerkiksi Visual Studio Codella, Emacsilla tai Sublimellä. Parhaan kehityskokemuksen saa Windows-käyttöjärjestelmällä ja Visual Studiolla, joka tarjoaa IntelliSensen eli älykkään koodin täydennyksen lisäksi täydet debuggausmahdollisuudet.

Avoimen lähdekoodin ASP.NET Core -sovelluskehys hyödyntää MVC-arkkitehtuuria, joka erottelee sovelluksen malleihin (models), näyttöihin (views) sekä kontrollereihin (controller). MVC-malli helpottaa monimutkaisten ohjelmistojen kehittämistä, sillä on helpompi testata ja kehittää luokkia, joilla on vain yksi tehtävä. ASP.NET Core MVC tukee myös Razor-merkintäsyntaksin käyttöä, joka mahdollistaa C#-ohjelmointikielen upottamista HTML-merkintäkielen sekaan.

ASP.NET Core web-sovellukset voidaan luoda suoraan komentoriviltä ja ne tulevat oletuksena Bootstrap tyyli- ja JavaScript -kirjastojen kanssa. Bootstrap kirjaston avulla voidaan kehittää helppokäyttöisiä ja responsiivisia web-sovelluksia myös tableteille ja mobiililaitteille. Visual Studiolla kehitettäessä web-sovelluksen selainpuolen ohjelmointia helpottaa Browser Link, joka mahdollistaa sovelluksen testauksen usealla selaimella samaan aikaan.

2.1 Miksi ASP.Net Core?

Asiakkuuksienhallintajärjestelmää suunnitellessa tuli ottaa huomioon, että kehitetäänkö sovellus erillisenä sovelluksena eri käyttöjärjestelmille vai web-sovelluksena. Projektin aloituspalaverissa tultiin siihen tulokseen, että .NET Core -pohjainen web-sovellus toimii skaalautuvuutensa ja käyttöjärjestelmäriippumattomuutensa takia parhaiten. Web-sovelluksen kehittämisen etuna toimii myös se, että mobiililaitteille ei tarvitse kehittää omia sovelluksia, vaan Bootstrap-kirjastolla saadaan kehitettyä kaikille laitteille responsiivisia näyttöjä.

Järjestelmässä tullaan käsittelemään asiakkaiden arkaluontoista tietoa, joten tietoturva on tärkeässä roolissa. Web-sovellus rajattiin Data Group Kuopion omaan sisäverkkoon, joka lisää sovelluksen tietoturvaa. ASP.NET Core -sovellukset on mahdollista ylläpitää IIS, Nginx, Apache, Docker tai itsehostattuna palveluna. Projektiin valittiin web-palvelinohjelmistoksi IIS sen konfigurointimahdollisuuksien vuoksi.

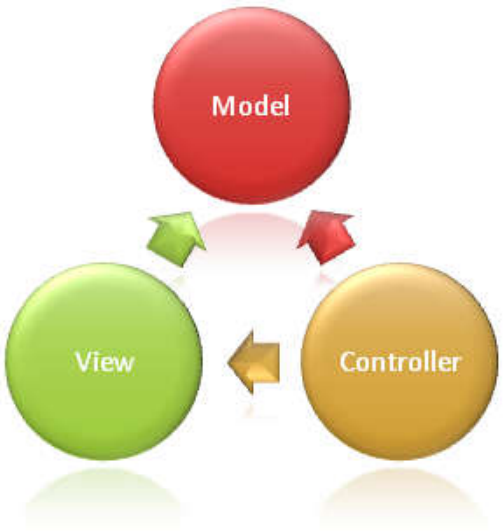
Alustan valintaan vaikutti myös .NET Core -sovellusten modulaarinen luonne. Tämän mahdollistaa Dependency injection (DI), eli mahdollisuus ”injektoida” sovellukseen riippuvuuksia muihin kirjastoihin tai luokkiin. Riippuvuudet ovat luokkia, joita muut luokat tarvitsevat toimiakseen. Näitä ovat esimerkiksi tietokanta rajapinta EntityFramework Core ja MVC. Riippuvuudet määritellään web-sovelluksen Startup-metodissa ja dependency injekction helpottaa riippuvuuksien käyttämistä ja testaamista. Ohjelmiston kehityksessä käytettiin seuraavia riippuvuuksia:

1. EntityFramework Core
    - Toimii rajapintana tietokannan ja sovelluksen välillä.
2. PDF-tools
    - Mahdollistaa HTML-merkintäkielen muuttamisen PDF-tiedostoiksi.
3. Identity
    - Mahdollistaa sovelluksen autentikoinnin.
4. MVC
    - Model-View-Controller -malli.

2.2 MVC-Malli

MVC-mallissa mallit, näytöt ja kontrollerit ovat vuorovaikutuksessa toistensa kanssa (kuva 1). Kontrollerit tarjoilevat käyttäjälle näyttöjä. Kun käyttäjä tarkastelee tai muokkaa malleja, palautetaan ne takaisin kontrollerille, joka käsittelee malleja ja tekee muutokset tietokantaan. Mallit on yhdistetty tietokantaan ja kontrollerit tekevät mallien muutosten perusteella muutokset myös tietokantaan. MVC-malli helpottaa ohjelmointia, debuggausta sekä testaamista, sillä jokaisella mallilla, näytöllä tai kontrollerilla on yksi tehtävä. Tämä noudattaa SRP:tä (Single Responsibility Principle), jonka mukaan jokaisella luokalla tulisi olla vain yksi syy tehdä muutoksia dataan (Martin 2002).

Kuva 1: MVC-malli



2.2.1 Mallit

Relationaalisen tietokannan tauluja käsitellään objekteina, eli C#-luokkina ASP.NET Core:ssa. Taulukon sarakkeet liitetään luokkien ominaisuuksiin (property) EntityFramework Core:n tietokantakontekstin avulla. Objektien ominaisuuksille voidaan antaa validaatio-attribuutteja, jotka määrittelevät esimerkiksi merkkijonon pituuden tai onko ominaisuus pakollinen arvo tietokannassa (not nullable). Näin näkymästä syötettävän datan validointi saadaan määritettyä jo objektia luodessa, eikä sitä tarvitse määritellä jokaisessa näytössä erikseen.

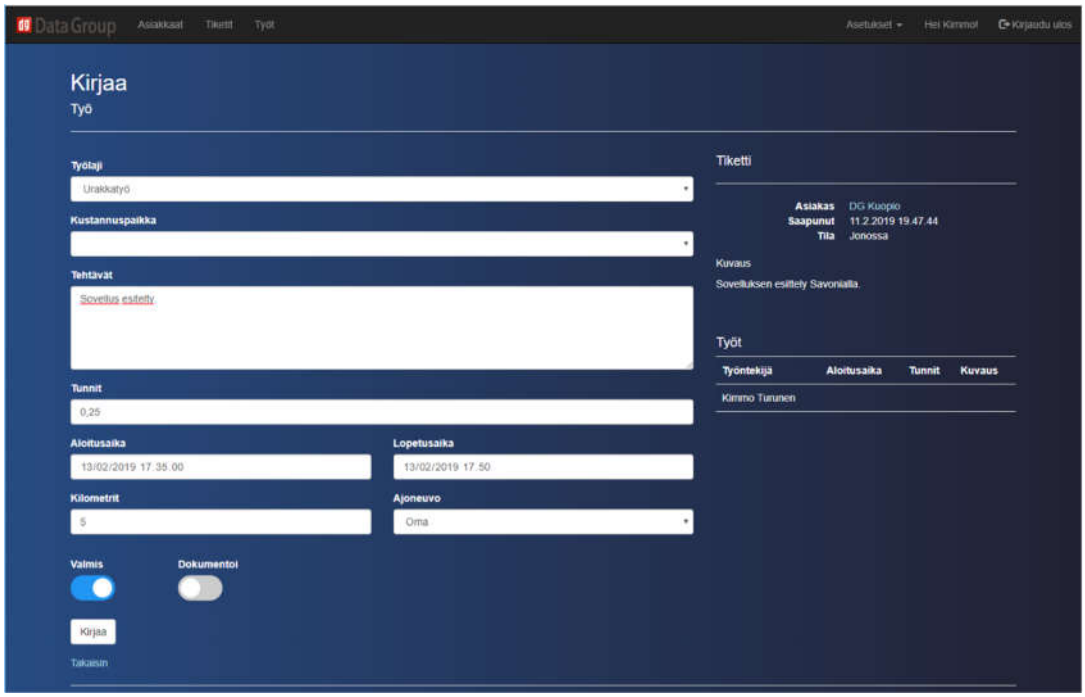
Malleille voidaan lisätä myös navigaatio-ominaisuuksia. Jos tietokanta sisältää yhdestä yhteen, yhdestä moneen tai monesta moneen yhteyksiä taulujen välillä, niin navigaatio-ominaisuuksilla pystytään kutsumaan luokkiin liittyviä muita luokkia.

2.2.2 Näytöt

MVC-mallin applikaatioissa näytöt vastaavat datan näyttämisestä sekä toimii rajapintana käyttäjän ja datan vuorovaikutuksessa. Näytöt ovat HTML-pohjia, joihin voidaan upottaa Razor-merkintäkieltä. (Smith ja Latham 2017). ASP.NET Coressa näytöt ovat .cshtml tiedostopäätteisiä, sillä HTML:n lisäksi niihin voidaan upottaa C#-koodia. Jokaiseen näyttöön voidaan liittää yksi malli, jota pystytään kutsumaan Model-nimisellä muuttujalla.

ASP.NET Coressa hyödynnetään pohjia (layouts) sivujen näyttämiseen. Pohjaan voidaan sijoittaa näytön elementit, joita näytetään useammalla sivulla kuten navigointipalkki tai alaviite. Tämä helpottaa merkintäkielen ylläpitoa, sillä muutokset täytyy tehdä vain yhteen tiedostoon, eikä jokaiseen sivuun erikseen. Itse näyttöihin pystytään upottamaan kokonaisia näyttöjä, näyttö-komponentteja (View components), tai osittaisia näyttöjä (Partial views) malleineen (kuva 2). Tämä helpottaa koodin uudelleenkäyttöä ja yksinkertaistaa sivuston ylläpitoa.

Kuva 2: Tiketti on haettu näyttöön partial viewinä



2.2.3 Kontrollerit

MVC-mallin web-sovelluksessa kontrolleri vastaa pyyntöjen vastaanottamisesta, mallin alustamisesta ja ohjelmointilogiikan toteuttamisesta mallille. Kontrollerit määrittelevät joukon toimintatapoja (action methods), jotka ottavat vastaan pyyntöjä ja palauttavat niiden perusteella joko näytön tai API-kutsun vastauksen. (Smith ja Addie 2017). Kontrollerit voivat palauttaa:

- HTTP status-koodin
- Uudelleenohjauksen
- Näytön
- Tiedoston
- JSON-merkkijonon

ASP.NET Core hyödyntää reititystä (routing middleware) yhdistämään saapuvien pyyntöjen URL:t kontrollerien toimintatapoihin.

2.3 Autentikointi ja identiteetti

Järjestelmässä käsitellään yritysasiakkaiden tietoja, jotka eivät saa näkyä ulkopuolisille henkilöille. Siitä huolimatta, että sovellus on Savon Tietokeskus Oy:n sisäverkossa, täytyi sovellukseen rakentaa käyttäjien tunnistus. ASP.NET Coressa on identity-niminen lisäosa, joka tuo kirjautumistoiminnot web-sovellukseen. Identiteettiä voidaan hyödyntää tietokannassa säilytettävillä tunnuksilla, tai ulkoisilla tileillä, kuten Facebook, Google tai Microsoft tilileillä. Identiteetin avulla saadaan estettyä pääsy kaikkiin sovelluksen osiin, paitsi kirjautumisikkunaan.

Kirjautuneita käyttäjiä sovelluksessa on kahdentasoisia: ylläpitäjiä sekä asentajia. Ylläpitäjät eli työnjohto sekä yrityksen johtohenkilöt voivat lisätä, muokata ja lukea kaikkia sovelluksessa olevia tietoja, määrätä tikettejä toisille käyttäjille sekä tulostaa raportteja. Asentajat pystyvät kirjaamaan tikettejä ja töitä. ASP.NET Core web-sovelluksissa kontrollereille tai kontrollerien funktioille voidaan määrittää, tarvitseeko käyttäjän olla tunnistettu sekä tarvitseeko hänen kuulua tiettyyn ryhmään. Sen lisäksi, että palvelinpuolen koodissa on otettu käyttäjien oikeudet huomioon, asentaja-tason käyttäjiltä on poistettu ne elementit näkyvistä, joita he eivät voi käyttää.

3 TIETOKANTA

Web-sovellus tarjoaa rajapinnan loppukäyttäjän ja tietokannassa sijaitsevan tiedon välille. Asiakkaiden varsinainen tieto tallennetaan tietokantaan. Web-sovellus perustuu suurimmaksi osaksi CRUD-operaatioihin ja sen hyödyntämä tieto on taulukkomuotoista, joten päädyttiin relationaaliseen tietokantaan. Visual Studion kehitysympäristöstä löytyy sisäänrakennettuna MSSQL-tietokantatyökalut, joka helpotti MSSQL-tietokannan liittämistä sovellukseen myös tuotantoympäristössä. ASP.NET Core:n voidaan lisätä Nuget-pakettina Entity Framework Core .NET -kirjasto, joka toimii rajapintana sovelluksen ja tietokannan välillä.

3.1 Entity Framework Core

Entity Framework Core toimii ORM:nä (Object-relational mapper), eli se sitoo tietokannan taulut .NET luokkiin. EF Coren ominaisuuksia ovat Code-First sekä migraatiot. Code-First ominaisuuden avulla kehittäjät voivat kirjoittaa .NET-luokat ja päivittää tai luoda tietokannan taulujen pohjalta. Migraatioiden avulla tietokannan tauluja voidaan muokata vaikuttamatta tauluissa oleviin riveihin tai muihin tietoihin. Migraatiot tarjoavat komentorivi komennot (CLI tools) mm. seuraaviin tilanteisiin:

1. Luo migraatio – Generoi koodin, jolla tietokantaa päivitetään inkrementaalisesti
2. Päivitä tietokanta – Päivittää tietokannan migraatioiden perusteella
3. Poista migraatio – Poistaa halutun migraation
4. Peruuta migraatio – Kumoaa tietokantaan tehdyt muutokset

Entity Framework Core toimii tietokanta-kontekstin avulla. Tietokantakontekstin OnModelCreating funktiossa määritellään entiteetit, jotka vastaavat tietokannan tauluja. Entiteetille määritellään tietokannan vastaavat sarakkeet sekä pää- ja vierasavaimet ja mahdollisesti myös taulun nimi, johon viitataan. Entiteetille voidaan määritellä myös viite-eheydet, eli mitä yhdistetyille tiedoille tapahtuu, jos taulujen tiedot poistetaan.

Kontrollerilla määriteltävän tietokantakontekstin kautta voidaan hakea tietoa tietokannasta. Sovelluksessa haetaan tiedot ”eager loading” -menetelmällä, eli kaikki haettava tieto ladataan tietokannasta yhdellä kerralla. Esimerkiksi tietyn tiketin tietoja hakiessa sovellus näyttää asiakkaan nimen, tikettiin liittyvät työt sekä tehtyjen töiden työlajit. Kaikki tikettiin liittyvä tieto näytetään samaan aikaan, joten on tehokkaampaa hakea tarvittavat tiedot yhdellä tietokantakyselyllä. Eager loading -menetelmän vastakohta on ”lazy loading”, jossa tarvittavat tiedot haetaan vasta silloin, kun navigaatio-ominaisuuksia tarvitaan. ”Explicit loading” on lazy loading -menetelmää vastaava, mutta siinä navigaatio-ominaisuudet haetaan manuaalisesti Load() -komennolla. (Miller ja kollegat 2016).

EF Core hyödyntää Language-Integrated Queryjä eli LINQ-kyselyjä tiedon hakemiseen, rajaamiseen sekä järjestämiseen. LINQ on joukko menetelmiä, jotka perustuvat kyselyjen integroimiseen suoraan C#-kieleen. (Wenzel, Wagner, Latham, Hoag 2017). EF Core tukee suurinta osaa tietokantamoottoreista. Projektissa käytettiin MSSQL Server -tietokantaa.

4 ASIAKKUUDENHALLINTA ELI CRM

Savon Tietokeskus Oy:llä on n. 100 ylläpidon tukiasiakasta, joilla on ylläpidossa 5 000 työasemaa ja yli 300 palvelinta. Asiakkaiden ja heidän laitteiden lisäksi Savon Tietokeskus Oy tarjoaa konesalipalveluita pilvipalveluista virtualisointiin ja sähköpostipalveluihin. Asiakkaiden laitteille, tietoverkoille ja heille tarjotuille palveluille tarvittiin järjestelmä, johon tallennettaisiin sekä tuen että hallinnon tarvitsemat tiedot asiakkaista.

Asiakkaiden sopimuksista ja niihin kuuluvista palveluista haluttiin yksinkertainen näkymä, josta saa mahdollisimman paljon tietoa yrityksestä yhdellä vilkaisulla. Tämä helpottaa myös yrityksen kirjanpitoa, sillä kaikki tieto asiakasyritykselle tarjotuista palveluista on koostettu samaan paikkaan. Tiedon pohjalta saadaan luotua asiakaskohtaisia raportteja heidän laitteistaan, palveluistaan tai heidän yritykselleen tehdyistä töistä ja vasteajoista. Raporttien perusteella Savon Tietokeskus Oy pystyy tarjoamaan ennakoivaa tukea ja palvelua esimerkiksi seuraamalla yrityksen laitteiden takuun loppumispäivämääriä.

4.1 Työn määrittely

Työ suunniteltiin kartoittamalla mitä tietoa järjestelmään tarvitaan ja mitä sillä pystytään tekemään. Nykyisistä järjestelmistä saatiin listaus yritysasiakkaiden sopimuksista, heille tarjotuista palveluista, tietoverkoista, laitteista ja heille tehdyistä töistä. Opinnäytetyössä käytettiin relaatiotietokantaa, joten tieto täytyi muokata taulukkomuotoon, jota voi hyödyntää tietokannassa. Loppukäyttäjille eli yrityksen hallinnolle, työnjohdolle ja järjestelmäasiantuntijoille näytettävät näytöt ja toiminnallisuudet

suunniteltiin vastaavasti kunkin käyttäjäryhmän kanssa. Työ toteutettiin web-sovelluksena, joten selainpuoli toteutettiin käyttämällä JavaScript- ja CSS-kirjastoja jQuery ja Bootstrap. Tämä mahdollistaa responsiivisen käyttäjäkokemuksen myös tableteilla ja mobiililaitteilla.

Nykyisestä yrityksen työajankirjauksesta saatiin listaus asiakkaista CSV-muodossa, jota muokattiin Excel-taulukkolaskentaohjelmalla järjestelmään sopivaksi. Asiakkaiden tietoihin täytettiin mm. yhteystietoja, lyhyt järjestelmäkuvaus ja vastaavat tukihenkilöt. Asiakkaiden ja Savontietokeskus Oy:n välisistä sopimuksista kirjattiin päivämäärät, kuvaukset ja sopimuskausi. Näin sopimuksien tietoihin voidaan tehdä tietokantakyselyjä esimerkiksi päivämäärillä. Savon Tietokeskus Oy:n hallintoon kuuluville henkilöille näytetään myös linkki suoraan sopimukseen PDF-muodossa. Sopimukset koskevat palveluja, joita Savon Tietokeskus Oy tarjoaa asiakkailleen. Näitä ovat esimerkiksi virtualisointipalvelut ja sähköpostipalvelut. Yrityksen tarjoamista palveluista kerättiin listaus ja ne muutettiin taulukkomuotoon. Sopimuksien alle voidaan kirjata mitä palveluita ne sisältävät ja kuinka monta niitä on. Sopimuksien ja niiden palveluiden kirjaaminen helpottaa Savon Tietokeskus Oy:n kirjanpitoa heidän tarjoamistaan palveluista ja niiden lukumääristä.

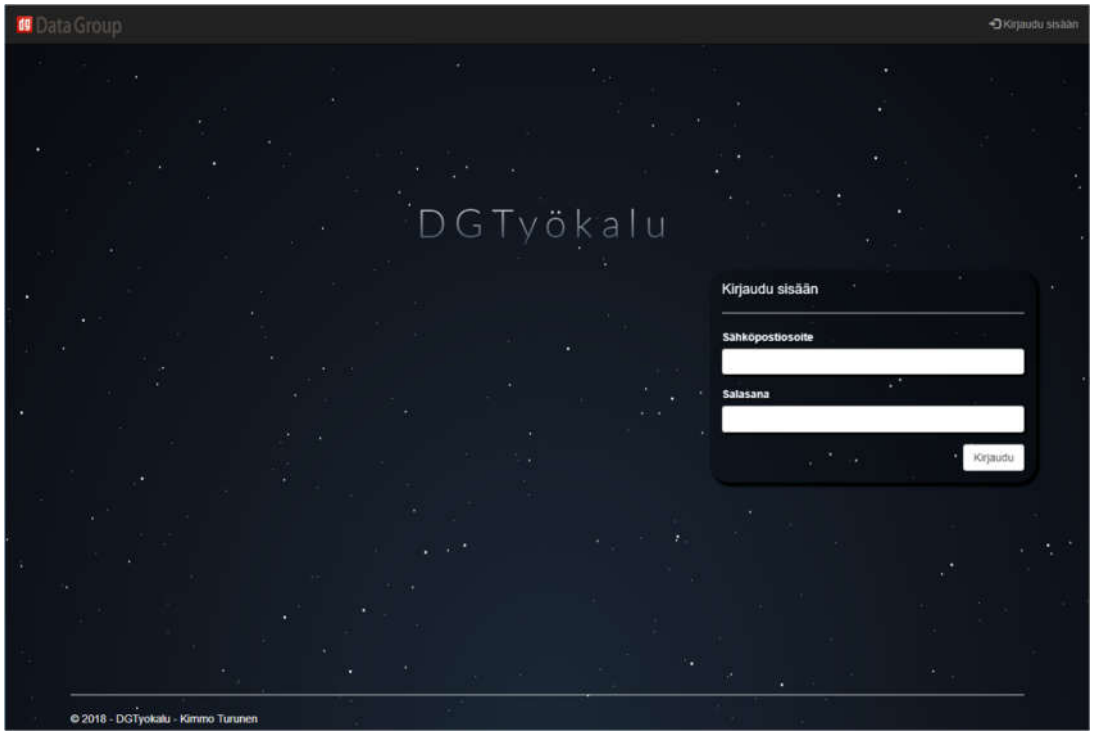
Sovellukseen pystytään kirjaamaan muutos- ja tukipyyntöjä asiakkaiden alle. Tämä yksinkertaistaa työnkulkua, sillä jokaisesta ongelmatilanteesta kirjataan tiketti, jonka alle työntekijät voivat kirjata tekemiään töitä. Tämä helpottaa suurempien projektien työnkirjausta ja selkeyttää hallinnon ja laskutuksen kirjanpitoa, koska he näkevät suoraan mihin tukipyyntöön kirjattu työ liittyy. Tiketti-työ -suhteen avulla myös asiakkalle saadaan tarkempia raportteja heiltä tulevista tukipyynnöistä ja niiden vaatimista toimenpiteistä.

4.2 Työn vaiheet

Projektityön aloituspalaverissa mitä ominaisuuksia he haluavat järjestelmältä. Samalla kartoitettiin mitä tietoa järjestelmässä voidaan hyödyntää. Järjestelmä päätettiin toteuttaa ASP.NET Core 2.0 web-sovelluksena, jota ylläpidetään virtuaalipalvelimella Savon Tietokeskus Oy:n alustapalvelimella. ASP.NET Core sovelluskehykseen päädyttiin sen avoimen lähdekoodin ja entuudestaan tutun ohjelmointikielen C# takia.

Web-sovelluksen ohjelmointi toteutettiin Microsoft Visual Studio 2017:lla, joka tarjoaa selkeän ohjelmankehitysympäristön .NET-pohjaisille sovelluksille. Tyhjään ASP.NET Core 2.0 projektiin lisättiin NuGet-kirjasto Entity Framework Core. Kirjaston ja Visual Studion Code First -menetelmällä järjestelmään saadaan lisättyä tietokantayhteys ja tietokannan taulut. Järjestelmään lisättiin myös Razor Class -kirjasto ASP.NET Core Identity. Kirjaston avulla projektiin voidaan lisätä käyttäjä-objekti, -kontrolleri ja näiden näytöt. Tämä lisää järjestelmään näytöt käyttäjän lisäykselle tietokantaan ja mahdollisuuden kirjautua sisään (kuva 3) ja ulos. Käyttäjille voidaan valita käyttäjäryhmä joille voidaan näyttää erilaisia sivuja ja toimintoja. Käyttäjäryhmiä lisättiin kaksi, järjestelmänvalvojat ja asentajat.

Kuva 3: Lopullinen kirjautumissivu



Dependency Injection -menetelmällä järjestelmään saatiin lisättyä tietokannan ja identiteetin lisäksi myös evästeiden käytön asetukset, lokalisaatio-asetukset, PDF-kirjasto ja HTTPS. Lisättyjä ”palveluita” kutsutaan sovelluksen Startup.cs tiedoston ConfigureServices -metodissa.

Järjestelmään lisättiin asiakas-malli (kuva 4). Mallin pohjalta projektiin sisäänkirjautuneet käyttäjät voivat lisätä ja näyttää asiakkaita. Asiakkaiden alle voidaan kirjoittaa tietoja yrityksestä ja lyhyt järjestelmäkuvaus. Asiakas-mallia suunnitellessa otettiin huomioon tietojen tahaton poisto. Tietokannan asiakas-tauluun tehtiin totuusarvon sisältävä sarake, joka kertoo onko asiakas poistettu vai ei. Asiakas-malliin lisättiin myös aikaleimat asiakkaan luomiselle sekä muokkausajalle.

Asiakas-mallia hallitsee asiakas-kontrolleri. Asiakas-kontrollerille on määritetty koko luokalle näkyvät muuttujat tietokantakonteksti ja käyttäjähallinta, joille määritellään arvot konstruktorilla.. Tietokantakontekstin avulla voi hakea tietoa tietokannasta Entity Framework Core rajapinnan kautta. Käyttäjänhallinta-muuttujan avulla voidaan hakea sovellukseen kirjautuneiden käyttäjien tietoja ja määritellä esimerkiksi oikeuksia kontrollerin eri metodeille. Asiakas-kontrolleri ottaa ja palauttaa pyyntöjä vain ”ylläpitäjät” ja ”asentajat” ryhmiin kuuluville käyttäjille.

Asiakas-kontrollerista löytyy useita metodeja, joilla asiakkaita koskevaa tietoa voidaan muokata. Index-metodi palauttaa listan kaikista asiakkaista index.cshtml-nimiselle näytölle, jossa listataan kaikki asiakkaat. Metodi ottaa argumenttina muuttujan ”searchString”, jonka perusteella voidaan rajata mitkä asiakkaat haetaan. Index-näytöstä löytyy hakukenttä ja -painike, joilla parametrit voidaan lähettää kontrollerin metodille. Details-metodi ottaa vastaan asiakkaan uniikin id-numeron ja näyttää tämän asiakkaan tiedot, dokumentaation, kustannuspaikat ja työlajit omassa details.cshtml-näkymässä.

Edit-metodeja asiakas-kontrollerista löytyy kaksi erilaista. Ensimmäinen ottaa vastaan id-numeron, kuuntelee http-get pyyntöjä ja palauttaa näkymän, jossa asiakkaan tietoja voi muokata. Toinen Edit-metodi ottaa vastaan asiakkaan id-numeron, asiakas-mallin ja kuuntelee http-post pyyntöjä. Tämä metodi ottaa vastaan muokkaus-näkymästä tulevan muokatun mallin ja tallentaa sekä muuttuneet tiedot että muokkausajan tietokantaan tai palauttaa virheen jos tallennus ei onnistu. Delete-metodeja on myös kaksi kappaletta. Ensimmäinen palauttaa poistettavan asiakkaan tiedot ja kysyy halutaanko tiedot varmasti poistaa. Toinen delete-metodi merkkaa tietokantaan, että asiakas on poistettu, mutta ei poista sitä tietokannasta.

Lisäämällä palvelu- ja sopimus-mallit järjestelmään voidaan lisätä yrityksen tarjoamat palvelut. Asiakkaiden voidaan lisätä sopimuksia, joilla on Savon Tietokeskus Oy:n tarjoamia palveluita. Palvelu- ja sopimus-kontrollereilta löytyy edellä mainitut index, details, create, edit ja delete -metodit ja näytöt. Sopimuksien ja palveluiden välillä on monesta-moneen suhde, joten sopimusta luodessa create-metodi ottaa vastaan myös listan valituista palveluista. Sopimuksia ja palveluita yhdistää sopimuksen-palvelut taulu, johon merkitään palvelut, jotka kuuluvat sopimuksille.

Dokumentaatio-mallien avulla järjestelmään saadaan mahdollisuus lisätä ja muokata ympäristöön liittyviä malleja, kuten tunnukset, laitteet tai tietoverkot. Asiakkaiden ympäristöjen dokumentaatio ja heille tarjotut palvelut lisättiin näkymään asiakkaan tietojen kanssa samaan näkymään, jolloin asiakkaasta saadaan paljon tietoa yhdellä vilkaisulla (kuva 5). Dokumentaatio-kontrollerista löytyy edellä mainitut index, details, create, edit ja delete -metodit ja näytöt. Kontrollerien näytöt on upotettu osittaisnäyttöinä Asiakas/Details -sivulle ja niiden lähettämät tiedot lähetetään AJAX-kutsuilla kontrollerille. Näin käyttöliittymästä saadaan sujuvampi, sillä sivua ei tarvitse päivittää jokaista kutsua tehdessä.

Asiakkaiden ympäristöjen dokumentointia varten suunniteltiin taulukkolaskenta-tiedosto, johon voisi dokumentoida kaikenkokoisia ympäristöjä. Asiakkaiden dokumentaatiot siirrettiin tiedostoon, josta ne ladattiin järjestelmän tietokantaan.

Kuva 4: Asiakas-malli ja navigaatio-ominaisuudet

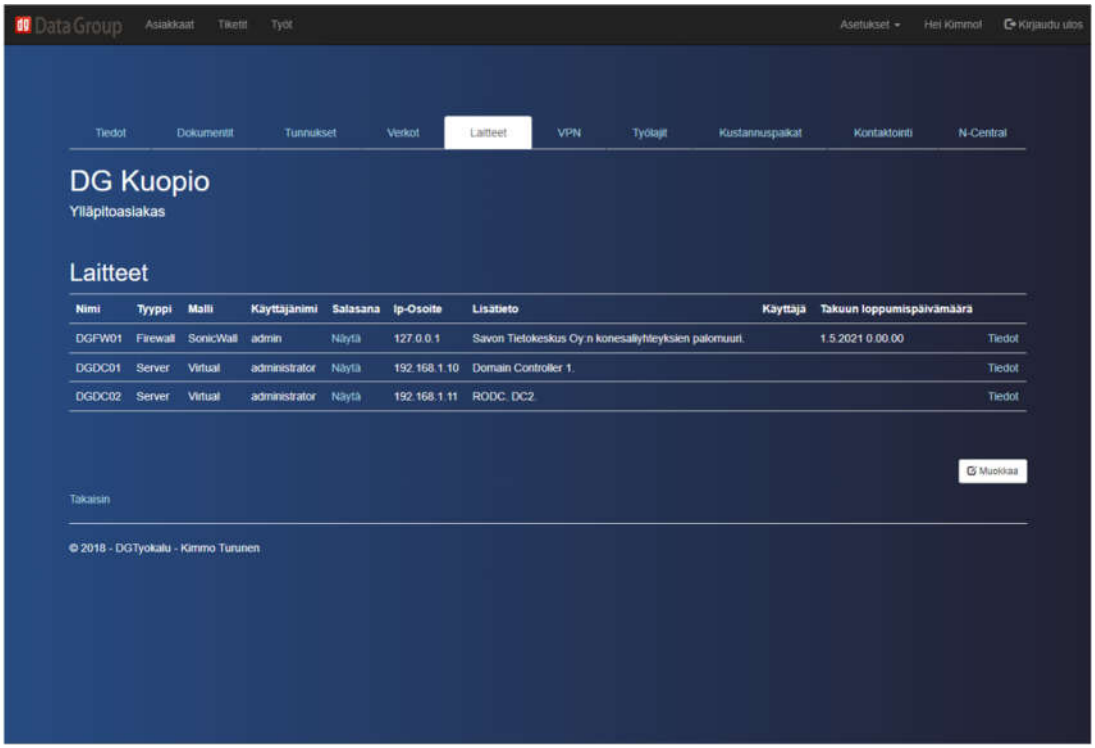
```
public partial class Asiakas
    {
        public Asiakas()
        {
            LuotuAika = DateTime.Now;
            MuokattuAika = DateTime.Now;
            OnPoistettu = false;
        }

        [Key]
        public int AsiakasId { get; set; }
        [Display(Name = "Asiakasnumero")]
        public string AsNro { get; set; }

        [Required]
        public string Nimi { get; set; }
        [StringLength(10, ErrorMessage = "Y-tunnus ei voi olla pidempi kuin 10
merkkiä.")]
        [Display(Name = "Y-tunnus")]
        public string Ytunnus { get; set; }
        [Display(Name = "Yhteyshenkilö")]
        public string Yhteyshenkilo { get; set; }
        [Display(Name = "Tekninen Yhteyshenkilö")]
        public string TeknYhteyshenkilo { get; set; }
        public string Osoite { get; set; }
        [Display(Name = "Puhelinnumero")]
        public string Puh { get; set; }
        [Display(Name = "Sähköpostiosoite")]
        public string Email { get; set; }
        [Display(Name = "Info")]
        public string Info { get; set; }
        [Display(Name = "Ylläpitoasiakas")]
        public bool Yllapito { get; set; }
        public bool OnPoistettu { get; set; }
        public DateTime LuotuAika { get; set; }
        public DateTime MuokattuAika { get; set; }
        public Guid ApplicationUserId { get; set; }

        public ICollection<Sopimus> Sopimukset { get; set; }
        public ICollection<Tiketti> Tiketit { get; set; }
        public ICollection<Laite> Laitteet { get; set; }
        public ICollection<Huolto> Huolto { get; set; }
        public ICollection<Tyolaji> Tyolajit { get; set; }
        public ICollection<Verkkojako> Verkkojaot { get; set; }
        public ICollection<Verkko> Verkot { get; set; }
        public ICollection<Sahkoposti> Sahkopostit { get; set; }
        public ICollection<VPN> VPN { get; set; }
        public ICollection<Tunnus> Tunnukset { get; set; }
        public ApplicationUser ApplicationUser { get; set; }
    }
```

Kuva 5: Asiakkaan tiedot



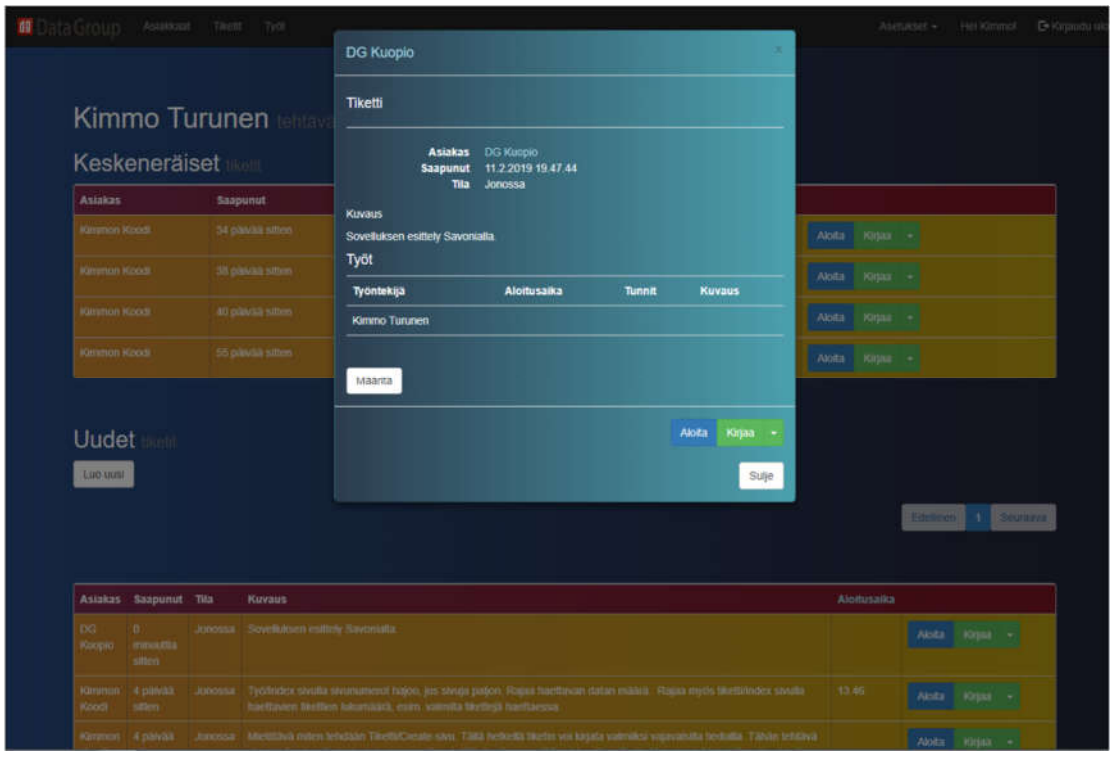
Tiketti-, työlaji- ja työ-malleilla voidaan rakentaa tiketöintijärjestelmä. Järjestelmään voidaan käyttöliittymän kautta lisätä tikettejä ja niitä voidaan listata. Asentajat voivat aloittaa töitä, jolloin tiketin alle luodaan, jonka aloitusajankohta tallennetaan tietokantaan. Asentajat voivat kirjata tiketeille töitä ja merkkaamaan sen valmiiksi. Jokaista työtä kirjattaessa sille tallennetaan järjestelmänvalvojien lisäämä työlaji, joka sisältää tiedon työn tuntiveloituksesta. Tikettijärjestelmän etusivulla (kuva 6) näkyy käyttäjälle ohjatut tiketit sekä käyttäjän keskeneräiset tiketit. Järjestelmänvalvojat pystyvät ohjaamaan tikettejä työntekijöille tikettien etusivulta. Työn ohjaaminen työntekijän alle luo työ-mallin, joka yhdistää tiketin, työntekijän sekä aloitusajankohdan. Tällöin työ tulee näkymään työntekijän omat tiketit -sivulle. Kun työ on saatu valmiiksi, voi työntekijä kirjata sen järjestelmään.

Tiketti-kontrollerista löytyy CRUD-operaatioiden lisäksi metodi ”Mytickets”, jonka avulla työntekijät voivat listata kaikki hänelle määrätyt tiketit. Mytickets-näytössä näkyy myös työntekijän aloittamat tiketit. Mytickets-metodi hyödyntää index.cshtml-näyttöä siten, että sille palautetaan vain työntekijälle kuuluvat tiketit ja työt. Index.cshtml-näyttö ottaa vastaan PaginatedList<Tiketti> muotoisen muuttujan, joka tarjoaa rajapinnan sivunumeroinnille. Näin kaikkia tikettejä ei tarvitse näyttää samalla sivulla, vaan käyttäjälle tarjotaan 20 tikettiä kerrallaan ja mahdollisuus siirtyä sivuilla eteen tai taaksepäin katsomaan seuraavat tai edelliset 20 tikettiä. Tästä syystä mytickets- ja index-metodit ottavat vastaan haku-tekstijonon lisäksi sivunumeron jota käyttäjä haluaa selata, aloitus- ja lopetuspäivämäärät minkä väliltä tikettejä halutaan selata sekä muuttujan tila, jos halutaan selata tietyssä tilassa olevia tikettejä.

Tiketti-kontrollerilla on myös modal-metodi, joka ottaa vastaan tiketin id-numeron ja palauttaa osittaisnäyttönä tiketin tiedot. Kun etusivulla painetaan tikettiä, haetaan metodilta AJAX-kutsulla tiketin tiedot ja näytetään ne esiin tulevalla osittaisnäytöllä (kuva 6). Tämä tekee käyttöliittymästä sujuvamman ja helppokäyttöisemmän, sillä sivua ei tarvitse päivittää jokaisella kerralla kun halutaan tarkastella eri tikettien tietoja. Aukeavasta osittaisnäytöstä löytyy painikkeet ”Aloita” ja ”Lopeta”, jotka viittaavat tiketti-kontrollerin start- ja stop-metodeihin. Nämä metodit vastaavasti merkkaavat tiketin aloitetuksi tai lopetetuksi. Tiketti merkataan aloitetuksi tietylle käyttäjälle siten, että sen alle luodaan tyhjä työ jolla on aloitusaika. Tämä työ yhdistää työntekijän sekä tiketin.

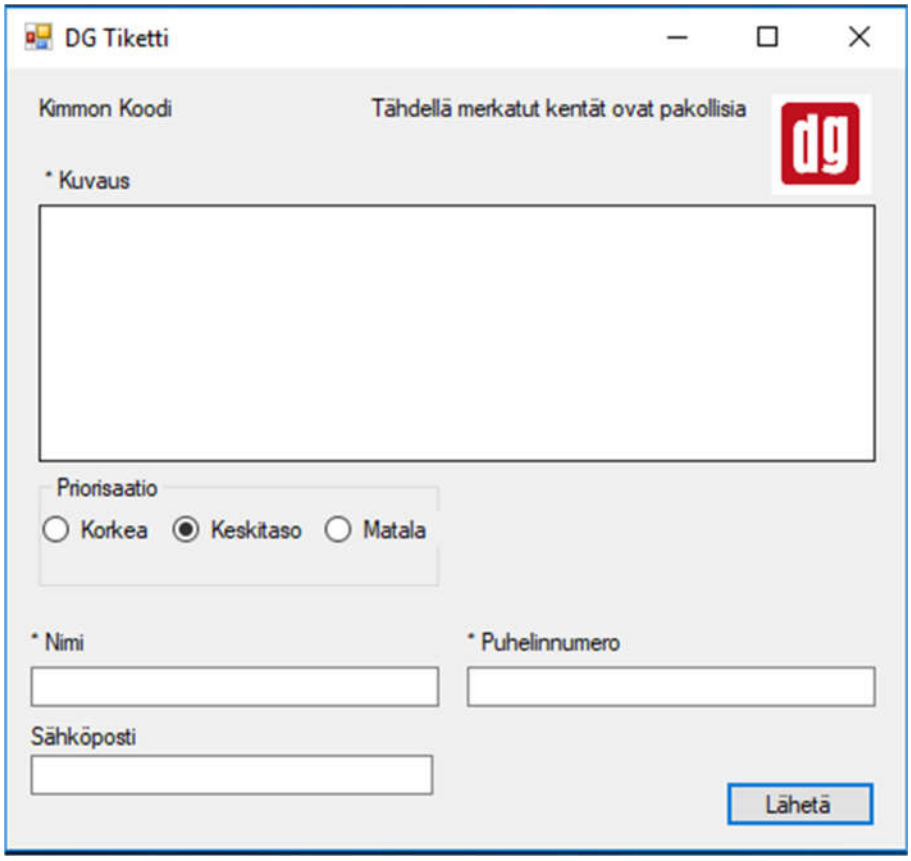
Järjestelmänvalvojat ryhmään kuuluvat käyttäjät pystyvät ohjaamaan tikettejä aukeavan osittaisnäytön kautta toisille työntekijöille. Normaaleilta käyttäjiltä piilotettu ”Määritä” painike avaa näkymän, josta voidaan valita tiketin aloitusaika sekä työntekijä, jolle tiketti halutaan ohjata. Metodin käyttö on rajattu myös kontrollerilta ja palauttaa 403 – kielletty vastauksen jos johonkin muuhun ryhmään kuuluva käyttäjä yrittää lähettää tietoa metodille.

Kuva 6: Tikettijärjestelmän etusivu



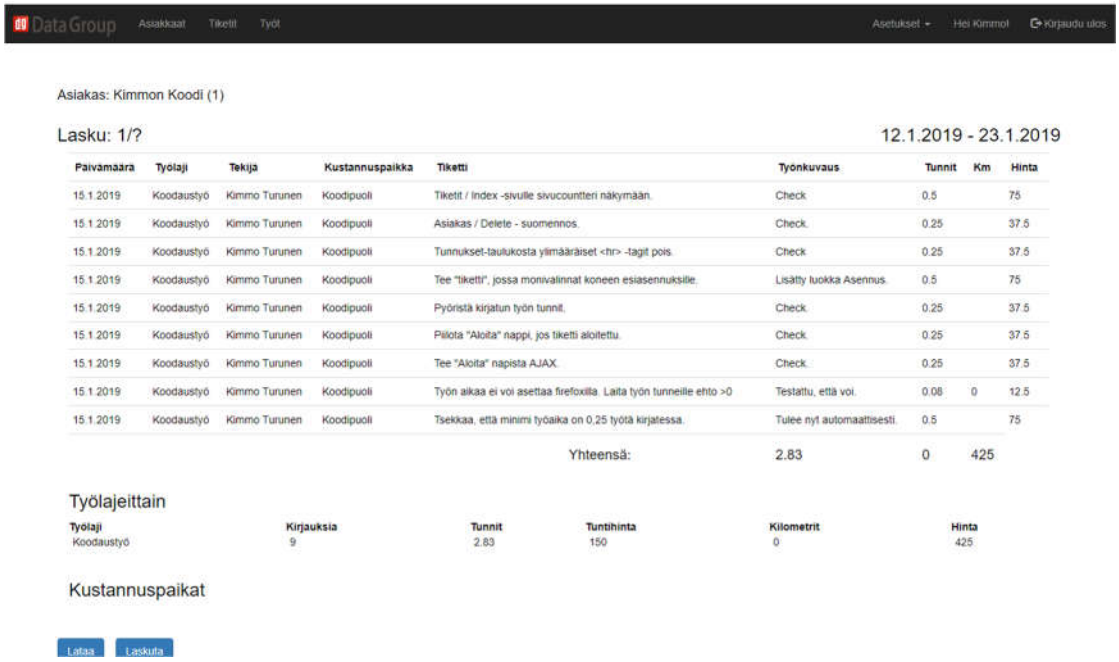
Tikettien järjestelmään syöttämistä varten kehitettiin erillinen C#:lla kehitetty Windows Forms -sovellus (kuva 7). Sovelluksen avulla asiakkaat pystyvät lähettämään tikettejä samalla palvelimella sijaitsevaan julkiseen API:n, joka lisää tiketit tietokantaan.

Kuva 7: Tiketti-sovellus



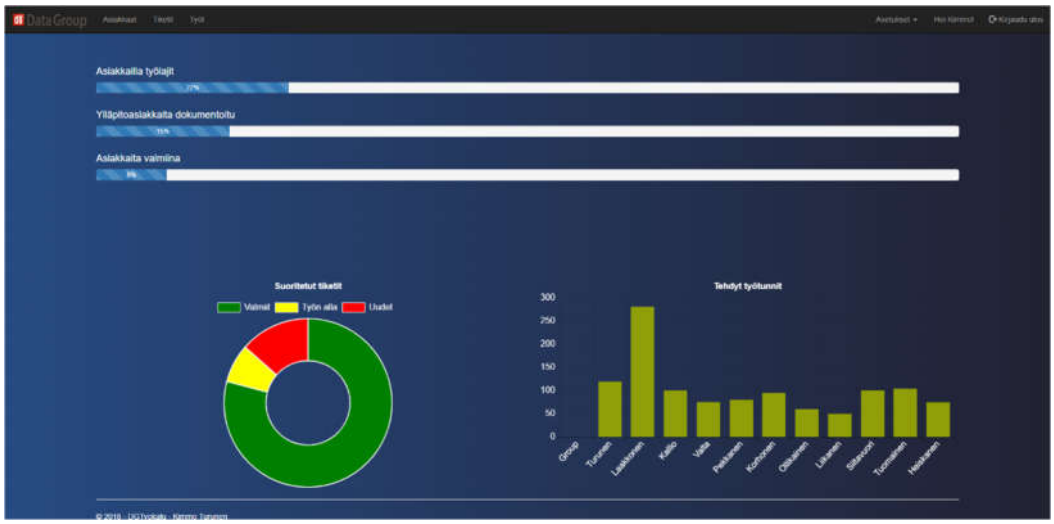
Projektiin lisättiin raportti-kontrolleri, jonka tehtävä on tehdä kyselyjä kirjattuihin töihin. Kontrolleri palauttaa näytölle raportin näyttöä varten suunnitellun mallin, jolla pystytään näyttämään esimerkiksi asiakkaalle tehtyjä töitä lajiteltuna työlajeittain tai kustannuspaikoittain (kuva 8). Raportointiin tehtiin myös mahdollisuus kirjata töitä laskutetuksi ja lähettää luotu raportti suoraan asiakkaan sähköpostiin PDF-muodossa.

Kuva 8: Esimerkkiraportti



Raportti-kontrolleria hyödynnetään myös työnjohdon näkymässä (kuva 9). Kojelauta-näkymään tiivistetään sovelluksessa olevaa tietoa luettavaan muotoon. Graafien näyttämiseen valittiin Chart.js JavaScript-kirjasto.

Kuva 9: Kojelauta-näkymä dokumentointivaiheessa



Sovelluksen testaaminen suoritettiin yrityksen sisäisesti ja kohdistettiin sille käyttäjäryhmälle, joka sovellusta tulee käyttämään. Näin testauksen lisäksi saatiin tietoa käyttäjien toiveista sovelluksen toiminnallisuutta ja ulkonäköä koskien. Testaajille pidettiin koulutustilaisuus sovelluksen käytöstä, jonka jälkeen jälkeen sovellus otettiin käyttöön muiden järjestelmien rinnalle. Ylläpitoon tulevat työt kirjattiin uuteen järjestelmään ja jaettiin sen kautta asentajille.

5 YMPÄRISTÖ

Savon Tietokeskus Oy:n konesaliin perustettiin virtuaalipalvelin web-sovelluksen ylläpitoa varten. Palvelimelle asennettiin Windows Server 2016 käyttöjärjestelmä, Microsoft SQL Server 2017 sovelluksen tietokantaa varten sekä palvelimen tietoturvaohjelmisto F-Secure Server Security. Web-sovellusta ylläpidetään IIS web-palvelinohjelmiston avulla. Sovelluksen tietokanta sekä koko sovelluspalvelin on varmuuskopioitu konesaliin sekä off-site -varmistuspalvelimelle VeeAm-varmistusohjelmiston avulla.

5.1 Tietoturva

Tietoturva on kriittinen osa järjestelmää, jossa käsitellään yritysasiakkaiden tietoja. Ensimmäinen osa tietoturvaa alkaa Savon Tietokeskus Oy:n palomuurilta, josta on estetty yhteydet sovellukseen kaikkialta muualta, paitsi yrityksen sisäverkosta. Sovelluksen käyttäjät pystyvät käyttämään sovellusta työasemilta sekä mobiilaitteilta SSL VPN-yhteyden yli. Sovelluksen tietoliikenne on salattu käyttäen TLS 1.2 -salausprotokollaa.

Järjestelmään täytyy saada lähetettyä tikettejä myös asiakkaan verkosta. Asiakkaille kehitettiin C#-pohjainen Windows-sovellus, joka pystyy lähettämään tietoa erilliseen rajapintaan, joka on yhdistetty tietokantaan. Sovellus pystyy kirjoittamaan vain tiketti-tauluun ja se tarkastaa pyynnön aitouden käyttäen avainta, joka on määritetty sekä sovellukseen, että palvelimeen. Rajapinta on erillinen web-sovellus, josta on avattu tietty portti ulkoverkkoon. Tiketti-ohjelmisto asennetaan työasemille muiden ylläpito-ohjelmistojen asennuksen yhteydessä.

6 TULEVAISUUS

Web-sovellukseen on mahdollista avata rajapintoja muihin järjestelmiin. Esimerkkejä tästä ovat Visman toiminnanohjausjärjestelmä Nova tai Microsoftin Power BI. Suorat integraatiot muihin järjestelmiin edistävät sovelluksen kaupallistamista. Ennen rajapintojen suunnittelua järjestelmän tietoturva tulee tarkastaa uudelleen, ja julkaista se julkisen osoitteen taakse. Sovelluksessa säilytettävän arkaluontoisen tiedon takia tietoturvan tulee olla kehityksessä etusijalla.

Järjestelmää pystytään tarjoamaan muille tietotekniikka-alan yrityksille ohjelmistona tai pilvipalveluna. Ohjelmistoratkaisussa asiakasyritys hoitaa verkon, virtuaalipalvelimen, tietokannan ylläpidon. Savon Tietokeskus Oy toimittaa sovelluksen ja hoitaa sen asennuksen. Helpompi tapa myydä ja ylläpitää sovellusta on myydä se palveluna (SaaS). Sovellusta ylläpidetään Savon Tietokeskus Oy:n omassa konesalissa ja asiakasta laskutetaan järjestelmän käytöstä esimerkiksi kuukausihinnalla.

Sovellukseen on kehitteillä tikettiohjelmiston lisäksi Office365-autentikointia hyväksikäyttävä rajapinta. ASP.NET Coren identiteetti on mahdollista lisätä ulkoisiin kirjautumismenetelmiin, kuten Microsoftin Office365:n. Tämä helpottaisi tiketöintijärjestelmän käyttöönottoa uusien asiakkaiden ympäristöissä paljon. Office365-laajennusta käyttöönotettaessa täytyy ottaa huomioon yleinen tietoturvalaki GDPR, sillä tietokantaan joudutaan tallentamaan yksityishenkilöitä yksilöivää tietoa, joka yhdistää Office365-tilin käyttäjään.

Sovellukseen suunnitellaan myös työntekijöiden työajanseurantaa. Näin saataisiin yhdistettyä mahdollisimman monta käytössä olevaa järjestelmää. Syötettyjen työaikojen ja asiakkailta laskutettavien töiden perusteella työntekijöille voidaan asettaa tavoitteita viikko ja kuukausitasolla. Näinä tavoitteita voidaan seurata järjestelmän kautta ja pyrkiä parantamaan työntekijöiden tehokkuutta. Työaikojen, tehtyjen töiden sekä tavoitteiden lisääminen järjestelmään edesauttaa työntekijäkohtaisten raporttien luomista. Jokaisesta asentajasta saadun tiedon perusteella voidaan optimoida työntekijöiden tehokkuutta vielä pidemmälle.

7 POHDINTA

Haastavin osuus sovelluksen kehityksessä oli suunnittelu. Suunnitteluvaiheessa on erittäin vaikea ottaa huomioon kaikki ongelma ja kehityskohdat. Sovelluksen kehityksen alkuvaiheessa jouduttiin usein palaamaan muokkaamaan aiemmin tehtyä. Kehityksen edetessä aiemmin ohjelmoidun koodin muokkaamisen määrä väheni. Uuden kehityksessä ja suunnittelussa osattiin ottaa useampia haastavuksia huomioon ja tämä nopeutti sovelluksen kehitystä. Alkuvaiheen muokkaus vahvisti sovelluksen pohjaa, jonka päälle erilaiset ominaisuudet rakennettiin ja samalla varmisti toimivamman kokonaisuuden myös kehitysvaiheessa.

Sovelluksen suunnittelu ja kehitysvaiheessa tiedettiin, että asiakkaiden ympäristöjen dokumentaatiot on saatava syötettyä käyttöliittymän kautta järjestelmään sekä niiden tulee olla luettavassa muodossa siellä myös mobiililaitteilla. Sovelluksessa käytettiin relationaalista tietokantaa, joten dokumentaatiot olivat saatava taulukkomuotoon, mikä osoittautui haastavaksi. Savon Tietokeskus Oy:n ylläpidossa olevat yritykset ovat pienimmillään yhden työntekijän ja suurimmillaan 200 työntekijää työllistäviä yrityksiä. Erikokoisilla yrityksillä voi olla hyvinkin erilaisia ympäristöjä, joille piti löytää yksi universaali pohja. Ympäristöjen dokumentaatioiden muokkaaminen taulukkomuotoon, dokumentaatioiden kerääminen Excel-muotoon ja kerätyn tiedon siirtäminen tietokantaan sovelluksen käytettäväksi onnistui erinomaisesti. Asiakkaiden ympäristöistä onnistuttiin kartoittamaan tärkeimmät tiedot ja tiedot onnistuttiin muuttamaan taulukon sarakkeiksi.

Sovelluksen suunnitteluvaiheessa tiedettiin, että tietokannassa ja järjestelmässä tullaan säilyttämään asiakasyritysten arkaluontoista dataa kuten salasanoja, sähköpostiosoitteita ja VPN-yhteyksien tunnuksia ja julkisia IP-osoitteita. ASP.Net Core:n tarjoamat työkalut kirjautumiseen helpottivat tietoturvallisen järjestelmän luomista. Tietoturvaa edistettiin rajaamalla sovellus vain Savon Tietokeskus Oy:n käyttöön vain yrityksen sisäverkosta. Rajauksesta huolimatta sovellusta voi käyttää myös mobiililaitteilla SSLVPN-yhteyden avulla. Palomuuriestojen lisäksi sovelluksen kaikki liikenne kulkee 443 portin kautta ja on salattu SSL-tekniikalla. Tietoturva on hyvin kriittinen osa järjestelmää ja sen toteutuksessa onnistuttiin hyvin.

Suurin onnistuminen sovelluksessa tuli julki testausvaiheessa. Kun työntekijät huomasivat, että eri työntekijät voivat kirjata töitä saman tiketin alle, sovelluksesta löydettiin uusi ominaisuus yrityksen käyttöön. Sovelluksella pystytään aikatauluttamaan projektit ja pilkkomaan ne pienempiin osiin eri työntekijöille. Tämä helpottaa projektien aikatauluttamista ja yksinkertaistaa projektien toteutumisen kulkua.

Sovelluksen raportointi-ominaisuus jouduttiin kiirehtimään laskutuksen aikataulun vuoksi ja tämän takia sen suunnittelussa ja toteuktuksessa ilmeni virheitä. Raporttia otettaessa raportista tallennettiin vain tietokantakysely ja näin ollen raporttien tieto voi muuttua, jos tietoja muutetaan jälkeen-

päin. Ongelma saatiin korjattua tallentamalla raportoitava data omaan tauluun tietokannassa ja tallentamaan raportti PDF-muodossa palvelimelle. Haasteen tuomat ongelmat sekä ratkaisu korostivat suunnittelun ja aikataulutuksen tärkeyttä suurempaa kokonaisuutta kehitettäessä.

LÄHTEET JA TUOTETUT AINEISTOT

ROTH, Daniel, ANDERSON, Rick, LUTTIN, Shaun 2018. Introduction to ASP.NET Core. Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-2.2

SMITH, Steve 2018. Overview of ASP.NET Core MVC. Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-2.2

MARTIN, Robert C. 2002. Agile Software Development, Principles, Patterns, and Practices. Pearson.

SMITH, Steve, ADDIE, Scott, LATHAM, Luke 2018. Dependency injection in ASP.NET Core. Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-2.2

SMITH, Steve, LATHAM, Luke 2017. Views in ASP.NET Core MVC. Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-2.2

SMITH, Steve, ADDIE, Scott 2017. Handle requests with controllers in ASP.NET Core MVC. Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/actions?view=aspnetcore-2.2

MILLER, Rowan, VEGA, Diego, WENZEL, Maira, VICKERS, Arthur, HALL, Steven, THOMAS, Nico, MARKOWSKI, Maurycy, BROWNE, David 2016. Loading related data. Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/ef/core/querying/related-data

WENZEL, Maira, WAGNER, Bill, LATHAM, Luke, HOAG, Steve 2017. Language Integrated Query (LINQ). Microsoft. Verkkodokumentti. Saatavissa: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/